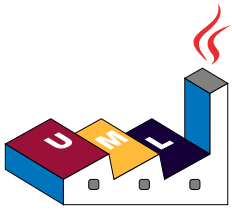
PlantUML を使った UML の描き方



言語リファレンスガイド

(2016 年 9 月 29 日 7:46)

PlantUML は以下のような図を手早く書くためのオープンソースプロジェクトです。

- シーケンス図,
- ユースケース図,
- クラス図,
- アクティビティ図,
- コンポーネント図,
- 状態遷移図,
- オブジェクト図.

これらの図はシンプルかつ直感的な言語によって定義されています。

1 シーケンス図

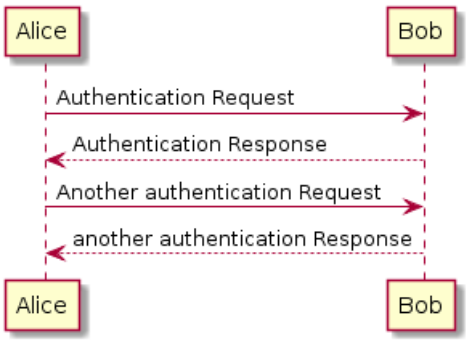
1.1 基本的な例

シーケンス”->” を、2 つの分類子間のメッセージを描画するために使います。分類子を、明示的に宣言する必要はありません。

点線の矢印を使う場合は、”-->” とします。

また、”<-” や”<--” を使うこともできます。これらは図を変更することなく、可読性を高めることができます。ただし、以上の方法はシーケンス図だけに当てはまります。ほかの種類の図には当てはまりません。

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

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



1.2 コメント

シングルクォート ' から始まるものはコメントとして扱われます。

また、/' で初まり'/で終わる複数行にまたがるコメントも書けます。

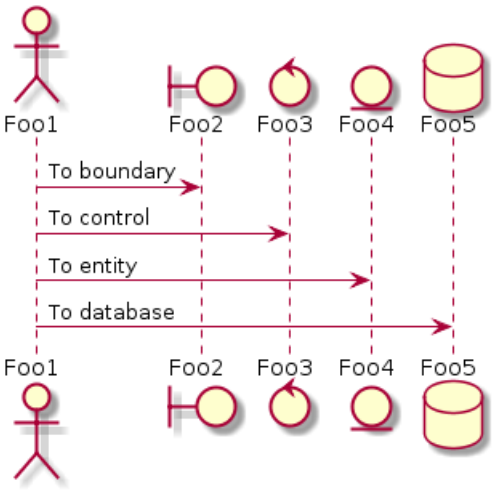
1.3 分類子の宣言

participant キーワードを使って、分類子の並び順を変えることができます。

分類子の宣言に別のキーワードを使用することも可能です：

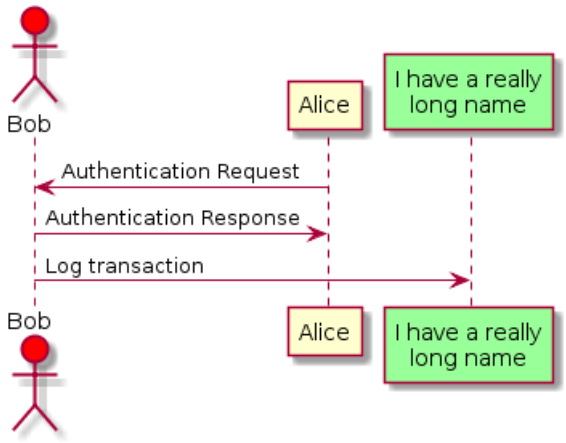
- actor
- boundary
- control
- entity
- database

```
@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
```



as キーワードを使って分類子の名前を変更することができます。
アクターや分類子の背景色を、HTML コードや色名を使って変更することもできます。

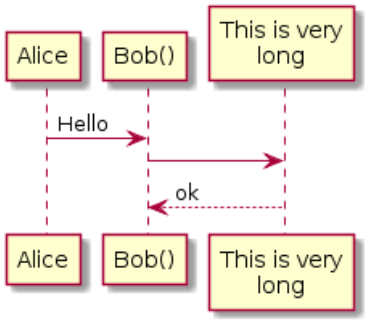
```
@startuml
actor Bob #red
' The only difference between actor
'and participant is the drawing
participant Alice
participant "I have a really\nlong name" as L #99FF99
/' You can also declare:
participant L as "I have a really\nlong name"  #99FF99
'/

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



1.4 分類子名にアルファベット以外を使う

分類子を定義するときに引用符を使用することができます。そして、分類子にエイリアスを与えるために as キーワードを使用することができます。

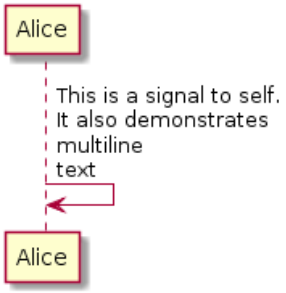
```
@startuml
Alice -> "Bob()" : Hello
"Bob()" -> "This is very\nlong" as Long
' You can also declare:
' "Bob()" -> Long as "This is very\nlong"
Long --> "Bob()" : ok
@enduml
```



1.5 自分自身へのメッセージ

分類子は自分自身へメッセージを送信できます。
\n を使用して、複数行のテキストを扱えます。

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

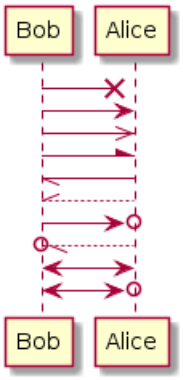


1.6 矢印の見た目を変える

矢印の見た目をいくつかの方法によって変更できます。

- メッセージの消失を示す最後の x を追加
- \ や / を < や > の代わりに使うと矢印の先端が上側だけまたは下側だけになります。
- 矢印の先端を繰り返す (たとえば >> や //) と、矢印の先端が細くなります。
- -- を - の代わりに使うと、矢印が点線になります。
- 矢じりに最後の"O" を追加
- 双方向の矢印を使用する

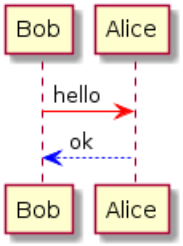
```
@startuml
Bob ->x Alice
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \\- Alice
Bob //-- Alice

Bob ->o Alice
Bob o\\-- Alice

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



1.7 矢印の色を替える

以下の表記を使って、個々の矢印の色を変えることができます。

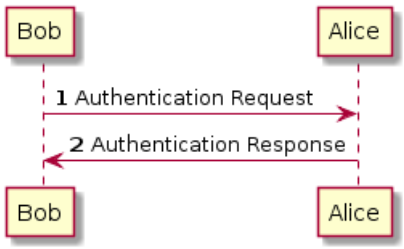
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.8 メッセージシーケンスの番号付け

メッセージへ自動で番号を振るために、autonumber キーワードを使います。

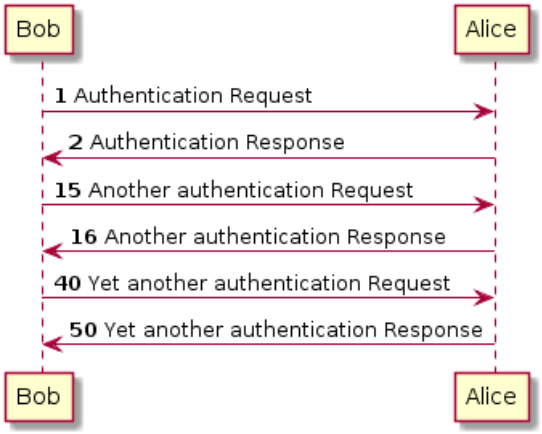
```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response
@enduml
```



autonumber ' 開始' で開始番号を、また、autonumber ' 開始' ' 増分' で増分も指定することができます。

```
@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
```

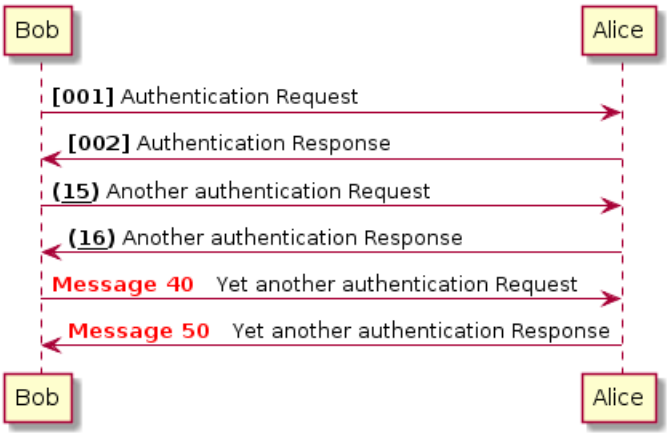


二重引用符で囲って番号の書式を指定することができます。

その書式指定は Java の DecimalFormat 方式で行う（'0' は桁を表し, '#' は存在しない場合は 0 で埋める桁を意味する）。

HTML タグを書式に使うこともできます。

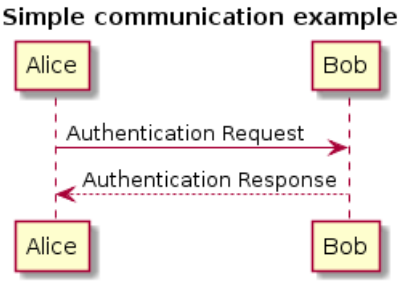
```
@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

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
```



autonumber stop と autonumber resume ' 増分' ' 書式' を自動採番の一時停止と再開にそれぞれを使用することができます。

```
@startuml
autonumber 10 10 "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber stop
Bob -> Alice : dummy
```

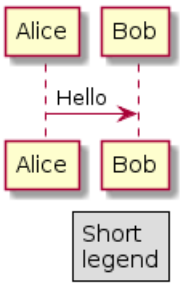
```
autonumber resume "<font color=red><b>Message 0  "
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

autonumber stop
Bob -> Alice : dummy

autonumber resume 1 "<font color=blue><b>Message 0  "
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response
@enduml
```



1.9 タイトル

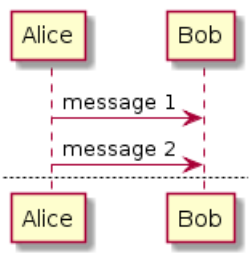
タイトルの指定に `title` キーワードを使います。

```
@startuml

title Simple communication example

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

@enduml
```



There is also a `caption` keyword to put a caption under the diagram.

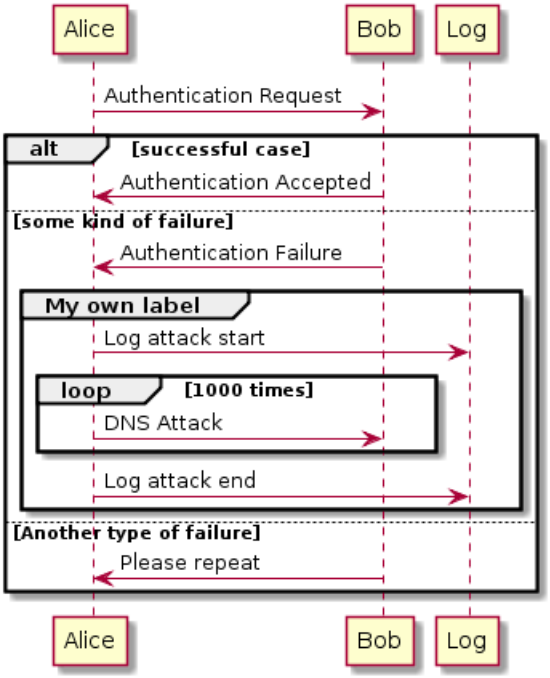
```
@startuml

caption figure 1

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

@enduml
```



1.10 図の説明文

legend と end legend は説明文を記入するために使用されるキーワードです。
必要に応じて left、right、center によって説明文の位置合わせをすることができます。

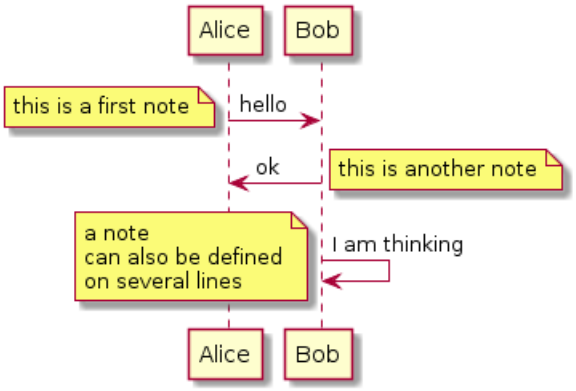
```
@startuml

Alice -> Bob : Hello
legend right
Short
legend
endlegend

@enduml
```



1.11 図の分割

図を複数の画像に分けるために newpage キーワードを使います。
新しいページのタイトルを newpage キーワードの直後に書くことができます。
これは、複数ページにわたる長い図を書くときに便利な機能です。

```
@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
```



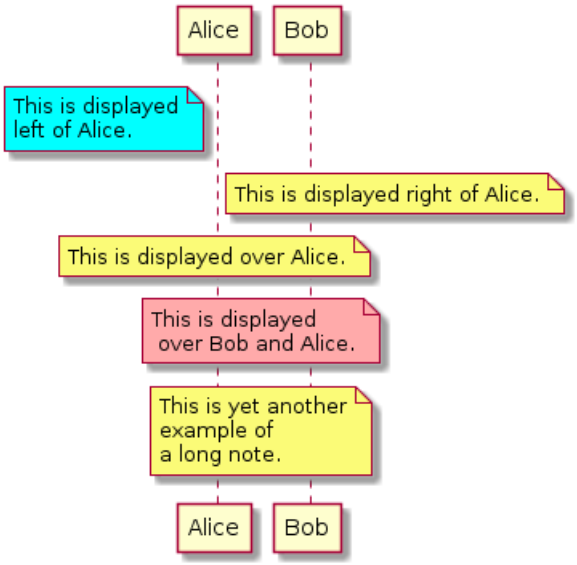
1.12 メッセージのグループ化

次のキーワードを使えば、メッセージをまとめてグループ化できます。

- alt/else
- opt
- loop
- par
- break
- critical
- group 表示するテキスト

ヘッダ部分に文字列を追加することが可能です。(group を除く)
グループの終了にはキーワード end を使用します。
注：グループはネスト可能です。

```
@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
```

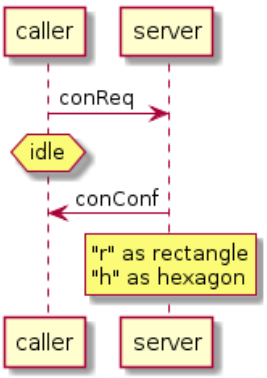


1.13 メッセージの注釈

メッセージのすぐ後ろにキーワード `note left` または `note right` を使用しメッセージの注釈をつけることが可能です。

`end note` キーワードを使って、複数行の注釈を付けることができます。

```
@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 その他の注釈

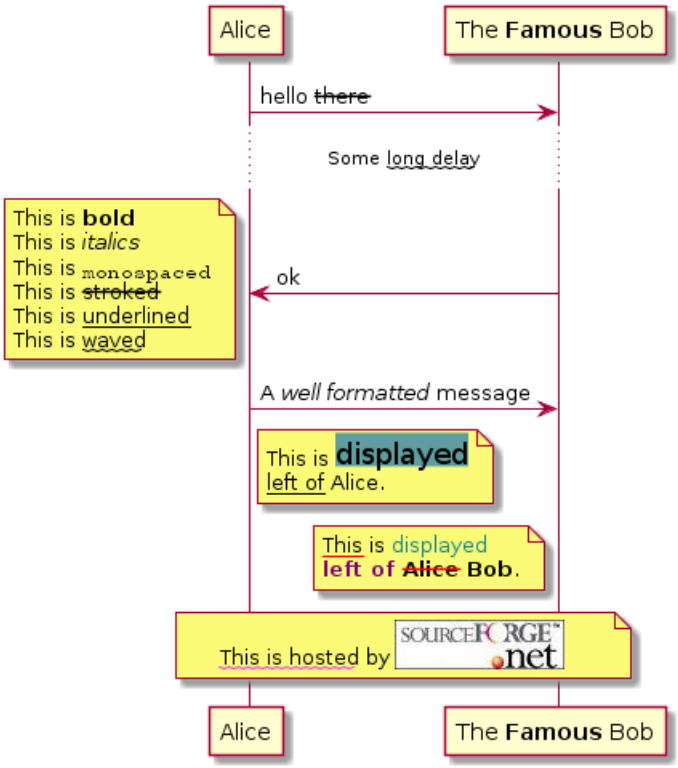
分類子との相対位置を指定して注釈を付けるには、次のものを使います:

注釈を目立たせるために、背景色を変えることができます。

また、`end note` キーワードを使って複数行の注釈を付けることができます。

```
@startuml
participant Alice
participant Bob
note left of Alice #aqua
```

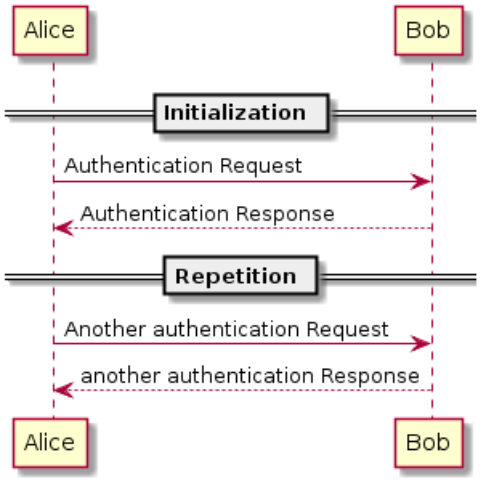
```
This is displayed
left of Alice.
end note

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 ノートの形を変える。

hnote と rnote のキーワードを使ってノートの形を変更できます。

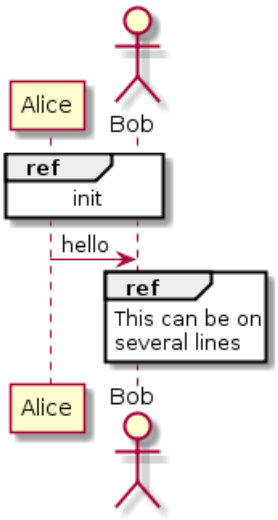
```
@startuml
caller -> server : conReq
hnote over caller : idle
caller <- server : conConf
rnote over server
"r" as rectangle
"h" as hexagon
endrnote
@enduml
```



1.16 Creole と HTML

PlantUML では creole フォーマットを使うこともできます。

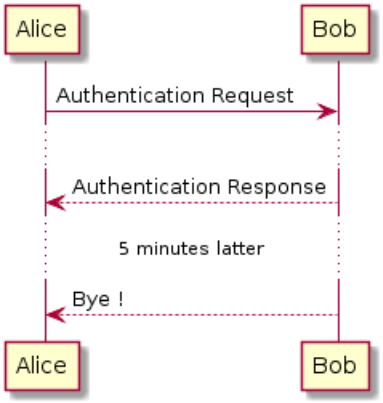
```
@startuml
participant Alice
participant "The **Famous** Bob" as Bob

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 境界線

== を使って、図を論理的なステップに分けることも出来ます。

