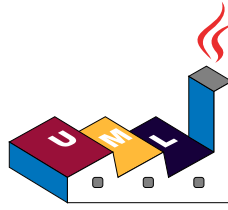


PlantUML を使った UML の描き方



言語リファレンスガイド

(2017 年 4 月 27 日 1:39)

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

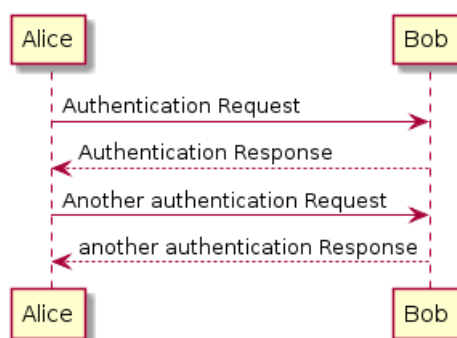
シーケンス”->”を、2つの分類子間のメッセージを描画するために使います。分類子を、明示的に宣言する必要はありません。

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
```



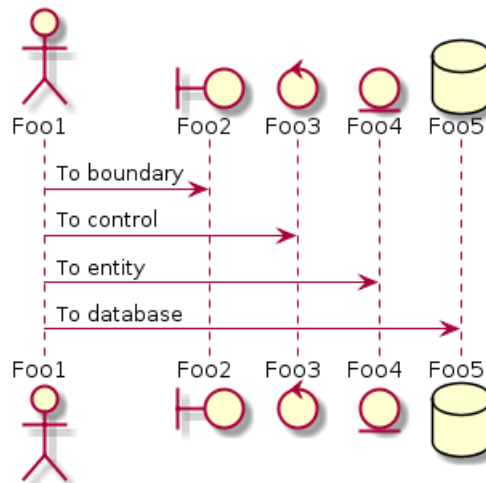
1.2 分類子の宣言

キーワード participant 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
```



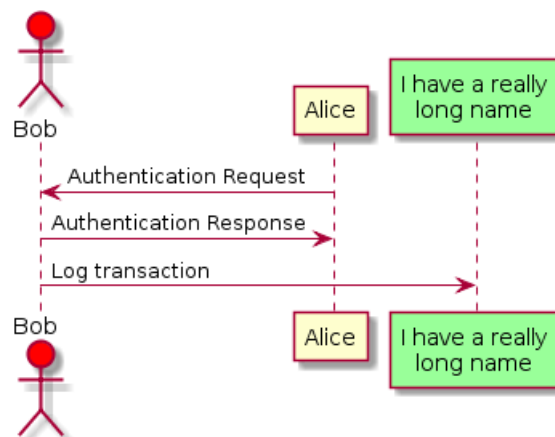
キーワード `as` を使って分類子の名前を変更することができます。
 アクターや分類子の背景色を、HTML コードや色名を使って変更することもできます。

```

@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
  
```

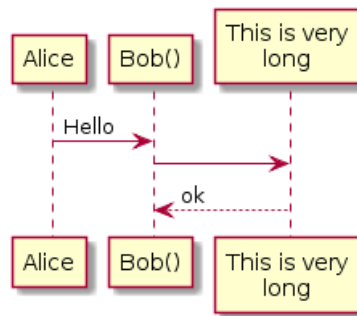


1.3 分類子名にアルファベット以外を使う

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キーワード `as` 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
  
```



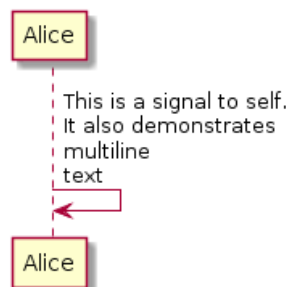
1.4 自分自身へのメッセージ

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
  
```



1.5 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。

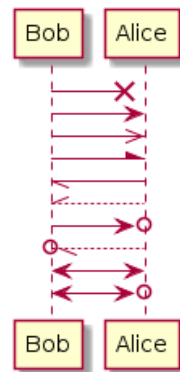
- メッセージの消失を示す最後の x を追加
- \ や / を < や > の代わりに使うと矢印の先端が上側だけまたは下側だけになります。
- 矢印の先端を繰り返す (たとえば >> や //) と、矢印の先端が細くなります。
- -- を - の代わりに使うと、矢印が点線になります。
- 矢じりに最後の "O" を追加
- 双方向の矢印を使用する

```

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

Bob ->o Alice
Bob o\-- Alice

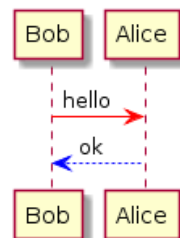
Bob <-> Alice
Bob <->o Alice
@enduml
  