```
@startuml

== Initialization ==

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

== Repetition ==

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

@enduml
```

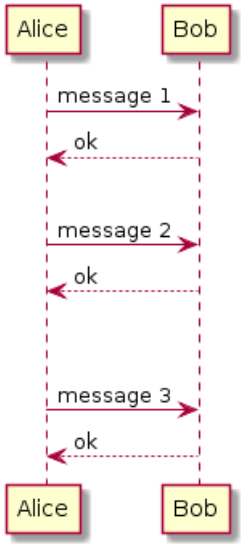


1.18 リファレンス

キーワード ref over を使用して、図中にリファレンスを挿入できます。

```
@startuml
participant Alice
actor Bob

ref over Alice, Bob : init

Alice -> Bob : hello

ref over Bob
This can be on
several lines
```

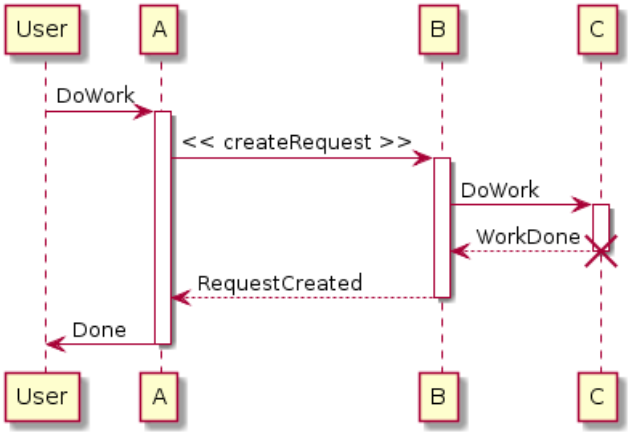
```
end ref
@enduml
```



1.19 遅延

処理の遅延を表すために ... が使えます。また、作成した遅延にコメントを付けることもできます。

```
@startuml

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

@enduml
```

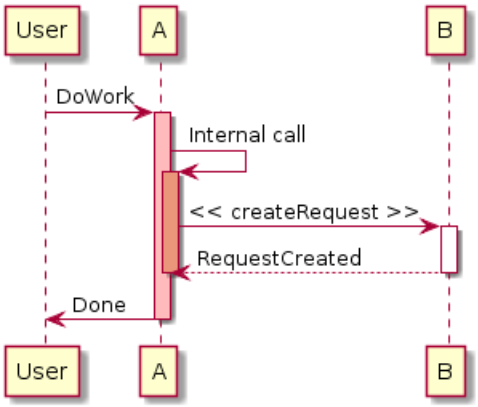


1.20 間隔

図の間隔を調整するために、記号 ||| を使用することができます。
さらにピクセル数を指定することもできます。

```
@startuml

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

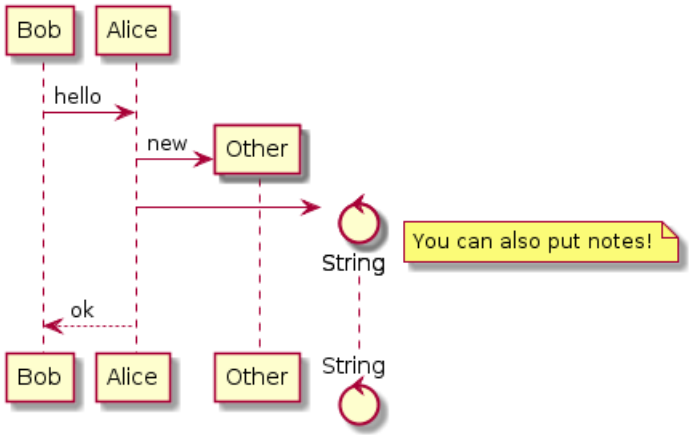
```
||45||
Alice -> Bob: message 3
Bob --> Alice: ok

@enduml
```



1.21 ライフラインの活性化と破壊

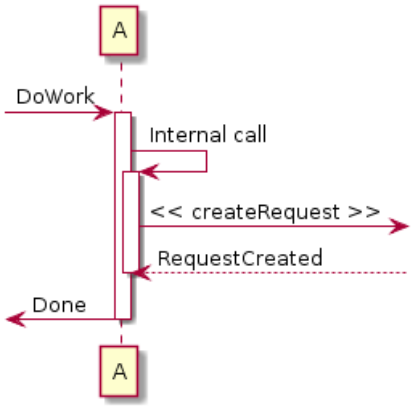
activate と deactivate を使って分類子の活性化を表します。
分類子の活性化はライフラインで表されます。
activate と deactivate は直前のメッセージに適用されます。
destroy は分類子のライフラインが終わったことを表します。

```
@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
```



ライフラインはネスト (入れ子に) することができ、色をつけることもできます。

```
@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 分類子の生成

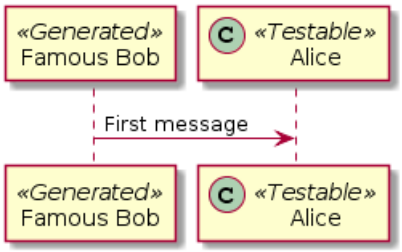
create キーワードを、オブジェクトが最初のメッセージを受信する直前に置くことにより、このメッセージがオブジェクトを新しく生成していることを強調して表現できます。

```
@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 インとアウトのメッセージ

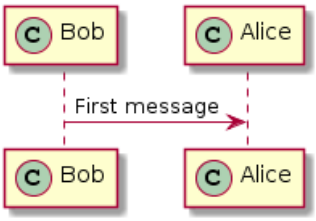
図の一部だけにフォーカスを当てたい場合には、インまたはアウトのメッセージを使えます。
左角括弧”[” を使って図の左端、右角括弧”]” を使って図の右側を表せます。

```
@startuml
[-> A: DoWork

activate A

A -> A: Internal call
activate A

A ->] : << createRequest >>

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



また、次の書き方も使えます：

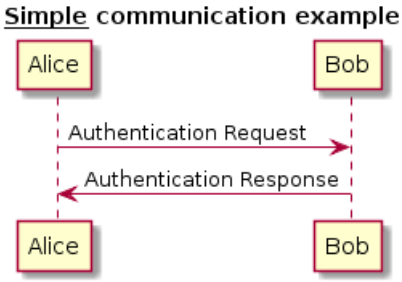
```
@startuml
[-> Bob
[o-> Bob
[o->o Bob
[x-> Bob

[<- Bob
[x<- Bob

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

Bob <-]
Bob x<-]
@enduml
```



1.24 ステレオタイプとスポット

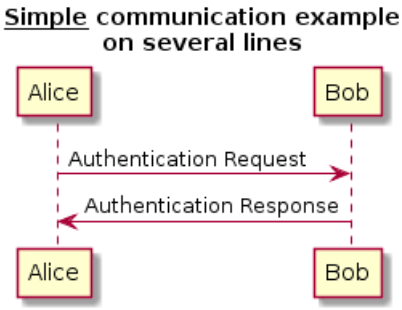
<< と >> を使い分類子にステレオタイプをつけることができます。
(X,color) と記述することによりステレオタイプに色付きの文字と円のアイコンをつけることができます。

```
@startuml

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

Bob->Alice: First message

@enduml
```



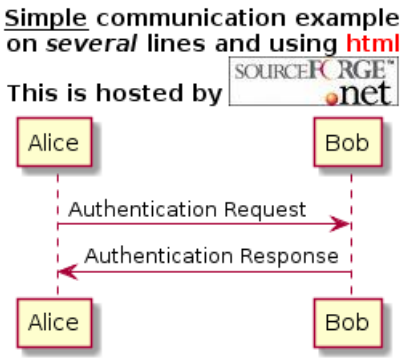
デフォルトでは *guillemet* キャラクターはステレオタイプを表示するために使用されます。スキンパラメータ guillemet を使用してこの動作を変更することができます：

```
@startuml

skinparam guillemet false
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) >>
```

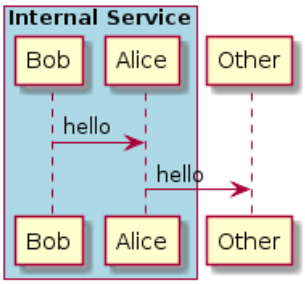
```
Bob->Alice: First message

@enduml
```



1.25 タイトルについての詳細

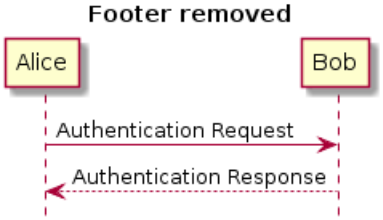
タイトルには creole フォーマットが使用できます。

```
@startuml

title __Simple__ **communication** example

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

@enduml
```



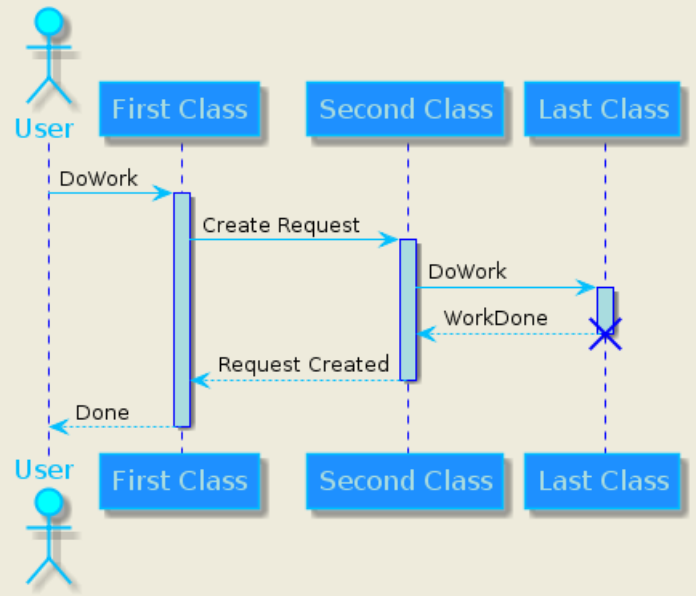
タイトルの説明では \n で改行を表せます。

```
@startuml

title __Simple__ communication example\non several lines

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

@enduml
```



また、`title` キーワードと `end title` キーワードを使うことにより、タイトルを複数行にわたって記述できます。

```
@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
```



1.26 分類子の囲み

box と end box コマンドを使い、分類子のまわりにボックスを描くことができます。
タイトルや背景色を box キーワードに続けて任意で追加できます。

```
@startuml

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

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

@enduml
```

1.27 フッターの除去

図のフッターを削除するには hide footbox キーワードを使います。

```
@startuml

hide footbox
title Footer removed

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

@enduml
```

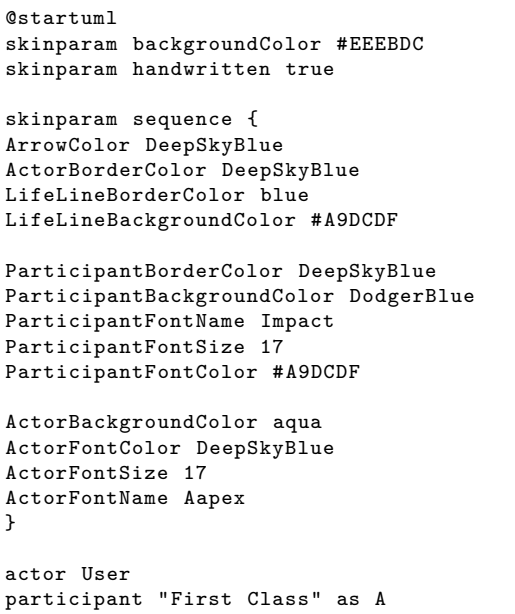
1.28 Skinparam コマンド

図の色やフォントを変えるには skinparam コマンドを使います。
コマンドを使用できるのは：

- 他のコマンドと同じように図の定義の中、
- インクルードしたファイルの中、
- またはコマンドラインで指定された設定ファイルや Ant タスクの中です。

次の例のように他のパラメータを変えることもできます。

```
@startuml
skinparam sequenceArrowThickness 2
skinparam roundcorner 20
skinparam maxmessagesize 60
skinparam sequenceParticipant underline

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
```



```
@startuml
skinparam backgroundColor #EEEBDC
skinparam handwritten true

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

ParticipantBorderColor DeepSkyBlue
ParticipantBackgroundColor DodgerBlue
ParticipantFontName Impact
ParticipantFontSize 17
ParticipantFontColor #A9DCDF

ActorBackgroundColor aqua
ActorFontColor DeepSkyBlue
ActorFontSize 17
ActorFontName Aapex
}

actor User
participant "First Class" as A
```

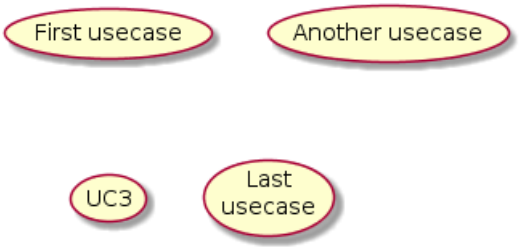
```
participant "Second Class" as B
participant "Last Class" as C

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　ユースケース図

2.1　ユースケース

ユースケースは丸括弧で囲んで使います (丸括弧の対は楕円に似ているからです)。
　usecase キーワードを使ってユースケースを定義することもできます。as キーワードを使ってエイリアスを定義することもできます。このエイリアスはあとで、ユースケースの関係を定義するために使います。

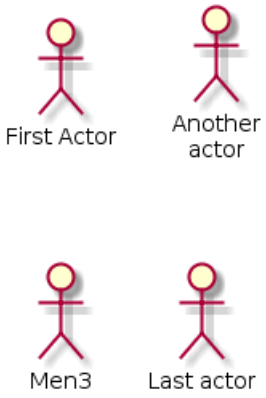
```
@startuml

(First usecase)
(Another usecase) as (UC2)
usecase UC3
usecase (Last\nusecase) as UC4

@enduml
```



2.2　アクター

アクターは 2 つのコロンで囲まれます。
　actor キーワードを使ってアクターを定義することもできます。as キーワードを使ってエイリアスを定義することもできます。このエイリアスはあとで、ユースケースの関係を定義するために使います。
　後から説明しますが、アクターの定義は必須ではありません。

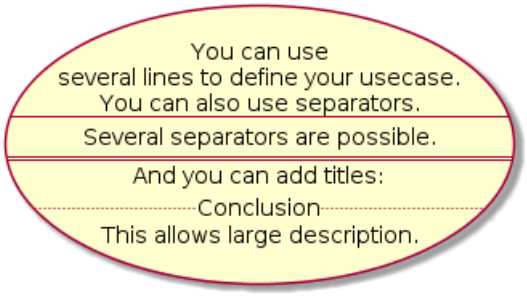
```
@startuml

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

@enduml
```



2.3　ユースケースの説明

クオート記号を使うことにより、複数行にわたる説明を記述できます。
　また、次の区切り記号を使用できます：-- .. == __。区切り記号の中にはタイトルを記入できます。

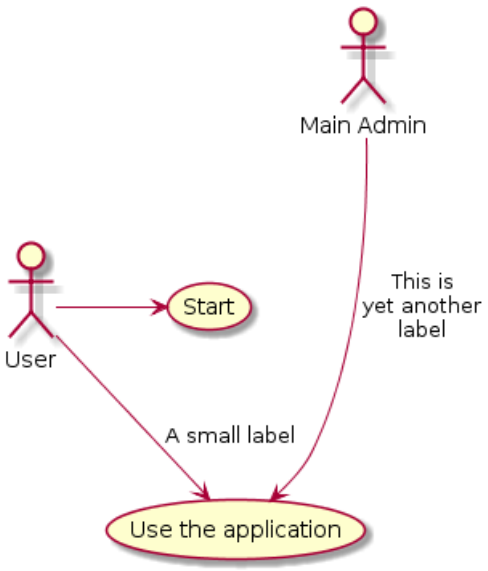
```
@startuml

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 簡単な例

アクターとユースケースを繋げるには --> 矢印を使います。
　矢印に使うハイフン "-" の数を増やすと矢印を長くできます。矢印の定義に":" を使うことにより矢印にラベルをつけることができます。
　以下の例では *User* は定義なしにアクターとして使われています。

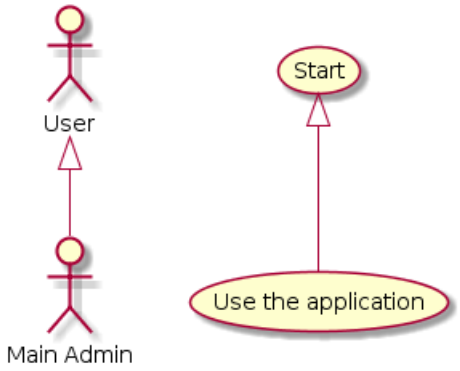
```
@startuml

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

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

@enduml
```



2.5 継承

もしアクターやユースケースが継承をする場合には、<|-- 記号を使います（これが  になります）。

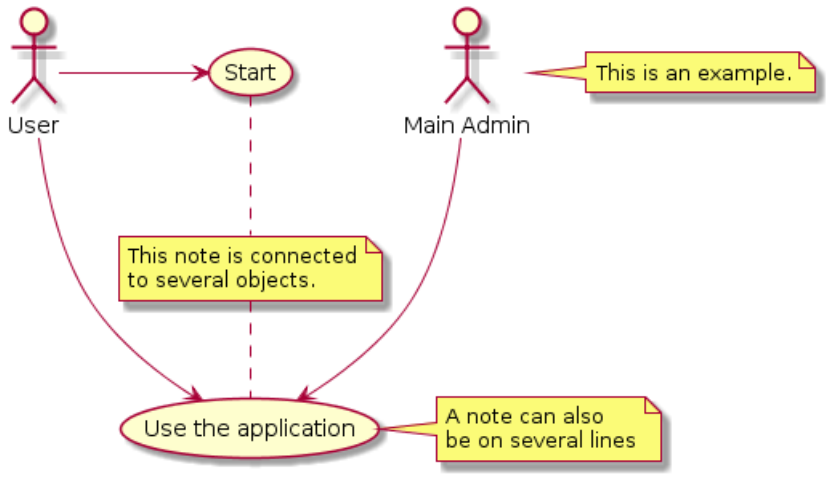
```
@startuml
:Main Admin: as Admin
(Use the application) as (Use)

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

@enduml
```



2.6 ノートの使用方法

オブジェクトに関連のあるノートを作成するには note left of 、note right of 、note top of 、note bottom of キーワードを使います。
　または note キーワードを使ってノートを作成し、.. 記号を使ってオブジェクトに紐づけることができます。

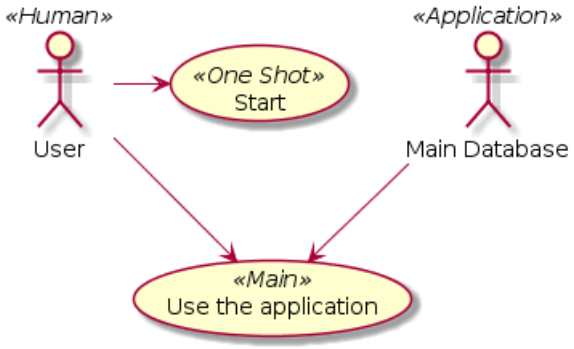
```
@startuml
:Main Admin: as Admin
(Use the application) as (Use)

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 ステレオタイプ

” <<” と” >> ” を使い、アクターとユースケースを定義中にステレオタイプを追加できます。

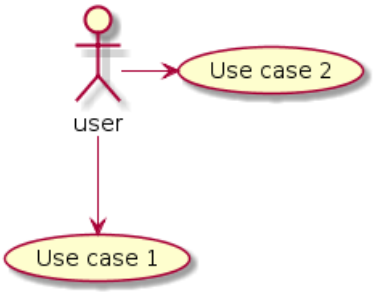
```
@startuml
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

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

MySql --> (Use)

@enduml
```



2.8 矢印の方向を変えるには

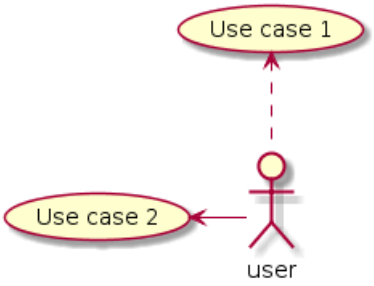
デフォルトでは、クラス間の線は 2 個のハイフン -- で表され、縦方向につながります。横方向の線を描くには以下のようにハイフン 1 つかドット 1 つを書きます。

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



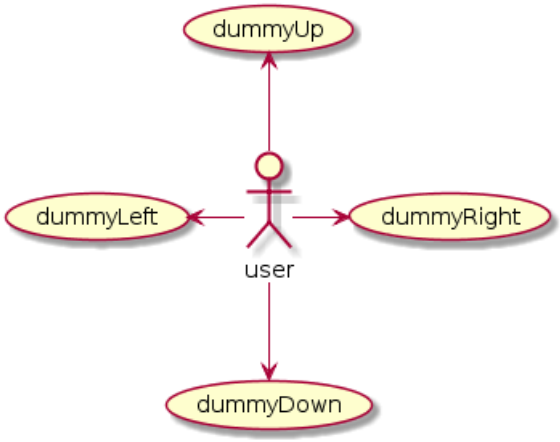
線を反対にすることでも方向を変えることができます。

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



矢印の内側に left、right、up、down を書くことによっても線の方向を変えられます。

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



例えば、-down- ではなく -d- など、各方向の頭文字、または頭 2 文字（-do- ）だけ使って矢印を短くすることも出来ます。

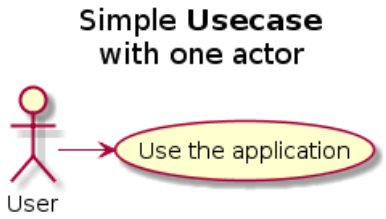
ただし、この機能の使いすぎには注意しましょう。ほとんどの場合、特別なことをしなくても *Graphviz* がその場にあった表示を選びます。

2.9 図にタイトルをつける

タイトルを付けるにはキーワード title を使います。

シーケンス図のように title と end title キーワードを使って、長いタイトルもつけられます。

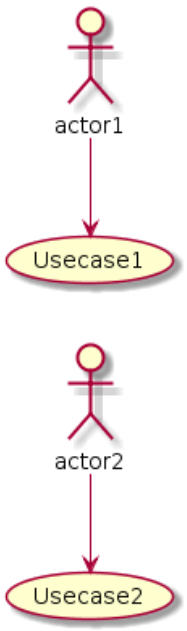
```
@startuml
title Simple <b>Usecase</b>\nwith one actor

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

@enduml
```



2.10 図を分割する

newpage キーワードは、いくつかのページや画像に図を分割します。

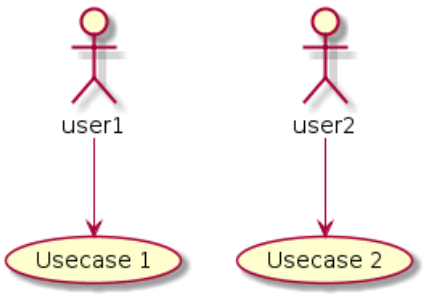
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



2.11 左から右に描画する

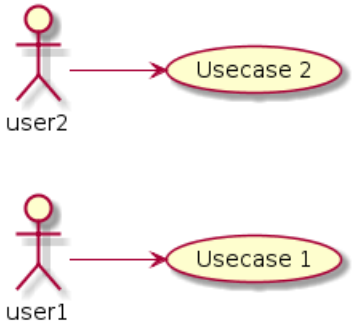
デフォルトの作図方向は top to bottom となっています。

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

@enduml
```



作図方向を **left to right** に変更するには `left to right direction` コマンドを使います。

```
@startuml

left to right direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)

@enduml
```



2.12 スキン設定（Skinparam）

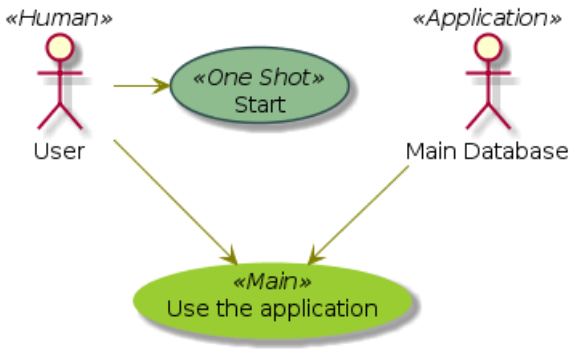
ダイアグラムの色やフォントを変更するには `skinparam` コマンドを使用します。
このコマンドは以下の場面で使用できます。

- ・ダイアグラム定義内で他のコマンドを同様に。
- ・インクルードされたファイル内。
- ・設定ファイルのコマンドライン内や ANT タスク内。

個別のステレオタイプ付きアクターやユースケースにそれぞれ色やフォントを定義することができます。

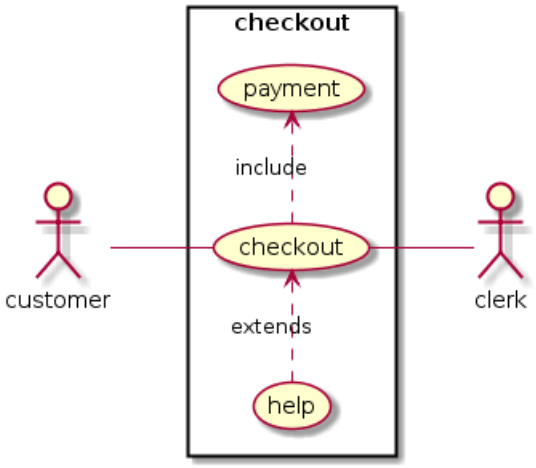
```
@startuml
skinparam handwritten true

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 完全な例

```
@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 クラス図

3.1 クラス間の関係

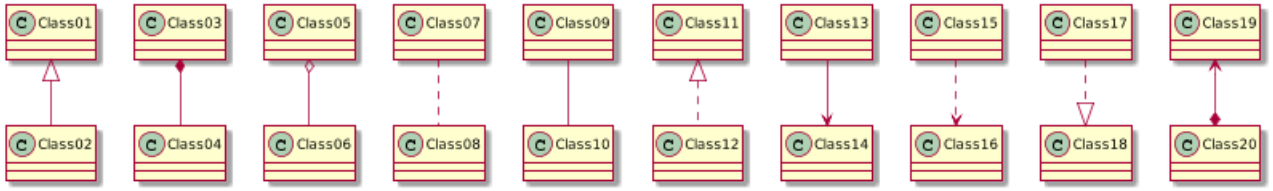
クラス間の関係は次の記号を使用して定義されています:

継承	<|--	
合成	*--	
集約	o--	

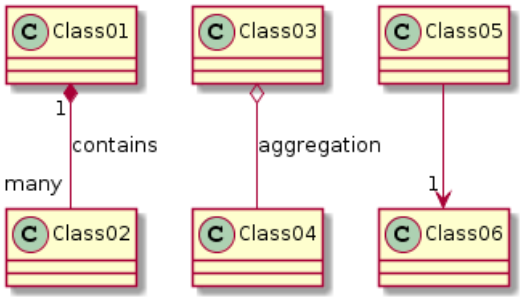
”--” を”..” に置き換えると点線にできます。

これらのルールを知ることで、以下の図面を描くことができます:

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




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

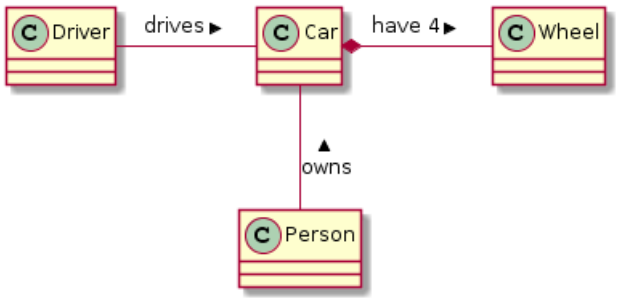


3.2 関連のラベル

”:” にテキストを続けることによって、関連へラベルを追加することが可能です。

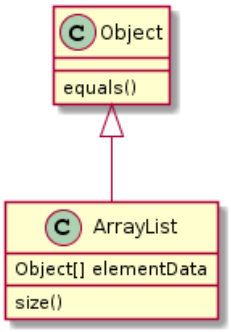
多重度を示す為に関連のそれぞれの側にダブルクォーテーション"" を使うことができます。

```
@startuml

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

Class03 o-- Class04 : aggregation

Class05 --> "1" Class06

@enduml
```



ラベルの最初または最後に < か > を使って、他のオブジェクトへの関連を示す矢印を追加できます。

```
@startuml
class Car

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

@enduml
```

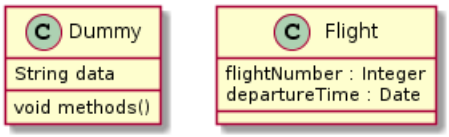


3.3 メソッドの追加

":" に続けてフィールド名やメソッド名を記述すると、フィールドやメソッドを宣言できます。

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

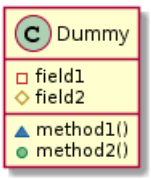
```
@startuml
Object <|-- ArrayList

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

@enduml
```



波括弧 { を使って、フィールドやメソッドをくくることもできます。

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

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

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

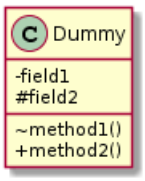


3.4 可視性の定義

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

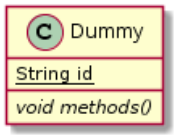
-	□	■	private （クラス外からは非可視）
#	◇	◆	protected （サブクラスからも可視）
~	△	▲	package private （パッケージ外からは非可視）
+	○	●	public （どこからでも可視）

```
@startuml

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

@enduml
```



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

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

@enduml
```

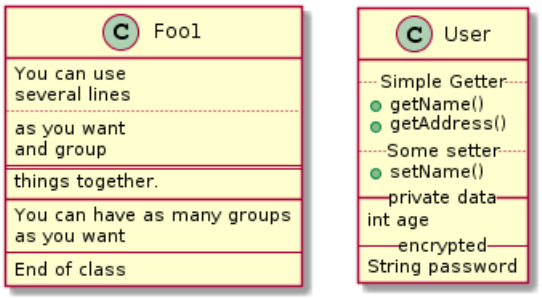


3.5 Abstract と Static

静的または抽象的なメソッドまたはフィールドは {static} または {abstract} 修飾子を使用することで定義することができます。

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

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

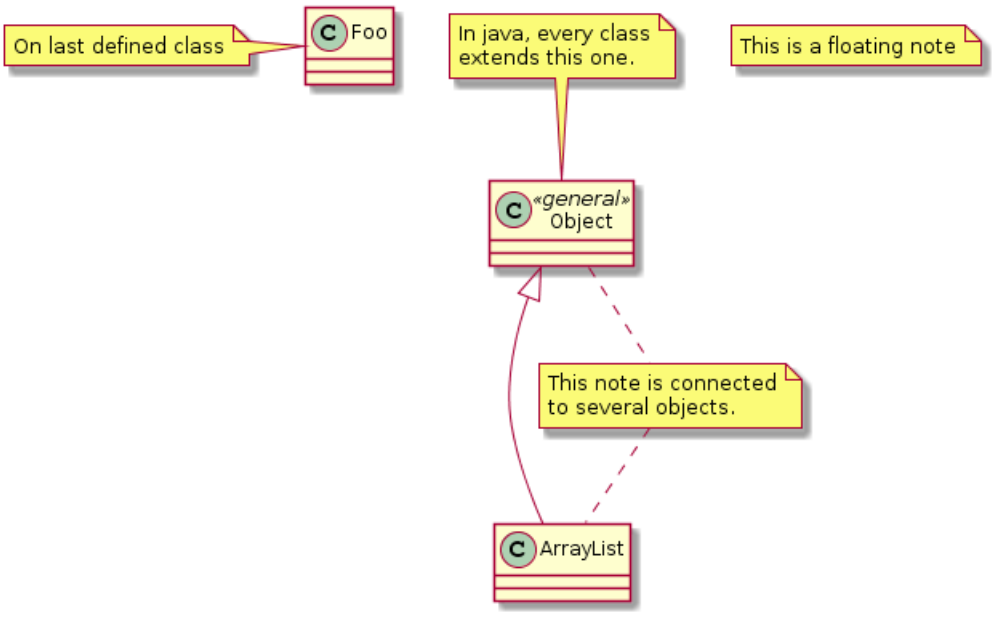


3.6 Advanced class body

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

You can also use titles within the separators:

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

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

@enduml
```



3.7 Notes and stereotypes

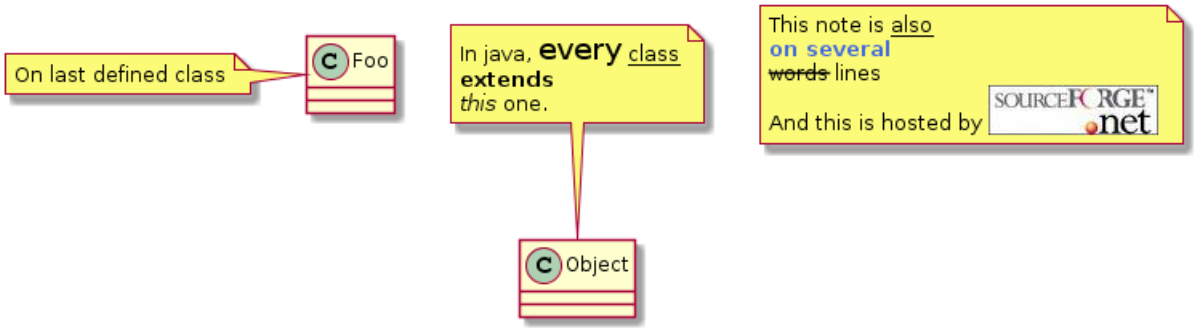
Stereotypes are defined with the `class` keyword, " `<<` " and " `>>` ".

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

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

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

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

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

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

class Foo
note left: On last defined class

@enduml
```



3.8 More on notes

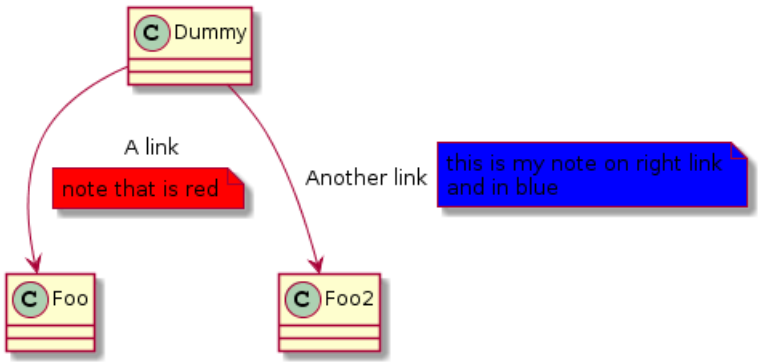
It is also possible to use few html tags like :

- `<b>`
- `<u>`
- `<i>`
- `<s>`, `<del>`, `<strike>`
- `<font color="#AAAAAA">` or `<font color="colorName">`
- `<color:#AAAAAA>` or `<color:colorName>`
- `<size:nn>` to change font size
- `<img src="file">` or `<img:file>` : the file must be accessible by the filesystem

You can also have a note on several lines.

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

```
@startuml

class Foo
note left: On last defined class

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

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

@enduml
```

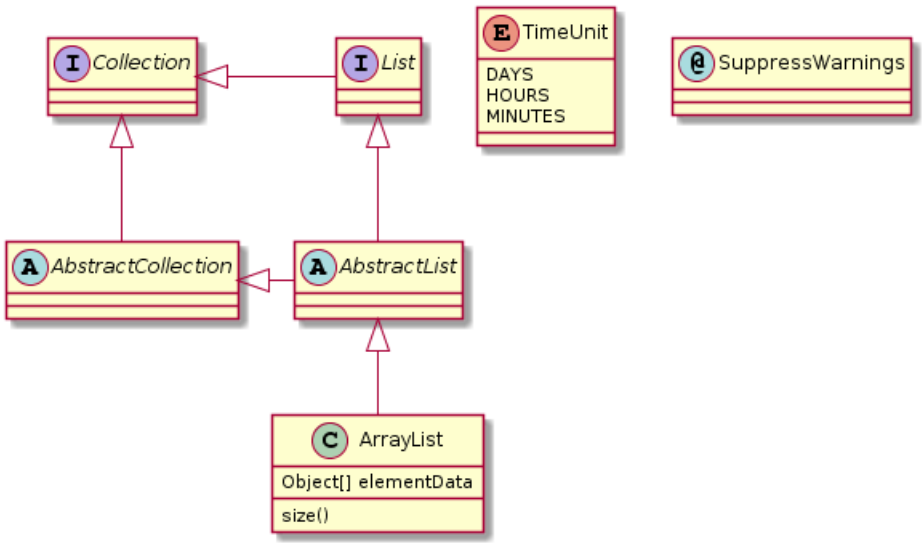


3.9 Note on links

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

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

```
@startuml

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

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

@enduml
```



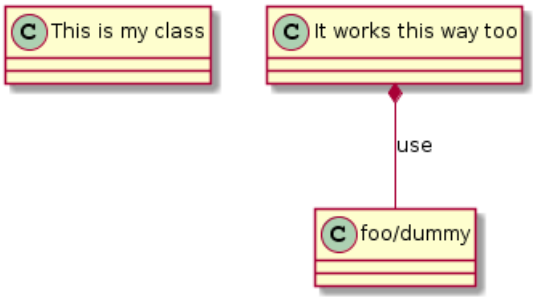
3.10 Abstract class and interface

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

The class will be printed in *italic*.