```



1.6 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。

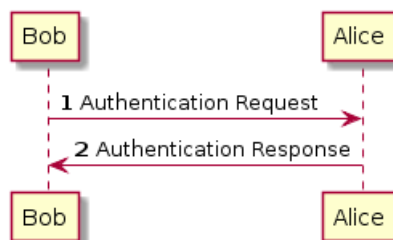
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.7 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、キーワード `autonumber` 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
```

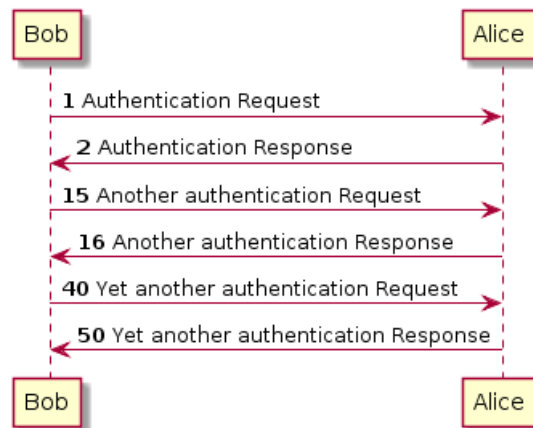


```

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。

```

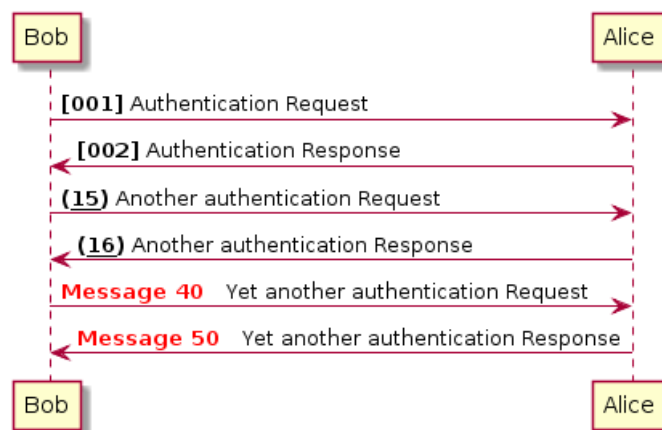
@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

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

```



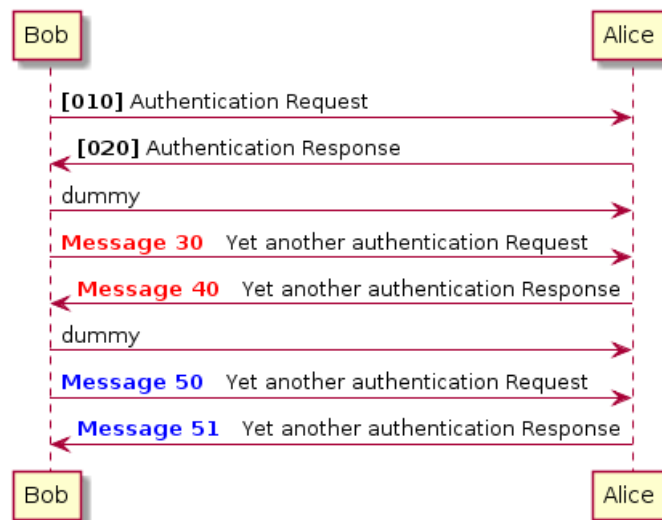
```

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

```



1.8 図の分割

図を複数の画像に分けるためにキーワード `newpage` を使います。

新しいページのタイトルをキーワード `newpage` 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

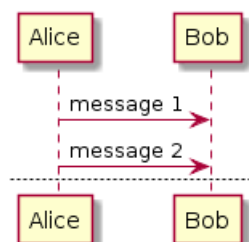
newpage

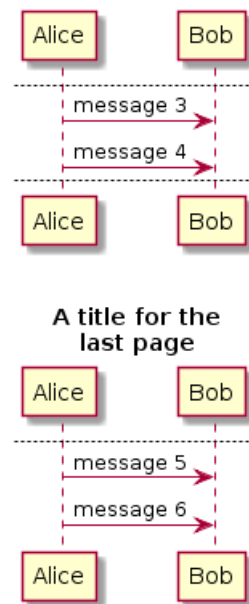
Alice -> Bob : message 3
Alice -> Bob : message 4

newpage A title for the\last page

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

```





1.9 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

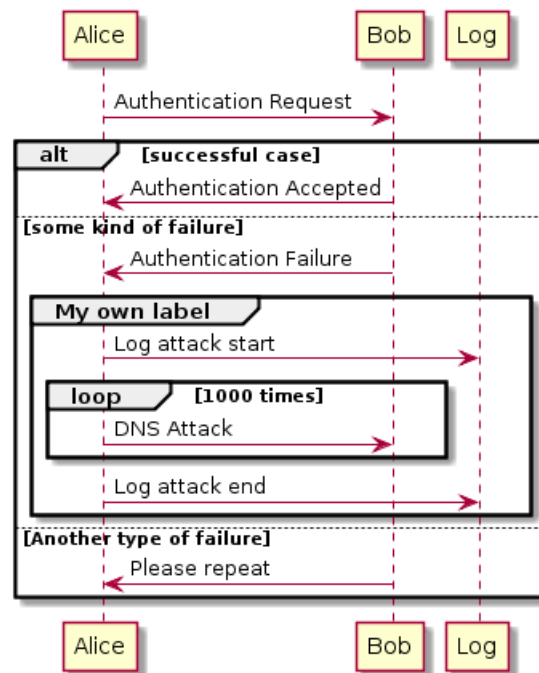
ヘッダ部分に文字列を追加することが可能です。(group を除く)
グループを閉じるにはキーワード end を使用します。

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
  