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

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

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

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

class ArrayList {
Object[] elementData
size()
}

enum TimeUnit {
DAYS
HOURS
MINUTES
}

annotation SuppressWarnings

@enduml
```

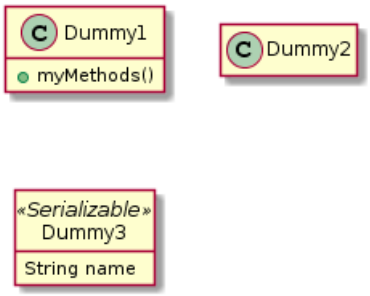


3.11 Using non-letters

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

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

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

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



3.12 Hide attributes, methods...

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

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

Instead of `empty members`, you can use:

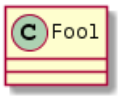
- `empty fields` or `empty attributes` for empty fields,
- `empty methods` for empty methods,
- `fields` or `attributes` which will hide fields, even if they are described,
- `methods` which will hide methods, even if they are described,
- `members` which will hide fields and methods, even if they are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

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

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

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

```
@startuml

class Dummy1 {
+myMethods()
}

class Dummy2 {
+hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

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

@enduml
```

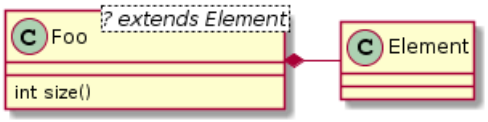


3.13 Hide classes

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

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

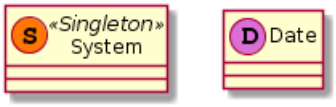
```
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```



3.14 Use generics

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

```
@startuml

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

@enduml
```

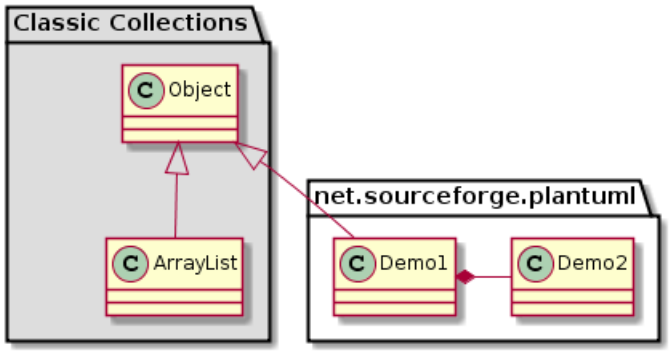


3.15 Specific Spot

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

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

```
@startuml

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

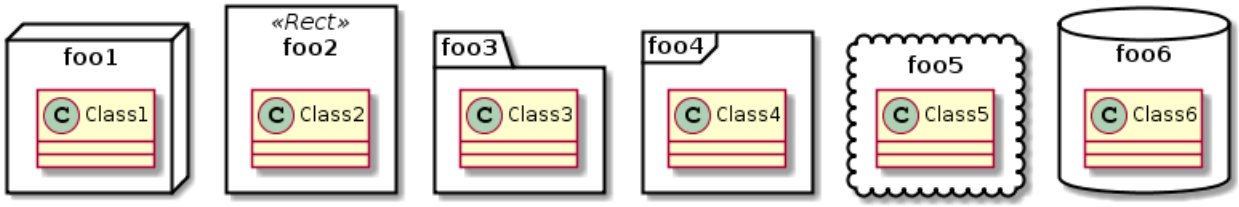


3.16 Packages

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

Note that package definitions can be nested.

```
@startuml

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

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

@enduml
```

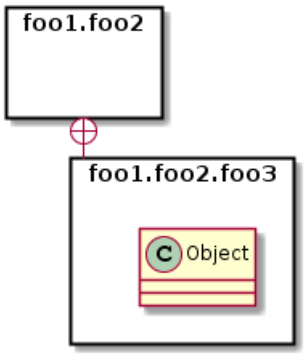


3.17 Packages style

There are different styles available for packages.

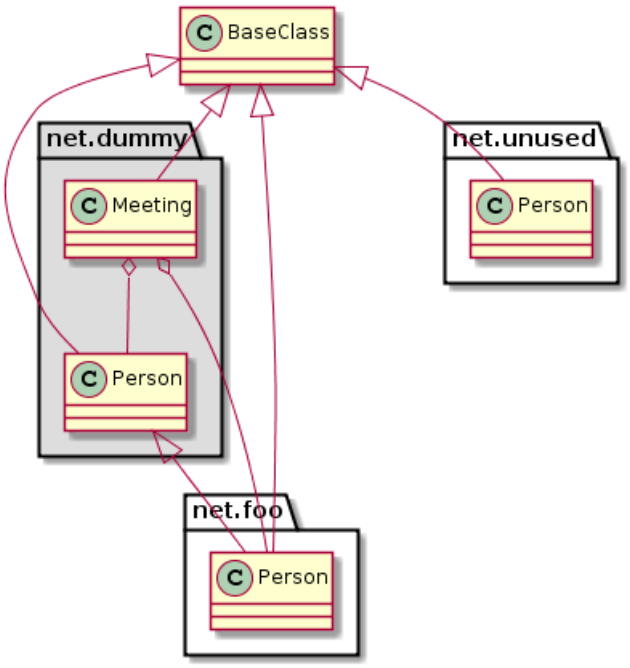
You can specify them either by setting a default style with the command : `skinparam packageStyle`, or by using a stereotype on the package:

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

package foo2 <<Rect>> {
class Class2
}

package foo3 <<Folder>> {
class Class3
}

package foo4 <<Frame>> {
class Class4
}

package foo5 <<Cloud>> {
class Class5
}

package foo6 <<Database>> {
class Class6
}

@enduml
```



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

```
@startuml

skinparam packageStyle rect

package foo1.foo2 {
}

package foo1.foo2.foo3 {
class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Namespaces

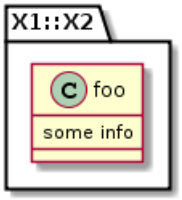
In packages, the name of a class is the unique identifier of this class. It means that you cannot have two classes with the very same name in different packages.

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

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

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

```
@startuml

class BaseClass

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

.BaseClass <|- Meeting
}

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

net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml
```



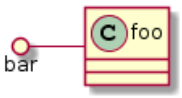
3.19 Automatic namespace creation

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

```
@startuml

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

@enduml
```

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

```
@startuml

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

@enduml
```

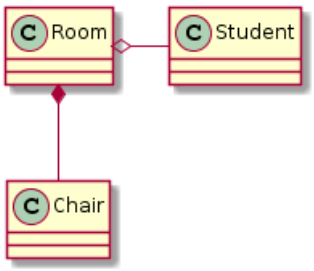


3.20 Lollipop interface

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

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

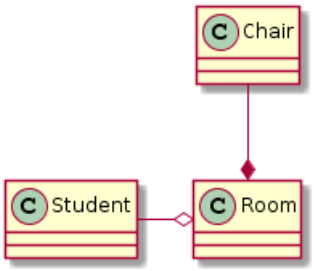
```
@startuml
class foo
bar ()- foo
@enduml
```



3.21 矢印の向きを変える

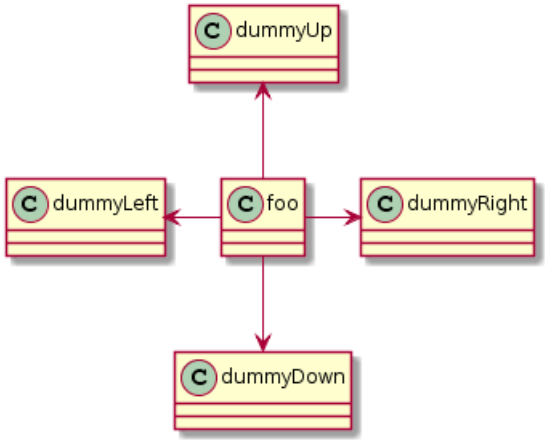
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



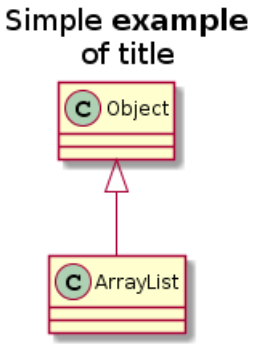
リンクをひっくり返すことにより向きを変えることができる:

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



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

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



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

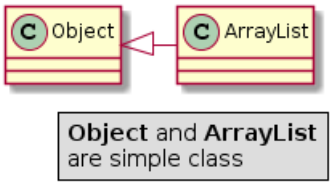
Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

3.22 Title the diagram

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

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

```
@startuml

title Simple <b>example</b>\nof title
Object <|-- ArrayList

@enduml
```

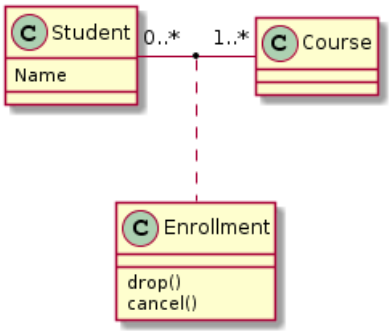


3.23 Legend the diagram

The `legend` and `end legend` are keywords is used to put a legend.

You can optionally specify to have `left`, `right` or `center` alignment for the legend.

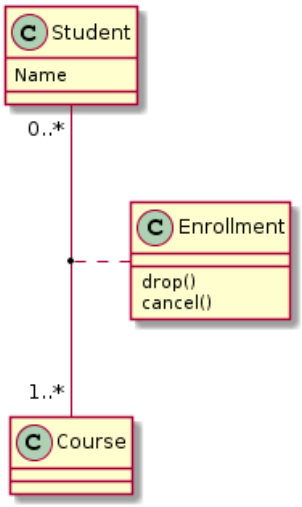
```
@startuml

Object <|- ArrayList

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

@enduml
```



3.24 Association classes

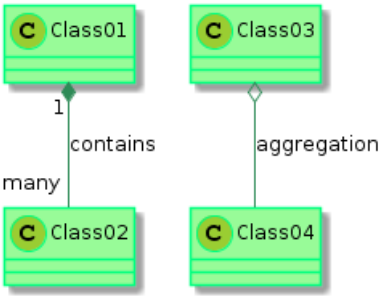
You can define *association class* after that a relation has been defined between two classes, like in this example:

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

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



You can define it in another direction:

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

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



3.25 Skinparam

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

You can use this command :

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

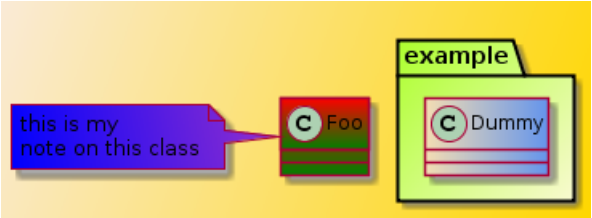
```
@startuml

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

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

Class03 o-- Class04 : aggregation

@enduml
```

```
[From /home/ec2-user/database/tmp/JA/classes.tex (line 1125) ]
... (skipping 12 lines) ...
Class01 << Foo >>
Class01 "1" *-- "many" Class02 : contains

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

3.26 Skinned Stereotypes

You can define specific color and fonts for stereotyped classes.

```
@startuml

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

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

Class03 o-- Class04 : aggregation

@enduml
```



3.27 Color gradient

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

You can use either standard color name or RGB code.

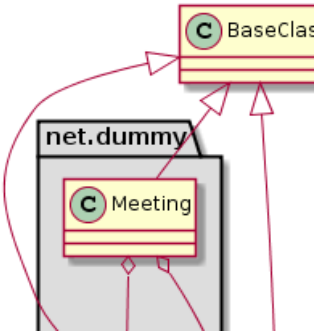
You can also use color gradient in background, with the following syntax: two colors names separated either by:

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

depending the direction of the gradient.

For example, you could have:

```
@startuml

skinparam backgroundcolor AntiqueWhite/Gold
skinparam classBackgroundColor Wheat|CornflowerBlue

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

package example #GreenYellow/LightGoldenRodYellow {
class Dummy
}

@enduml
```



3.28 Help on layout

Sometimes, the default layout is not perfect...

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

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

```
@startuml

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


@enduml
```

3.29 Splitting large files

Sometimes, you will get some very large image files.

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

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

```
@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 アクティビティ図

4.1 単純なアクティビティ

(*) をアクティビティ図の開始点と終了点に使います。

In some occasion, you may want to use (*top) to force the starting point to be at the top of the diagram.

--> で矢印を表します。

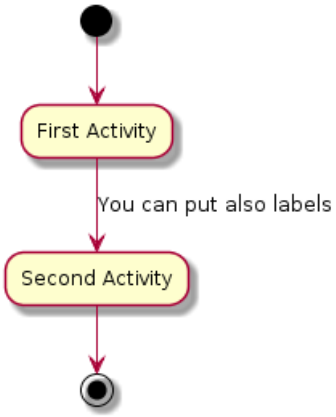
```
@startuml

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

@enduml
```

4.2 矢印のラベル

デフォルトで、矢印は最後に書いたアクティビティを起点に描かれます。

矢印にラベルを付けるために brackets [and] を使います just after the arrow definition.

```
@startuml

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

@enduml
```

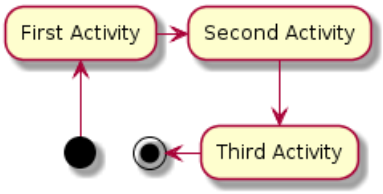


4.3 矢印の方向を変える

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

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

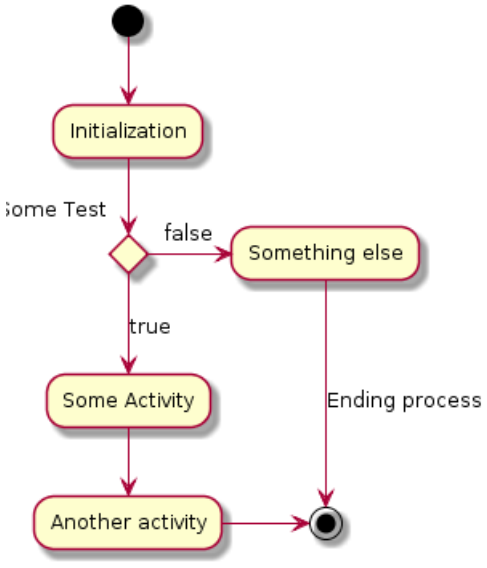
```
@startuml

(*) -up-> "First Activity"
-right-> "Second Activity"
--> "Third Activity"
-left-> (*)

@enduml
```



4.4 分岐

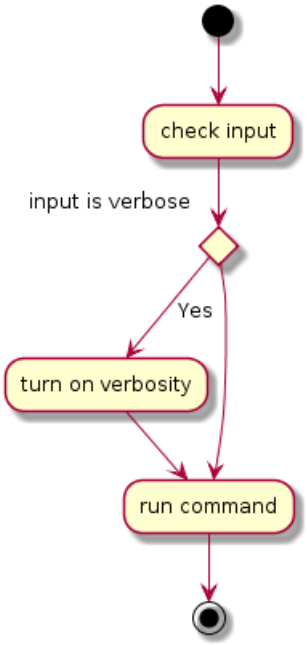
You can use `if/then/else` keywords to define branches.

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

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

@enduml
```



Unfortunately, you will have to sometimes repeat the same activity in the diagram text:

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



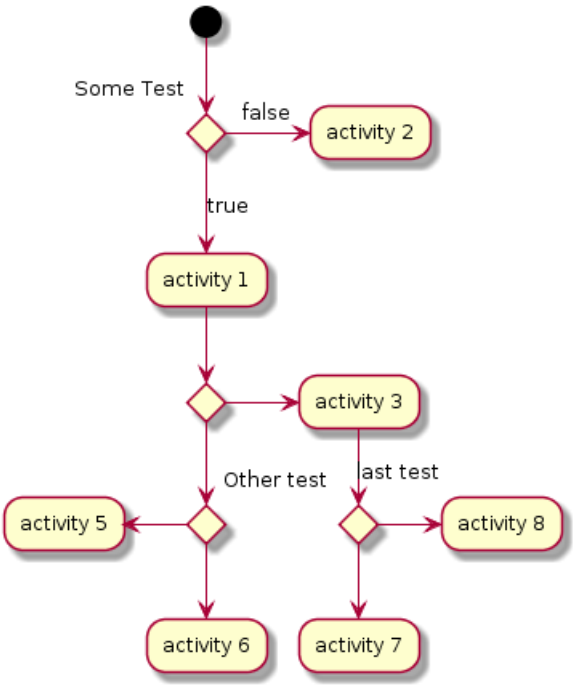
4.5 より多い分岐

By default, a branch is connected to the last defined activity, but it is possible to override this and to define a link with the `if` keywords.

It is also possible to nest branches.

```
@startuml

(*) --> if "Some Test" then

-->[true] "activity 1"

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

else

->[false] "activity 2"

endif

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

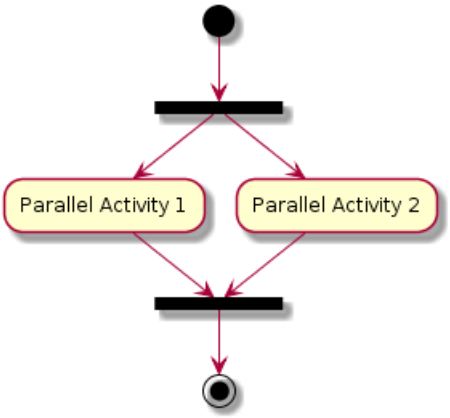
```
endif

@enduml
```



4.6 同期

You can use `===` `code` `===` to display synchronization bars.

```
@startuml

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

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

--> (*)

@enduml
```

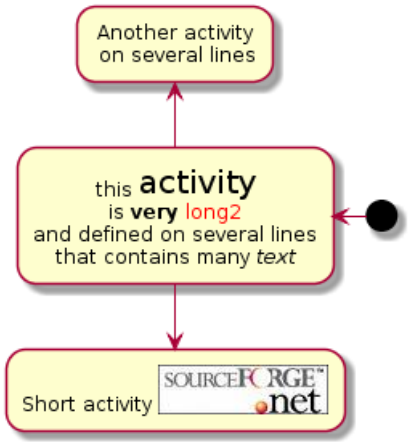


4.7 長いアクティビティの記述

When you declare activities, you can span on several lines the description text. You can also add `\n` in the description.

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

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



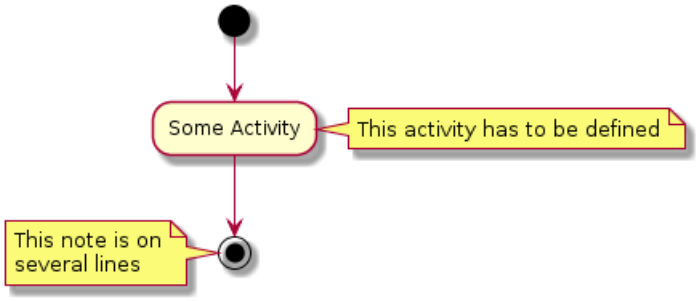
4.8 Notes

You can add notes on a activity using the commands `note left`, `note right`, `note top` or `note bottom`, just after the description of the activity you want to note.

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

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

```
@startuml

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

@enduml
```



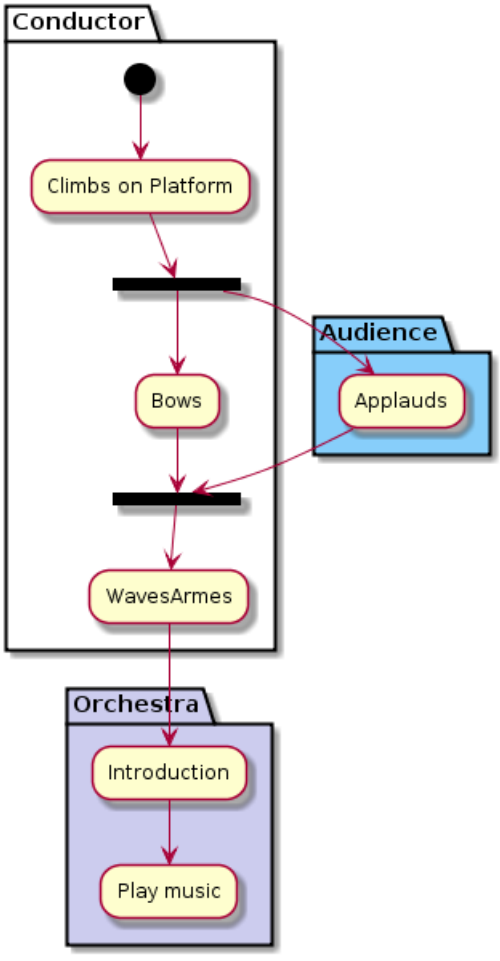
4.9 Partition

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

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

You can close the partition definition using a closing bracket `}`.

```
@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
```

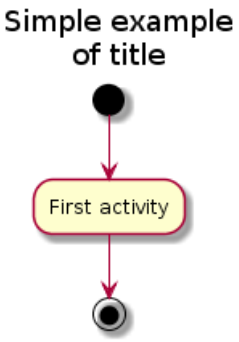


4.10 Title the diagram

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

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

```
@startuml
title Simple example\nof title

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



4.11 Skinparam

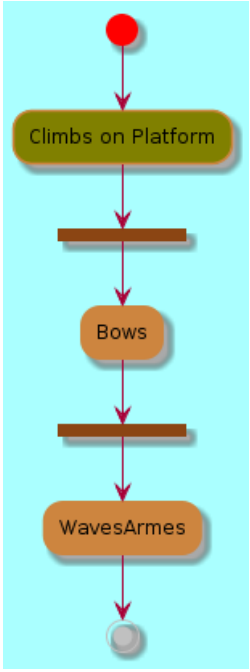
You can use the `skinparam` command to change colors and fonts for the drawing.

You can use this command :

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

You can define specific color and fonts for stereotyped activities.

```
@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 Octagon

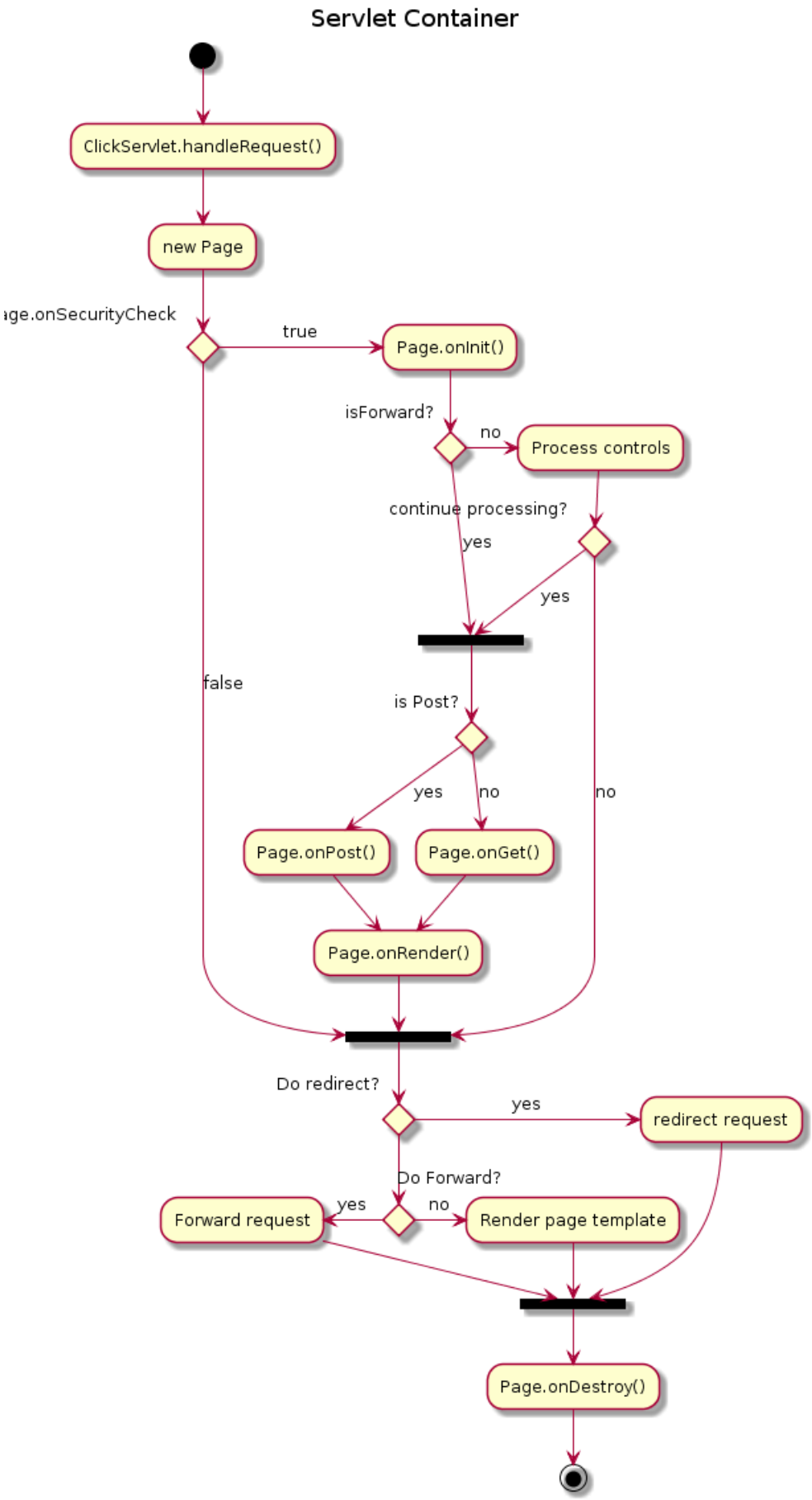
You can change the shape of activities to octagon using the `skinparam activityShape octagon` command.

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

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

@enduml
```

4.13 完全な例

```
@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
```


Servlet Container
ClickServlet.handleRequest()
new Page
ige.onSecurityCheck
true
Page.onInit()
isForward?
no
Process controls
continue processing?
yes
yes
false
is Post?
yes
no
no
Page.onPost()
Page.onGet()
Page.onRender()
Do redirect?
yes
redirect request
Do Forward?
yes
no
Forward request
Render page template
Page.onDestroy()

5　アクティビティ図（ベータ版）

アクティビティ図ための現在の構文には、いくつかの制限と欠点（例えば、メンテナンスが難しい）があります。

私たちがより良い書式と構文を定義できるように、そのような完全な構文と実装は、ベータ版としてユーザーに提案されています。

この新しい実装のもう一つの利点は、Graphviz パッケージのインストールを必要としないことです（シーケンス図には必要）。

新しい構文は古いものに取って代わるでしょう。しかし、互換性の理由で、上位互換を確保することで、古い構文は認識されるでしょう。

ユーザーは新しい構文に移行することをつよく奨励されます。

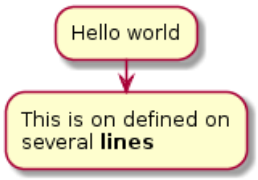
5.1　単純なアクティビティ

アクティビティのラベルは: で開始し; で終了します。

テキストの書式設定は、Creole 記法の Wiki 構文を使用して行うことができます。

それらは定義順に暗黙的にリンクされます。

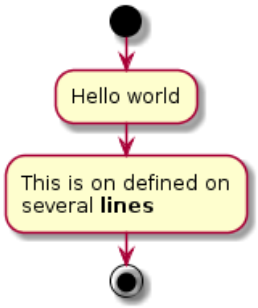
```
@startuml
:Hello world;
:This is on defined on
several **lines**;
@enduml
```



5.2　開始／終了

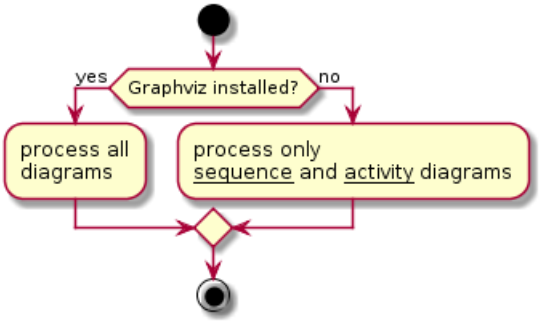
図の開始と終了を示すために、キーワード `start` と `stop` を使用できます。

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



キーワード `end` もまた使用できます。

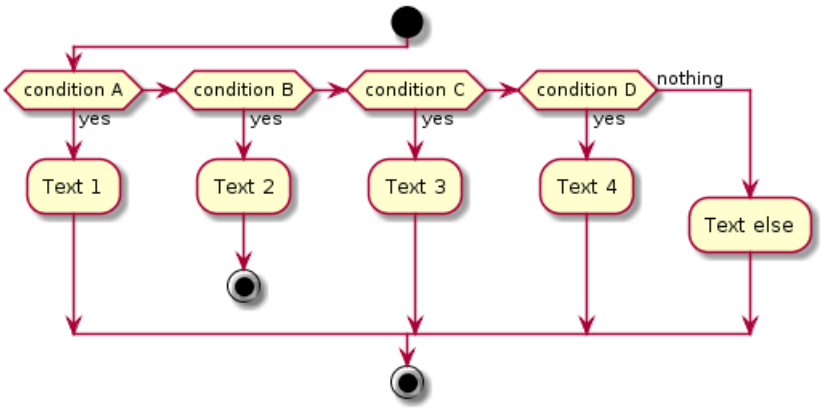
```
@startuml
start
:Hello world;
:This is on defined on
several **lines**;
end
@enduml
```



5.3 条件文

図に条件分岐を追加したい場合は、キーワード `if`、`then` そして `else` を使用することができます。ラベルは括弧を使用することで与えることができます。

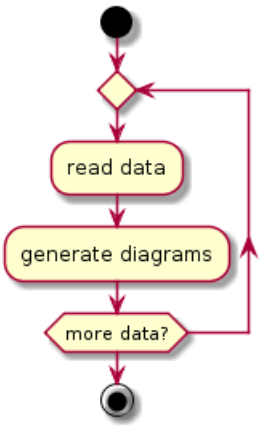
```
@startuml

start

if (Graphviz installed?) then (yes)
:process all\ndiagrams;
else (no)
:process only
__sequence__ and __activity__ diagrams;
endif

stop

@enduml
```



いくつもの条件分岐がある場合には、キーワード `elseif` を使用できます：

```
@startuml
start
if (condition A) then (yes)
:Text 1;
elseif (condition B) then (yes)
:Text 2;
stop
```

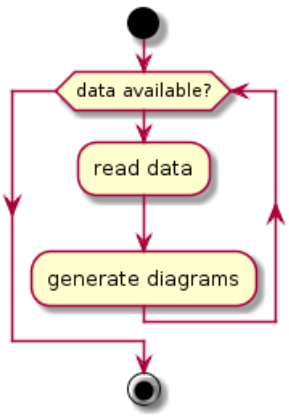
```
elseif (condition C) then (yes)
:Text 3;
elseif (condition D) then (yes)
:Text 4;
else (nothing)
:Text else;
endif
stop
@enduml
```



5.4 繰り返し（後判定）

繰り返し処理（後判定）がある場合には、キーワード `repeat` と `repeat while` を使用できます。

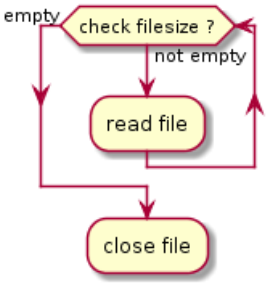
```
@startuml

start

repeat
:read data;
:generate diagrams;
repeat while (more data?)

stop

@enduml
```



5.5 繰り返し（前判定）

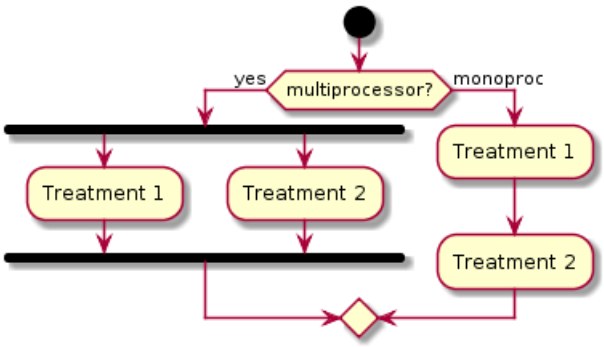
繰り返し処理（前判定）がある場合には、キーワード `while` と `end while` を使用できます。

```
@startuml

start

while (data available?)
:read data;
:generate diagrams;
endwhile

stop

@enduml
```



キーワード endwhile の後ろ、または、キーワード is を使用することで、ラベルを与えることができます。

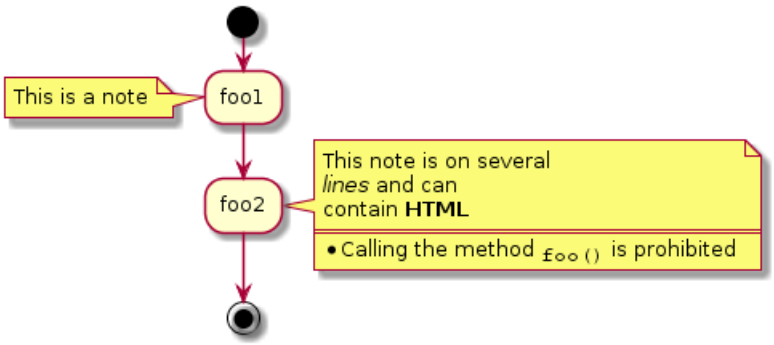
```
@startuml
while (check filesize ?) is (not empty)
:read file;
endwhile (empty)
:close file;
@enduml
```



5.6 並列処理

並列処理を示すために、キーワード fork、fork again そして end fork が使用できます。

```
@startuml

start

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

@enduml
```

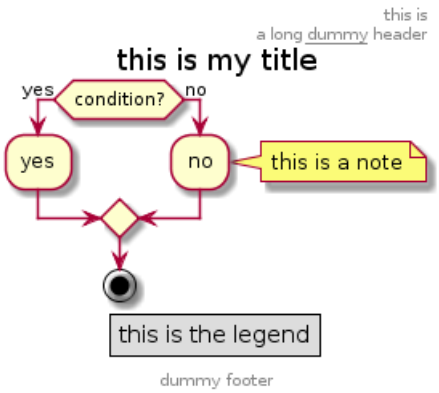


5.7 注釈

Creole 表記の Wiki 構文を使用することで、テキストの書式設定ができます。

キーワード floating を使用し、注釈を遊離させることもできます。

```
@startuml

start
:foo1;
floating 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
```

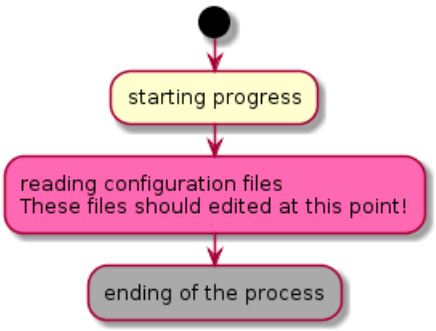


5.8 タイトル&説明文

タイトル、ヘッダ、フッタ、図の説明を付け足すことができます：

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

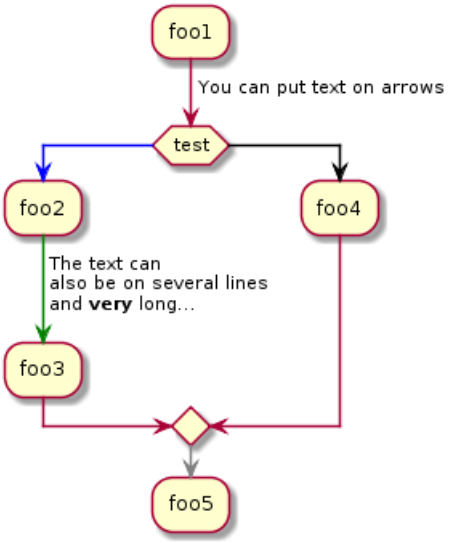
```
stop

legend
this is the legend
endlegend

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

@enduml
```



5.9 色指定

各アクティビティに、色を指定することができます。

```
@startuml

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

@enduml
```

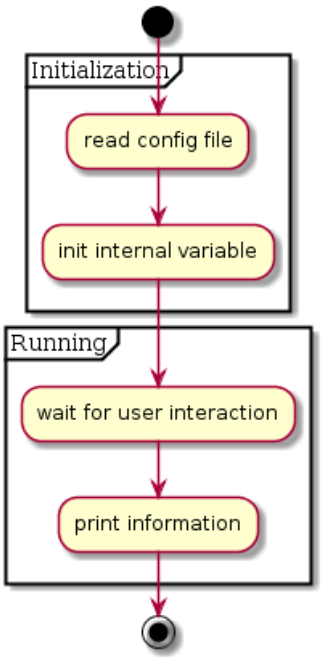


5.10 矢印

記号 -> を用いて、矢印にテキストを添えることができ、また、色を変えることもできます。

点線、破線、太線、または、矢印なし、もまた可能です。

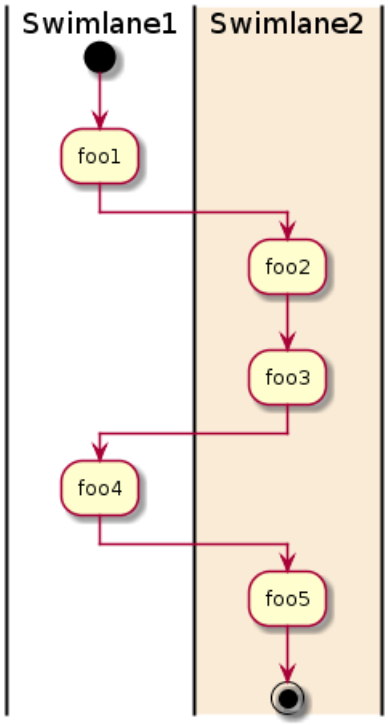
```
@startuml
:foo1;
-> You can put text on arrows;
if (test) then
-[#blue]->
:foo2;
-[#green,dashed]-> The text can
also be on several lines
and **very** long...;
:foo3;
else
-[#black,dotted]->
:foo4;
endif
-[#gray,bold]->
:foo5;
@enduml
```



5.11 グループ化

キーワード partition で複数のアクティビティを分割して、グループ化できます：

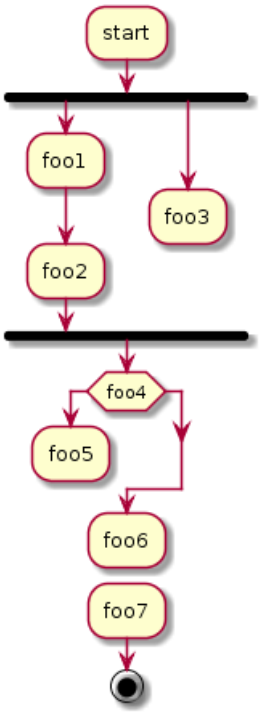
```
@startuml
start
partition Initialization {
:read config file;
:init internal variable;
}
partition Running {
:wait for user interaction;
:print information;
}

stop
@enduml
```



5.12 動線

パイプ記号 | を用いて、複数の動線を定義することができます。
さらに、動線毎に色を変えることができます。

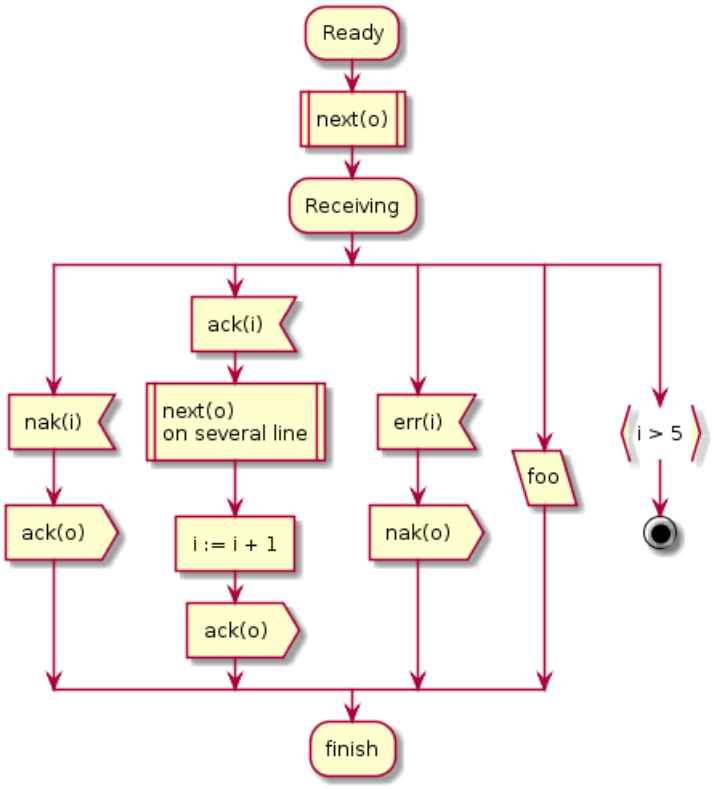
```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```



5.13 分離

キーワード detach を使用して、矢印を取り除くことができます。

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

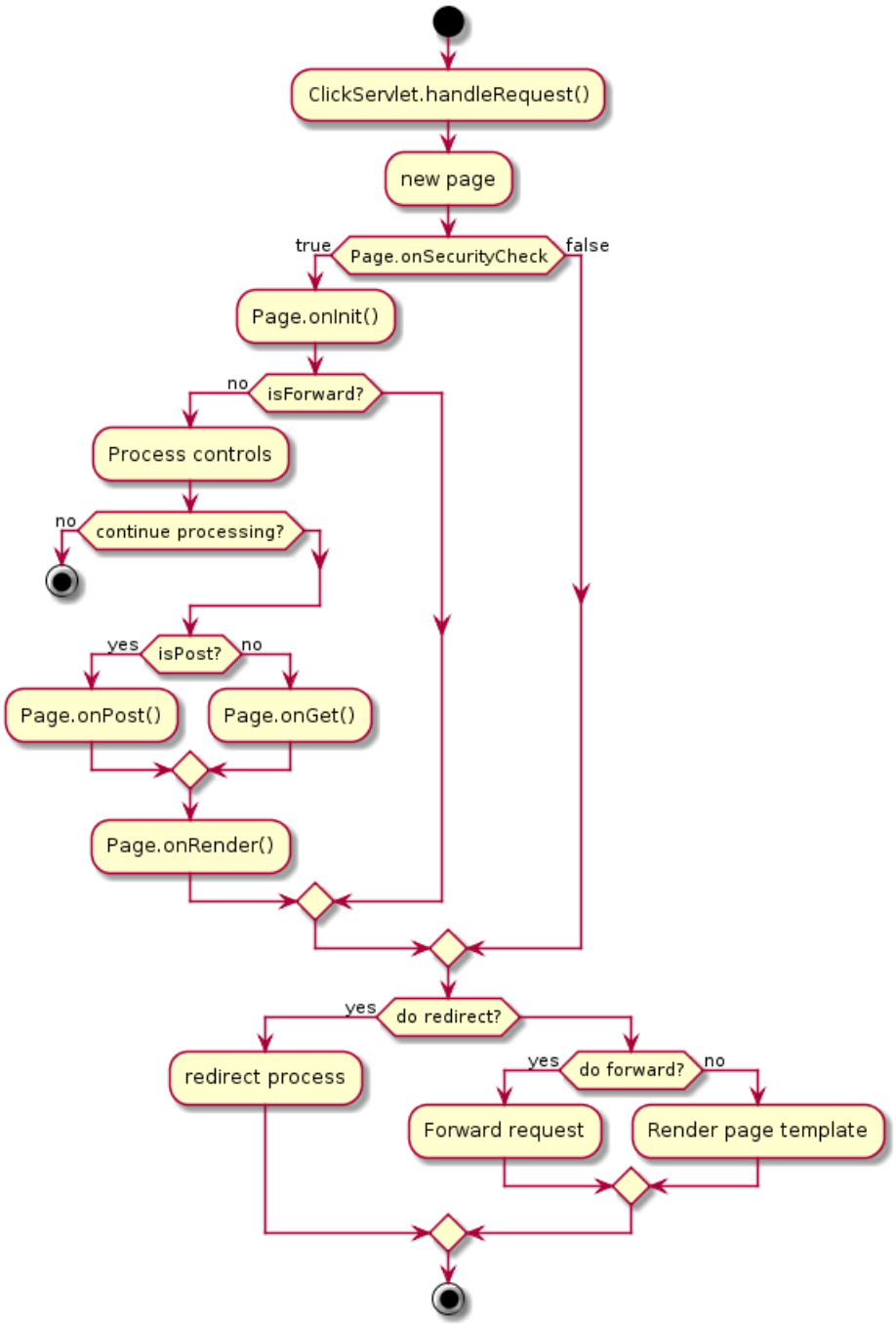


5.14 SDL 図

終端記号; を置き換えることで、アクティビティの表現形式を変えることができます：

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

```
@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 完全な例

```
@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

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
```

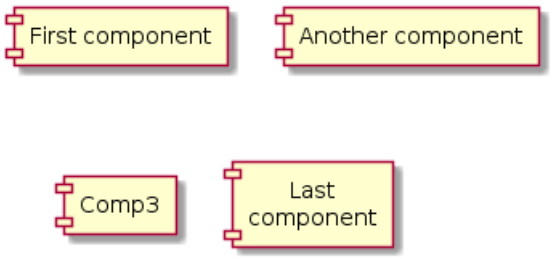
6 コンポーネント図

6.1 コンポーネント

コンポーネントは括弧でくくります。

また、component キーワードでもコンポーネントを定義できます。そして、コンポーネントには as キーワードにより別名をつけることができます。この別名は、後でリレーションを定義するときに使えます。

```
@startuml

[First component]
[Another component] as Comp2
component Comp3
component [Last\ncomponent] as Comp4

@enduml
```



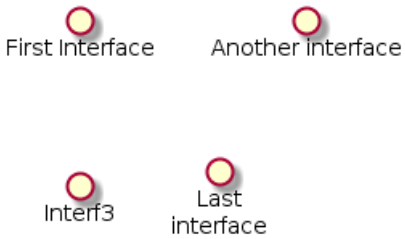
6.2 インタフェース

Interface can be defined using the () symbol (because this looks like a circle).

You can also use the **interface** keyword to defines an interface. And you can define an alias, using the **as** keyword. This alias will be used latter, when defining relations.

We will see latter that interface definition is optional.

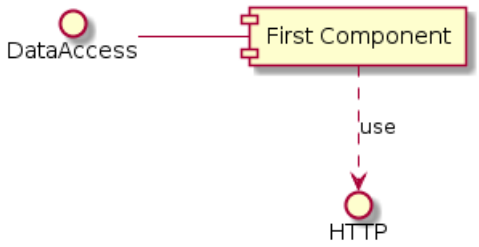
```
@startuml

() "First Interface"
() "Another interface" as Interf2
interface Interf3
interface "Last\ninterface" as Interf4

@enduml
```



6.3 Basic example

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

```
@startuml

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

@enduml
```

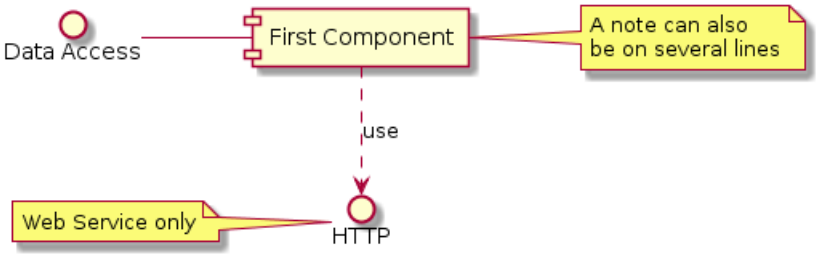


6.4 Using notes

You can use the `note left of` , `note right of` , `note top of` , `note bottom of` keywords to define notes related to a single object.

A note can be also define 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
```

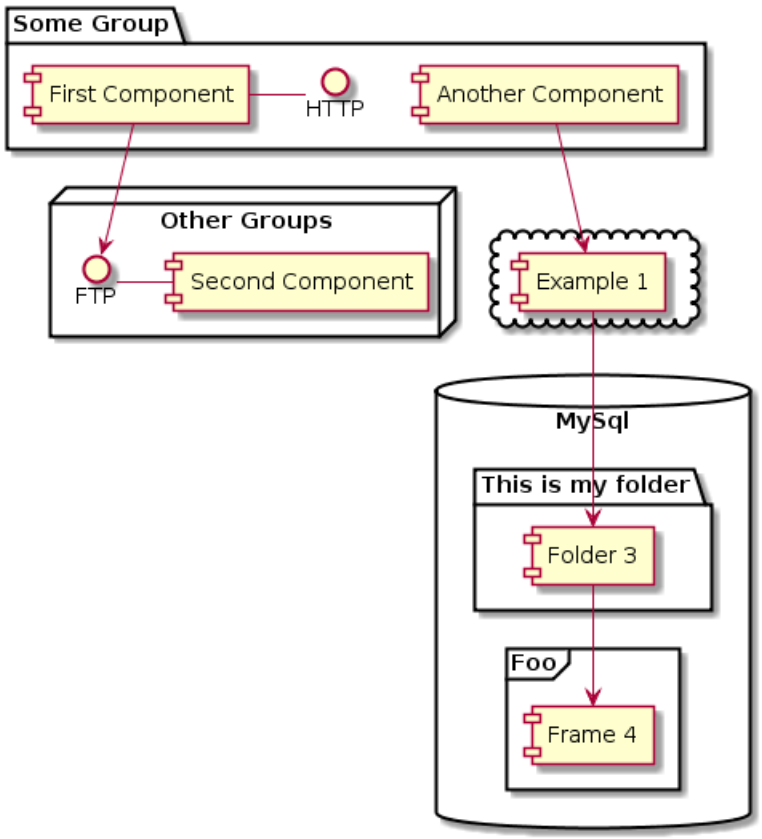


6.5 Grouping Components

You can use several keywords to group components and interfaces together:

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

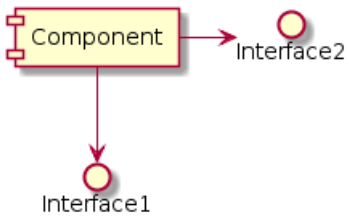
```
@startuml

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 Changing arrows direction

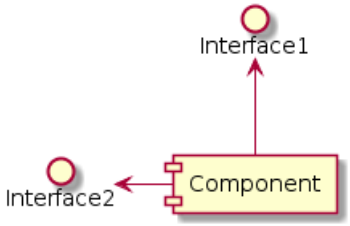
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



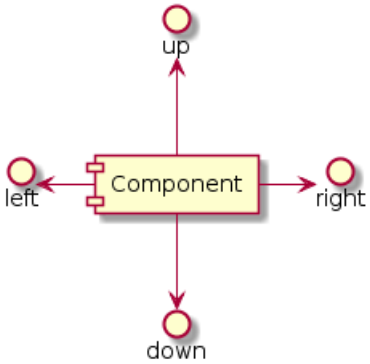
You can also change directions by reversing the link:

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



また、left を加えることで矢印の向きを変更することもできます。right, up, down などのキーワードを矢印の間に記述します。:

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



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

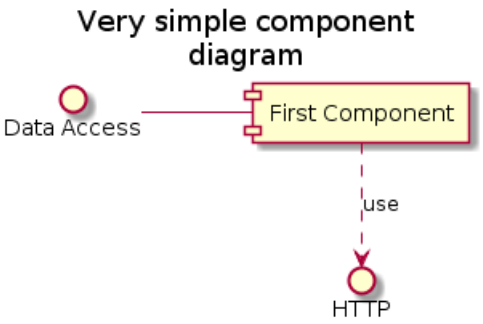
Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

6.7 Title the diagram

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

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

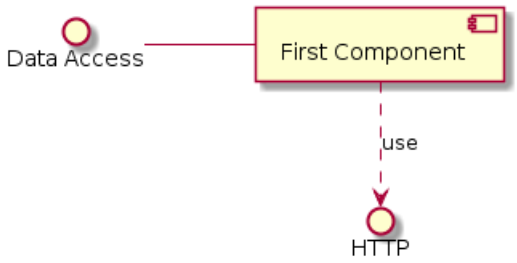
```
@startuml
title Very simple component\ndiagram

interface "Data Access" as DA

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

@enduml
```



6.8 Use UML2 notation

The `skinparam componentStyle uml2` command is used to switch to UML2 notation.

```
@startuml
skinparam componentStyle uml2

interface "Data Access" as DA

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

@enduml
```



6.9 Individual colors

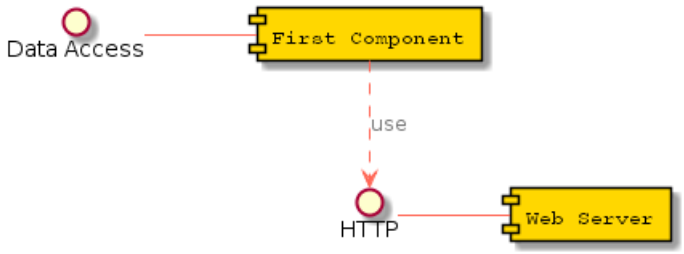
You can specify a color after component definition.

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

6.10 Using Sprite in Stereotype

You can use sprites within stereotype components.

```
@startuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFF0FFFFF
FFFFFFFFFF00FFFF
FF00000000000FFF
FF0000000000000F
FF00000000000FFF
FFFFFFFFFF00FFFF
FFFFFFFFFF0FFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
}


rectangle " End to End\nbusiness process" <<$businessProcess>> {
rectangle "inner process 1" <<$businessProcess>> as src
rectangle "inner process 2" <<$businessProcess>> as tgt
src -> tgt
}
@enduml
```



6.11 Skinparam

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

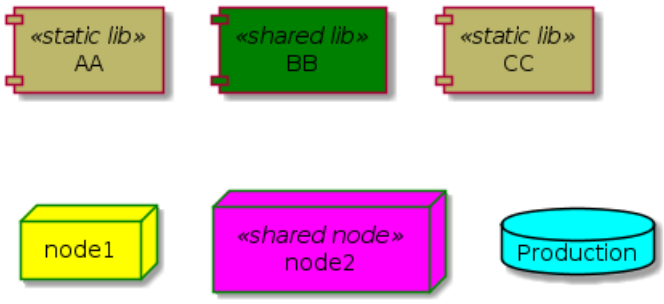
You can use this command :

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

You can define specific color and fonts for stereotyped components and interfaces.

```
@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
```



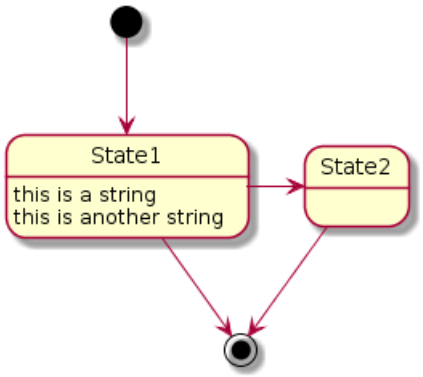

```
@startuml
[AA] <<static lib>>
[BB] <<shared lib>>
[CC] <<static lib>>

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　ステート図

7.1　簡単なステート

ステート図の始点と終点は、[*] で示します。
矢印は、--> で示します。

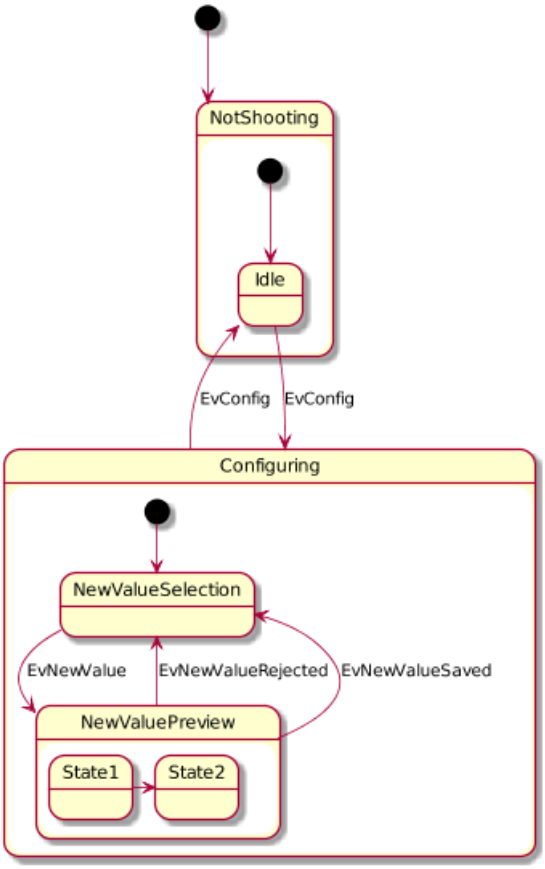
```
@startuml

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

State1 -> State2
State2 --> [*]

@enduml
```



7.2　複合状態

状態は複合することができます。キーワード state と中括弧を使用して定義することができます。

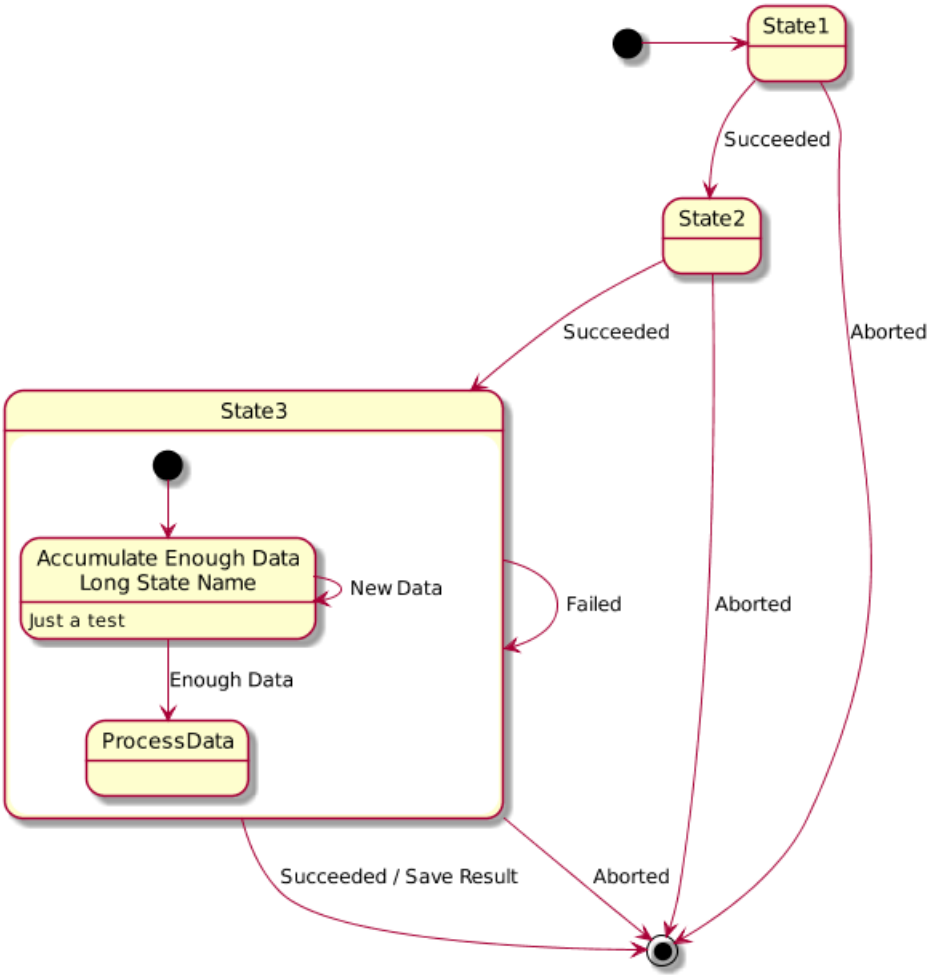
```
@startuml
scale 350 width
[*] --> NotShooting

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 長い言葉

キーワード state によって、状態についての長めの記述を使用することができます。

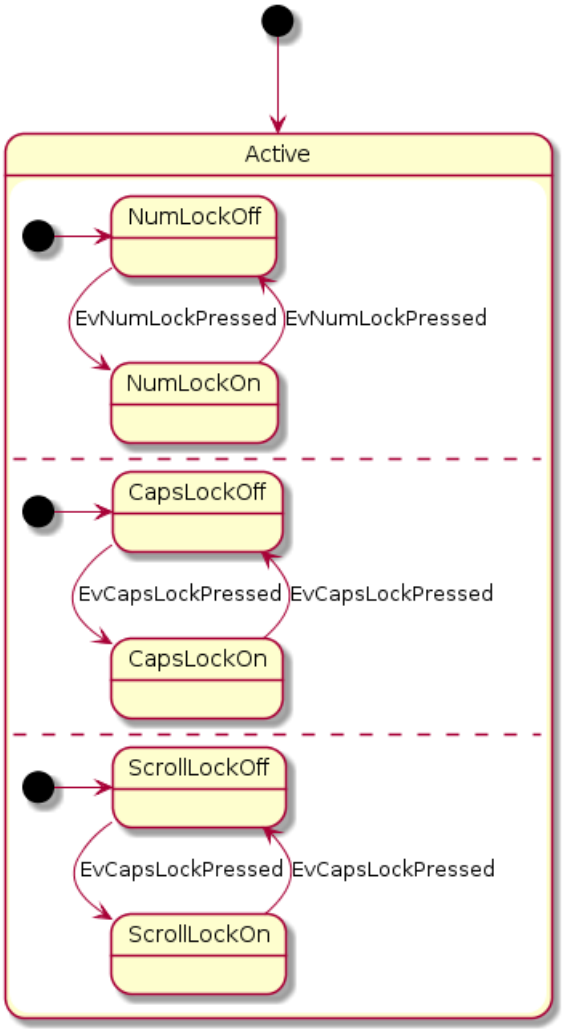
```
@startuml
scale 600 width

[*] -> 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 同時状態

記号 --で分離することで、同時状態となる複合状態を定義することができます。

```
@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
```

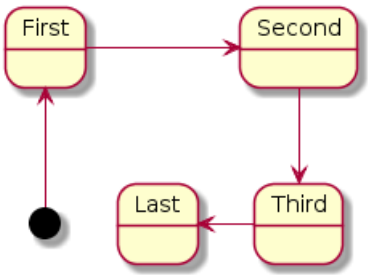


7.5 矢印の方向

記号 -> を水平矢印として使用でき、以下の構文を使用することで、矢印の方向を支配することができます。

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

```
@startuml

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

@enduml
```



方向を示す単語の、最初の文字だけ（例：-down-の代わりに -d-）、または 2 文字（-do-）を使用することで、矢印の記述を短くすることができます。

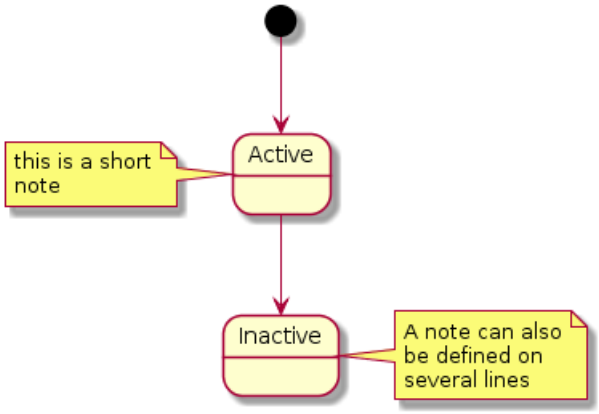
この機能を乱用しないよう注意しなくてはいけません：通常、*Graphviz* は微調整なしでよい結果をもたらしてくれます。

7.6 注釈

キーワード `note left of`, `note right of`, `note top of`, `note bottom of` を使用して注釈を定義することができます。

さらに、いくつもの行で注釈を定義できます。

```
@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
```



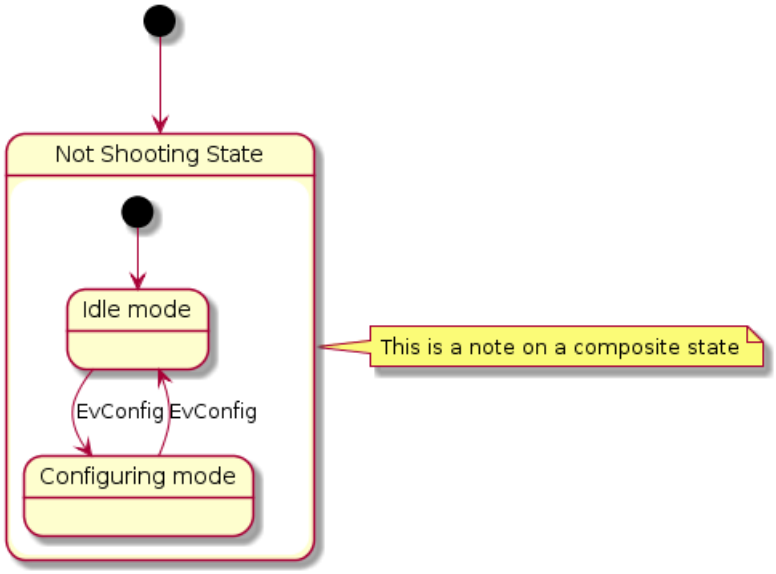
さらに、状態にひもづかない注釈を定義できます。

```
@startuml

state foo
note "This is a floating note" as N1

@enduml
```

7.7 もっと注釈

複合状態にも注釈をつけることができます。

```
@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 見栄え

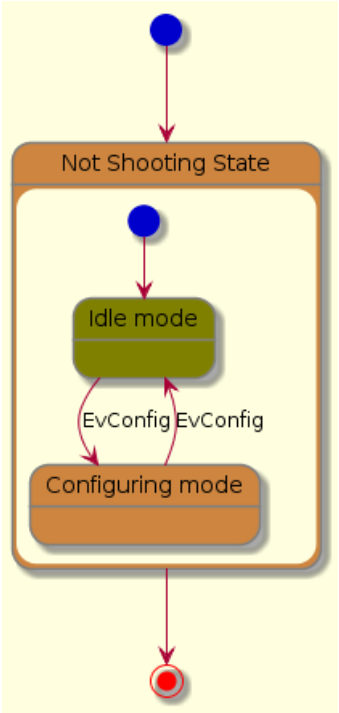
描画のフォントや色を変えるコマンド skinparam を使用できます。
このコマンドは以下のように使用できます。

- 図の定義において、他のコマンドと同じように。
- インクルードファイルとして。
- 設定ファイルとして、コマンドラインや ANT タスクから提供する。

定型化した状態に、特定の色とフォントを定義することができます。

```
@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 オブジェクト図

8.1 オブジェクトの定義

オブジェクトのインスタンスを、キーワード object を使用して定義します。

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

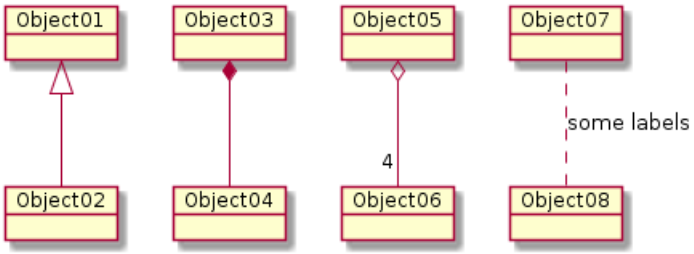
8.2 オブジェクト間の関係

オブジェクト間の関係は次の記号を用いて定義します:

継承	<|--	
合成	*--	
集約	o--	

-- を .. に置き換えることで点線を示すことができます。
これらのルールを知ることで、以下の図を描くことができます。
関係にラベルをつけることができ、”：”を用い、ラベルの文字列を続けます。
関係の各側のスペースを含む文字列を引用符 "" で囲むことができます。

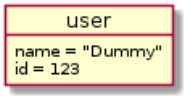
```
@startuml
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

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



8.3 Adding fields

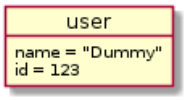
To declare fields, you can use the symbol ":" followed by the field's name.

```
@startuml

object user

user : name = "Dummy"
user : id = 123

@enduml
```



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

```
@startuml

object user {
name = "Dummy"
id = 123
}

@enduml
```



8.4 Common features with class diagrams

- Visibility
- Defines notes
- Use packages
- Title the diagram
- Skin the output
- Split the image

9 共通コマンド

9.1 フッターとヘッダー

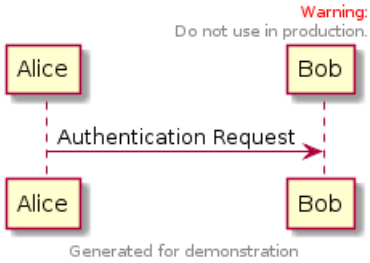
どのダイアグラムも、header 及び footer で、ヘッダー及びフッターを追加できます。

キーワード center、left、right を追記することで、ヘッダー／フッターを中央、左、右に寄せることが可能です。

タイトルと同様にヘッダー／フッターは複数行記述可能です。

ヘッダー／フッターを HTML で記述することも可能です。

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

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

center footer Generated for demonstration

@enduml
```



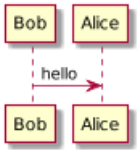
9.2 ズーム

scale コマンドで、生成するイメージのズームを指定できます。

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

- scale 1.5
- scale 2/3
- scale 200 width
- scale 200 height
- scale 200*100
- scale max 300*200

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



10 Salt

Salt is a subproject included in PlantUML that may help you to design graphical interface.
You can use either `@startsalt` keyword, or `@startuml` followed by a line with `salt` keyword.

10.1 Basic widgets

A window must start and end with brackets. You can then define:

- Button using `[` and `]`.
- Radio button using `(` and `)`.
- Checkbox using `[` and `]`.
- User text area using `"`.

```
@startuml
salt
{
Just plain text
[This is my button]
()  Unchecked radio
(X) Checked radio
[]  Unchecked box
[X] Checked box
"Enter text here   "
^This is a droplist^
}
@enduml
```

The goal of this tool is to discuss about simple and sample windows.

10.2 Using grid

A table is automatically created when you use an opening bracket `{`.
And you have to use `|` to separate columns.
For example:

```
@startsalt
{
Login    | "MyName   "
Password | "****     "
[Cancel] | [  OK   ]
}
@endsalt
```

Just after the opening bracket, you can use a character to define if you want to draw lines or columns of the grid :

`#` To display all vertical and horizontal lines

`!` To display all vertical lines

`-` To display all horizontal lines

`+` To display external lines

```
@startsalt
{+
Login    | "MyName   "
Password | "****     "
[Cancel] | [  OK   ]
}
@endsalt
```

10.3 Using separator

You can use several horizontal lines as separator.

```
@startsalt
{
Text1
..
"Some field"
==
Note on usage
~~
Another text
--
[Ok]
}
@endsalt
```

10.4 Tree widget

To have a Tree, you have to start with `{T` and to use `+` to denote hierarchy.

```
@startsalt
{
{T
+ World
++ America
+++ Canada
+++ USA
++++ New York
++++ Boston
+++ Mexico
++ Europe
+++ Italy
+++ Germany
++++ Berlin
++ Africa
}
}
@endsalt
```

10.5 Enclosing brackets

You can define subelements by opening a new opening bracket.

```
@startsalt
{
Name          | "                 "
Modifiers:    | { (X) public | () default | () private | () protected
[] abstract | [] final    | [] static }
Superclass:   | { "java.lang.Object " | [Browse...] }
}
@endsalt
```

10.6 Adding tabs

You can add tabs using `{/` notation. Note that you can use HTML code to have bold text.

```
@startsalt
{+
{/ <b>General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```

Tab could also be vertically oriented:

```
@startsalt
{+
{/ <b>General
Fullscreen
Behavior
Saving } |
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
[Close]
}
}
@endsalt
```

10.7 Using menu

You can add a menu by using `{*` notation.

```
@startsalt
{+
{* File | Edit | Source | Refactor }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```

It is also possible to open a menu:

```
@startsalt
{+
{* File | Edit | Source | Refactor
Refactor | New | Open File | - | Close | Close All }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```

10.8 Advanced table

You can use two special notations for table :

- `*` to indicate that a cell with span with left
- `.` to denotate an empty cell

```
@startsalt
{#
. | Column 2 | Column 3
Row header 1 | value 1 | value 2
Row header 2 | A long cell | *
}
@endsalt
```

11 Creole

A light Creole engine have been integrated into PlantUML to have a standardized way of defining text style.

All diagrams are now supporting this syntax.

Note that ascending compatibility with HTML syntax is preserved.

11.1 Emphasized text

```
@startuml
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
@enduml
```

11.2 List

```
@startuml
object demo {
* Bullet list
* Second item
** Sub item
}

legend
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
end legend
@enduml
```

11.3 Escape character

You can use the tilde ~ to escape special creole characters.

```
@startuml
object demo {
This is not ~___underscored__.
This is not ~""monospaced"".
}
@enduml
```

11.4 Horizontal lines

```
@startuml
database DB1 as "
You can have horizontal line
----
Or double line
====
Or strong line
____
Or dotted line
..My title..
Enjoy!
"
note right
This is working also in notes
You can also add title in all these lines
==Title==
--Another title--
end note

@enduml
```

11.5 Headings

```
@startuml
usecase UC1 as "
= Extra-large heading
Some text
== Large heading
Other text
=== Medium heading
Information
....
==== Small heading"
@enduml
```

11.6 Legacy HTML

Some HTML tags are also working:

- `<b>` for bold text
- `<u>` or `<u:#AAAAAA>` or `<u:colorName>` for underline
- `<i>` for italic
- `<s>` or `<s:#AAAAAA>` or `<s:colorName>` for strike text
- `<w>` or `<w:#AAAAAA>` or `<w:colorName>` for wave underline text
- `<color:#AAAAAA>` or `<color:colorName>`
- `<back:#AAAAAA>` or `<back:colorName>` for background color
- `<size:nn>` to change font size
- `<img:file>` : the file must be accessible by the filesystem
- `<img:http://url>` : the URL must be available from the Internet

```
@startuml
:* You can change <color:red>text color</color>
* You can change <back:cadetblue>background color</back>
* You can change <size:18>size</size>
* You use <u>legacy</u> <b>HTML <i>tag</i></b>
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
```

```
----
* Use image : <img:sourceforge.jpg>
;

@enduml
```

11.7 Table

```
@startuml
skinparam titleFontSize 14
title
Example of simple table
|= |= table |= header |
| a | table | row |
| b | table | row |
end title
[*] --> State1
@enduml
```

11.8 Tree

You can use `|_` characters to build a tree.

```
@startuml
skinparam titleFontSize 14
title
Example of Tree
|_ First line
|_ **Bom(Model)**
|_ prop1
|_ prop2
|_ prop3
|_ Last line
end title
[*] --> State1
@enduml
```

11.9 Special characters

It's possible to use any unicode characters with `&#` syntax or `<U+XXXX>`

```
startuml
usecase foo as "this is &#8734; long"
usecase bar as "this is also <U+221E> long"
enduml
```

11.10 OpenIconic

OpenIconic is an very nice open source icon set. Those icons have been integrated into the creole parser, so you can use them out-of-the-box.

You can use the following syntax: `<&ICON_NAME>`.

```
@startuml
title: <size:20><&heart>Use of OpenIconic<&heart></size>
class Wifi
note left
Click on <&wifi>
end note
@enduml
```

The complete list is available on OpenIconic Website, or you can use the following special diagram:

```
@startuml
listopeniconic
@enduml
```

11.11 Defining and using sprites

A *Sprite* is a small graphic element that can be used in diagrams.

In PlantUML, sprites are monochrome and can have either 4, 8 or 16 gray level.

To define a sprite, you have to use a hexadecimal digit between 0 and F per pixel.

Then you can use the sprite using `<$XXX>` where XXX is the name of the sprite.

```
@startuml
sprite $foo1 {
FFFFFFFFFFFFFFF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
FFFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1>
@enduml
```

11.12 Encoding Sprite

To encode sprite, you can use the command line like:

```
java -jar plantuml.jar -encodesprite 16z foo.png
```

where `foo.png` if the image file you want to use (it will be converted to gray automatically).

After `-encodesprite`, you have to specify a format: `4, 8, 16, 4z, 8z` or `16z`.

The number indicates the gray level and the optionnal `z` is used to enable compression in sprite definition.

11.13 Importing Sprite

You can also launch the GUI to generate a sprite from an existing image.

Click in the menubar then on `File/Open Sprite Window`.

After copying an image into you clipboard, several possible definitions of the corresponding sprite will be displayed : you will just have to pickup the one you want.

11.14 Examples

```
@startuml
sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPEc
start
:click on <$printer> to print the page;
@enduml
```

```
@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S4OT7C53cKpsHpaYPDJY_12MHM-BLRyywPhrrlw3qur
sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPEc
sprite $disk {
444445566677881
436000000009991
43600000000ACA1
53700000001A7A1
53700000012B8A1
53800000123B8A1
63800001233C9A1
634999AABBC99B1
744566778899AB1
7456AAAAA99AAB1
8566AFC228AABB1
8567AC8118BBBB1
867BD4433BBBBB1
39AAAAABBBBBBC1
}

title Use of sprites (<$printer>, <$bug>...)

class Example {
Can have some bug : <$bug>
Click on <$disk> to save
}

note left : The printer <$printer> is available

@enduml
```

12　フォントや色の変更

12.1　使い方

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

```
skinparam backgroundColor yellow
```

You can use this command :

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

12.2　Nested

To avoid repetition, it is possible to nest definition. So the following definition :

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

is strictly equivalent to:

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

12.3 色

標準的な色名または RGB コードのいずれかを使用できます。

パラメータ名	既定値	色	解説
backgroundColor	white		ページ背景色
activityArrowColor	#A80036		アクティビティ図の矢線色
activityBackgroundColor	#FEFECE		アクティビティ毎の背景色
activityBorderColor	#A80036		アクティビティの枠線色
activityStartColor	black		アクティビティ図の開始円
activityEndColor	black		アクティビティ図の終了円
activityBarColor	black		アクティビティ図の同期バー
usecaseArrowColor	#A80036		ユースケース図の矢線色
usecaseActorBackgroundColor	#FEFECE		ユースケース図のアクターの頭の色
usecaseActorBorderColor	#A80036		ユースケース図のアクターの縁取り色
usecaseBackgroundColor	#FEFECE		各ユースケースの背景
usecaseBorderColor	#A80036		ユースケース図のユースケース枠線色
classArrowColor	#A80036		クラス図の矢線の色
classBackgroundColor	#FEFECE		クラス図のクラス／インタフェース／列挙型の背景色
classBorderColor	#A80036		クラス図のクラス／インタフェース／列挙型の枠線色
packageBackgroundColor	#FEFECE		クラス図のパッケージの背景色
packageBorderColor	#A80036		クラス図のパッケージの枠線色
stereotypeCBackgroundColor	#ADD1B2		クラス図のクラススポットの背景色
stereotypeABackgroundColor	#A9DCDF		クラス図の抽象クラススポットの背景色
stereotypeIBackgroundColor	#B4A7E5		クラス図のインタフェーススポットの背景色
stereotypeEBackgroundColor	#EB937F		クラス図の列挙型スポットの背景色
componentArrowColor	#A80036		コンポーネント図の矢線色
componentBackgroundColor	#FEFECE		コンポーネントの背景色
componentBorderColor	#A80036		コンポーネントの枠線色
componentInterfaceBackgroundColor	#FEFECE		コンポーネント図のインタフェースの背景色
componentInterfaceBorderColor	#A80036		コンポーネント図のインタフェースの枠線色
noteBackgroundColor	#FBFB77		注釈の背景色
noteBorderColor	#A80036		注釈の枠線色
stateBackgroundColor	#FEFECE		状態遷移図のステートの背景色
stateBorderColor	#A80036		状態遷移図のステートの枠線色
stateArrowColor	#A80036		状態遷移図の矢線色
stateStartColor	black		状態遷移図の始点の色
stateEndColor	black		状態遷移図の終点の色
sequenceArrowColor	#A80036		シーケンス図の矢線色
sequenceActorBackgroundColor	#FEFECE		シーケンス図のアクターの頭の色
sequenceActorBorderColor	#A80036		シーケンス図のアクターの縁取り色
sequenceGroupBackgroundColor	#EEEEEE		シーケンス図の alt/opt/loop のヘッダ色
sequenceLifeLineBackgroundColor	white		シーケンス図のライフラインの背景色
sequenceLifeLineBorderColor	#A80036		シーケンス図のライフラインの枠線色
sequenceParticipantBackgroundColor	#FEFECE		シーケンス図の分類子の背景色
sequenceParticipantBorderColor	#A80036		シーケンス図の分類子の枠線色

12.4 フォントの色、名前、サイズ

テキスト描画に使用されるフォントを、パラメータ xxxFontColor、xxxFontSize、xxxFontName で変更することができます。

例：

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

skinparam defaultFontName を使用して既定のフォントを変更することができます。

例:

```
skinparam defaultFontName Aapex
```

フォント名はシステム依存性が非常に高いため、ポータビリティを考慮する場合は多用しないでください。

パラメータ名	デフォルト値	解説
activityFontColor activityFontSize activityFontStyle activityFontName	black 14 plain	アクティビティボックスに使用される
activityArrowFontColor activityArrowFontSize activityArrowFontStyle activityArrowFontName	black 13 plain	アクティビティ図の矢印につけるテキストに使用される
circledCharacterFontColor circledCharacterFontSize circledCharacterFontStyle circledCharacterFontName circledCharacterRadius	black 17 bold Courier 11	クラスや列挙値などを囲う円の中のテキストに使用される
classArrowFontColor classArrowFontSize classArrowFontStyle classArrowFontName	black 10 plain	クラス図の矢印につけるテキストに使用される
classAttributeFontColor classAttributeFontSize classAttributeIconSize classAttributeFontStyle classAttributeFontName	black 10 10 plain	クラスの属性とメソッド
classFontColor classFontSize classFontStyle classFontName	black 12 plain	クラス名に使用される
classStereotypeFontColor classStereotypeFontSize classStereotypeFontStyle classStereotypeFontName	black 12 italic	クラス内のステレオタイプに使用される
componentFontColor componentFontSize componentFontStyle componentFontName	black 14 plain	コンポーネント名に使用される
componentStereotypeFontColor componentStereotypeFontSize componentStereotypeFontStyle componentStereotypeFontName	black 14 italic	コンポーネント内のステレオタイプに使用される

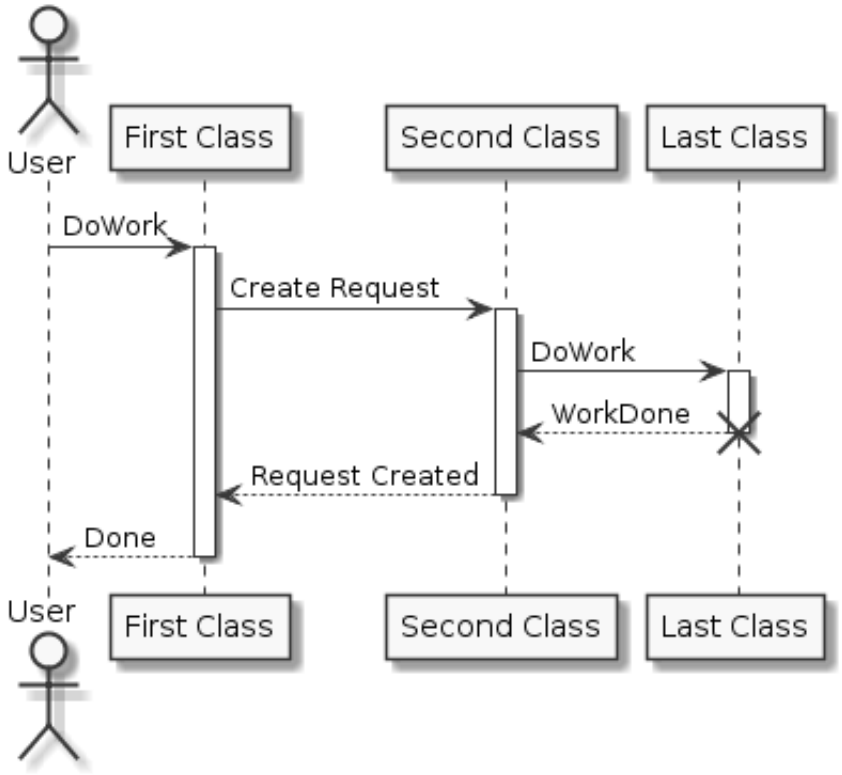
componentArrowFontColor componentArrowFontSize componentArrowFontStyle componentArrowFontName	black 13 plain	コンポーネント図の矢印につけるテキストに使用される
noteFontColor noteFontSize noteFontStyle noteFontName	black 13 plain	シーケンス図以外のすべての図の注釈に使用される
packageFontColor packageFontSize packageFontStyle packageFontName	black 14 plain	パッケージ名とパーティション名に使用される
sequenceActorFontColor sequenceActorFontSize sequenceActorFontStyle sequenceActorFontName	black 13 plain	シーケンス図のアクターに使用される
sequenceDividerFontColor sequenceDividerFontSize sequenceDividerFontStyle sequenceDividerFontName	black 13 bold	シーケンス図の分割線につけるテキストに使用される
sequenceArrowFontColor sequenceArrowFontSize sequenceArrowFontStyle sequenceArrowFontName	black 13 plain	シーケンス図の矢印につけるテキストに使用される
sequenceGroupingFontColor sequenceGroupingFontSize sequenceGroupingFontStyle sequenceGroupingFontName	black 11 plain	シーケンス図の”else” に使用される
sequenceGroupingHeaderFontColor sequenceGroupingHeaderFontSize sequenceGroupingHeaderFontStyle sequenceGroupingHeaderFontName	black 13 plain	シーケンス図のヘッダ”alt/opt/loop” に使用される
sequenceParticipantFontColor sequenceParticipantFontSize sequenceParticipantFontStyle sequenceParticipantFontName	black 13 plain	シーケンス図の分類子につけるテキストに使用される
sequenceTitleFontColor sequenceTitleFontSize sequenceTitleFontStyle sequenceTitleFontName	black 13 plain	シーケンス図のタイトルに使用される
titleFontColor titleFontSize titleFontStyle titleFontName	black 18 plain	シーケンス図を除く、全ての UML 図のタイトルに使用される
stateFontColor stateFontSize stateFontStyle stateFontName	black 14 plain	状態遷移図の状態を表すテキストに使用される
stateArrowFontColor stateArrowFontSize stateArrowFontStyle stateArrowFontName	black 13 plain	状態遷移図の矢印につけるテキストに使用される
stateAttributeFontColor stateAttributeFontSize stateAttributeFontStyle stateAttributeFontName	black 12 plain	状態遷移図の状態を説明するテキストに使用される

usecaseFontColor usecaseFontSize usecaseFontStyle usecaseFontName	black 14 plain	ユースケース図のユースケースにつけるラベルに使用される
usecaseStereotypeFontColor usecaseStereotypeFontSize usecaseStereotypeFontStyle usecaseStereotypeFontName	black 14 italic	ユースケース図のステレオタイプに使用される
usecaseActorFontColor usecaseActorFontSize usecaseActorFontStyle usecaseActorFontName	black 14 plain	ユースケース図のアクターにつけるラベルに使用される
usecaseActorStereotypeFontColor usecaseActorStereotypeFontSize usecaseActorStereotypeFontStyle usecaseActorStereotypeFontName	black 14 italic	アクターのステレオタイプに使用される
usecaseArrowFontColor usecaseArrowFontSize usecaseArrowFontStyle usecaseArrowFontName	black 13 plain	ユースケース図の矢印につけるテキストに適用される
footerFontColor footerFontSize footerFontStyle footerFontName	black 10 plain	フッタに使用される
headerFontColor headerFontSize headerFontStyle headerFontName	black 10 plain	ヘッダに使用される

12.5 白黒

コマンド `skinparam monochrome true` を使用して白黒表示を強制することができます。

```
@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
```



13 プリプロセッシング

PlantUML は、すべてのダイアグラムで利用できるいくつかのマイナーなプリプロセッシング機能を持っています。

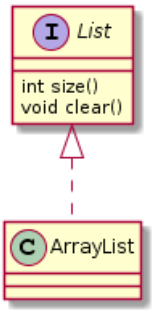
それらの機能は、”#” が、感嘆符”!” であることを除けば、C 言語のプリプロセッサととても良く似ています。

13.1 ファイルのインクルード

ダイアグラム内にファイルをインクルードするには、`!include` ディレクティブを使います。
Imagine you have the very same class that appears in many diagrams. Instead of duplicating the description of this class, you can define a file that contains the description.

全く同じクラスが多数のダイアグラムに登場する状況を想像してみてください。このクラスについての記述を複製する代わりに、記述の内容をファイルに定義することが可能です。

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



File List.iuml: interface List List : int size() List : void clear()

この `List.iuml` ファイルは複数のダイアグラムでインクルード可能です。また、このファイルの変更はインクルードした全てのダイアログに反映されます。

複数の `@startuml/@enduml` ブロックをインクルードファイルに記述することも可能です。`!0` を追加してインクルードすることで、0 番目のブロックをインクルードすることができます。

たとえば、`!include foo.txt!1` と記述した場合、`foo.txt` ファイルの二番目の `@startuml/@enduml` ブロックの内容がインクルードされます。

You can also put an id to some `@startuml/@enduml` text block in an included file using `@startuml(id=MY_OWN_ID)` syntax and then include the block adding `!MY_OWN_ID` when including the file, so using something like `!include foo.txt!MY_OWN_ID`.

13.2 Including URL

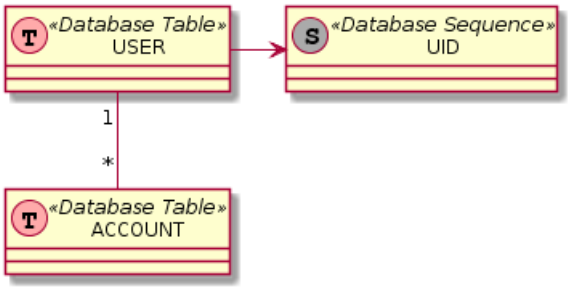
Use the `!includeurl` directive to include file from Internet/Intranet in your diagram.

You can also use `!includeurl http://someurl.com/mypath!0` to specify which `@startuml/@enduml` block from `http://someurl.com/mypath` you want to include. The `!0` notation denotes the first diagram.

13.3 定数を定義する

定数を定義するには `!define` ディレクティブを使います。定数名は言語 C と同様に、英数字とアンダースコアを含むもので、数字で始めることはできません。

```
@startuml

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

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



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

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

You can also specify constants within the command line, with the `-D` flags.

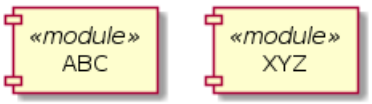
```
java -jar plantuml.jar -DTITLE="My title" atest1.txt
```

Note that the `-D` flag must be put after the "`-jar plantuml.jar`" section.

13.4 Macro definition

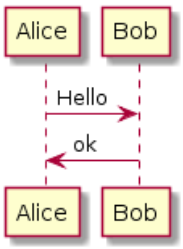
You can also define macro with arguments.

```
@startuml

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



Macro can have several arguments.

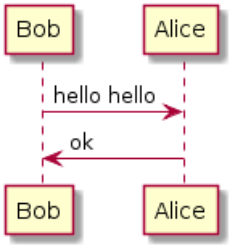
```
@startuml
!define send(a,b,c) a->b : c
send(Alice, Bob, Hello)
send(Bob, Alice, ok)
@enduml
```



13.5 Macro on several lines

You can also define macro on several lines using `!definelong` and `!enddefinelong`.

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

AUTHEN(Bob,Alice)
@enduml
```



13.6 Conditions

You can use `!ifdef XXX` and `!endif` directives to have conditionnal drawings.

The lines between those two directives will be included only if the constant after the `!ifdef` directive has been defined before.

You can also provide a `!else` part which will be included if the constant has **not** been defined.

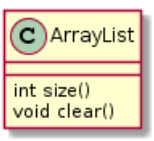
```
@startuml
!include ArrayList.iuml
@enduml
```

File ArrayList.iuml:

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

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

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



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

13.7 Search path

You can specify the java property ”plantuml.include.path” in the command line.

For example:

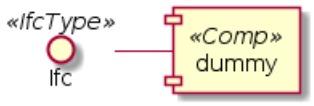
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

Note the this -D option has to put before the -jar option. -D options after the -jar option will be used to define constants within plantuml preprocessor.

13.8 Advanced features

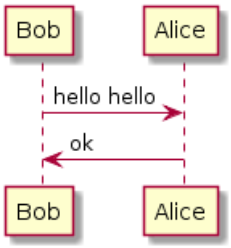
It is possible to append text to a macro argument using the ## syntax.

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



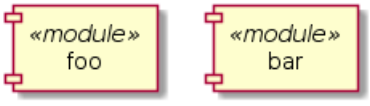
A macro can be defined by another macro.

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



A macro can be polymorphic with argument count.

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



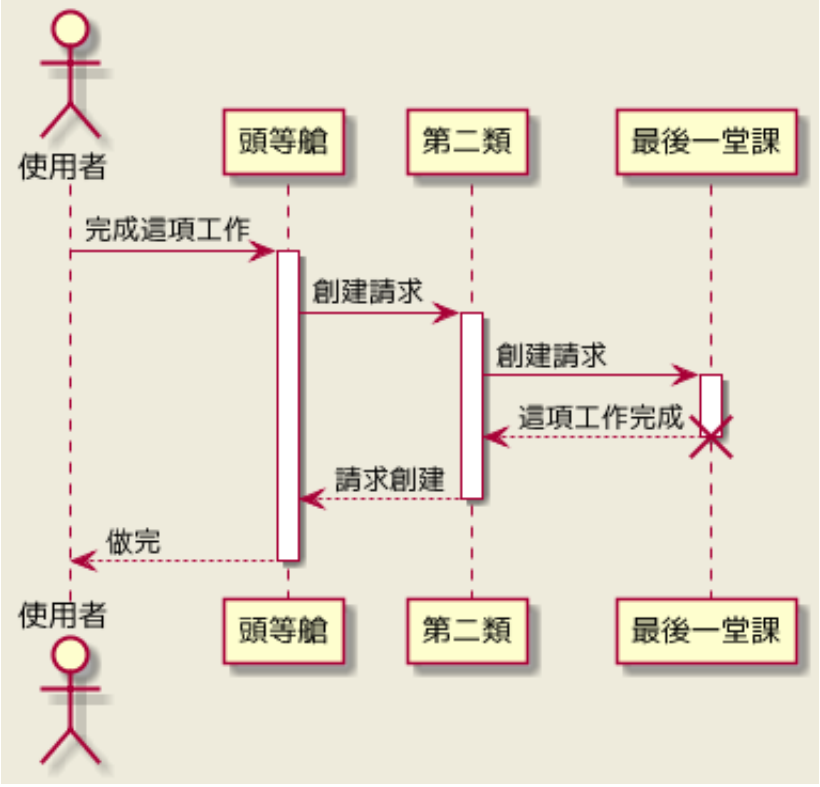
You can use system environment variable or constant definition when using include:

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

14 国際化

PlantUML 言語は、文字を使ってアクターやユースケースを定義することができます。この 文字には A から Z までのラテン文字に限らず、すべての言語のすべての文字を使うことができます。

```
@startuml
skinparam backgroundColor #EEEBDC
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
```



14.1 キャラクターセット

UML の記述を含むテキストファイルが読み込まれるときのキャラクターセットはシステムに依存します。

通常はそれだけで問題ないはずですが、必要にならば別のキャラクターセットを使うこともできます。例えば、コマンドラインに以下のように入力します:

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

または ant タスクを使うこともできます:

```
<target name="main">
<plantuml dir="./src" charset="UTF-8" />
</target>
```

必要に応じて、次に示すキャラクターセットが使えるように Java がインストールされている必要があります: ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.

15　色の名前

ここにある色は PlantUML によって認識されます。色の名前は、大文字と小文字が区別されることに注意してください。

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