```

1.10 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。

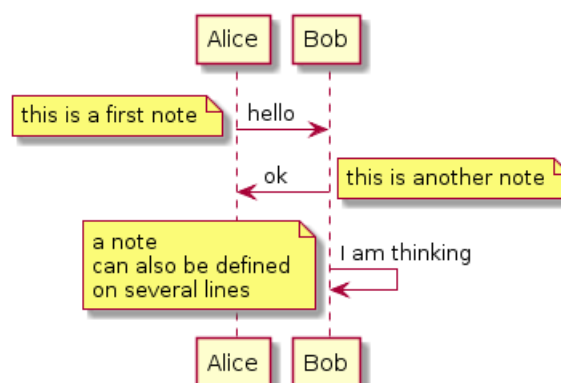
`end note` キーワードを使って、複数行の注釈を付けることができます。

```

@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
  
```



1.11 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、キーワード `end note` 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

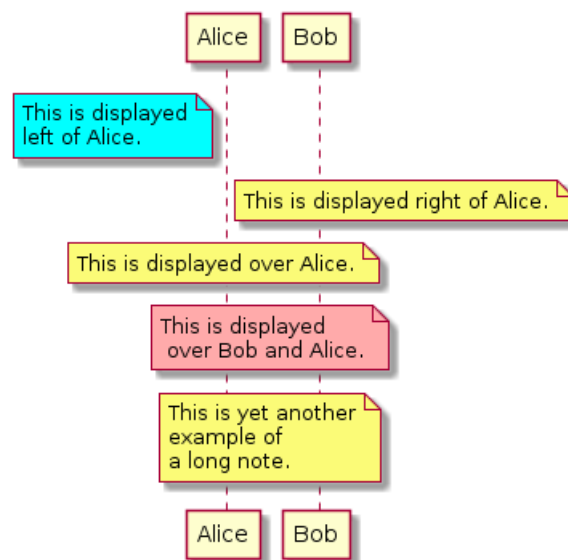
note right of Alice: This is displayed right of Alice.

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

```



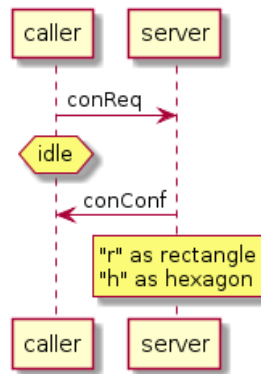
1.12 ノートの形を変える。

キーワード `hnote` と `rnote` 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
"x" as rectangle
"h" as hexagon
endrnote
@enduml

```



1.13 Creole と HTML

PlantUML では creole フォーマットを使うこともできます。

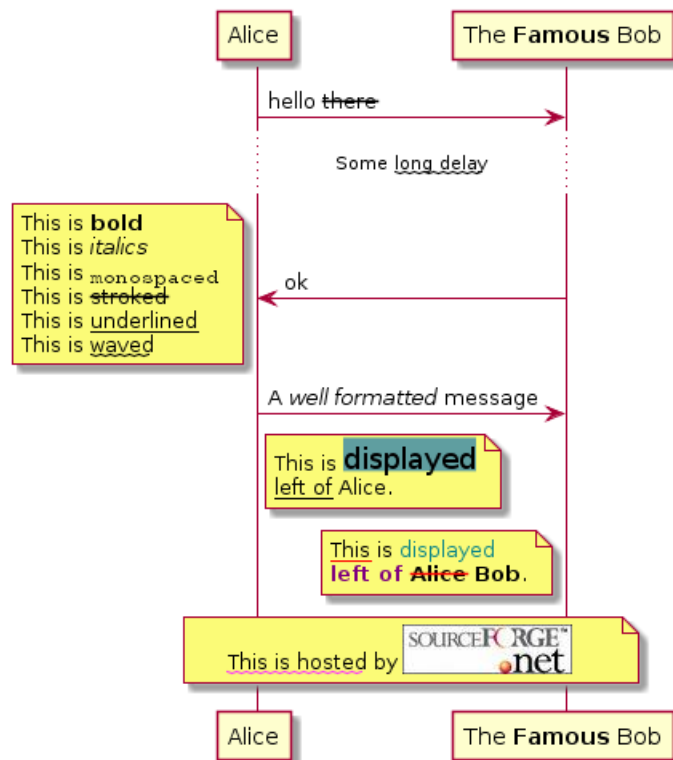
```

@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

```



1.14 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
```

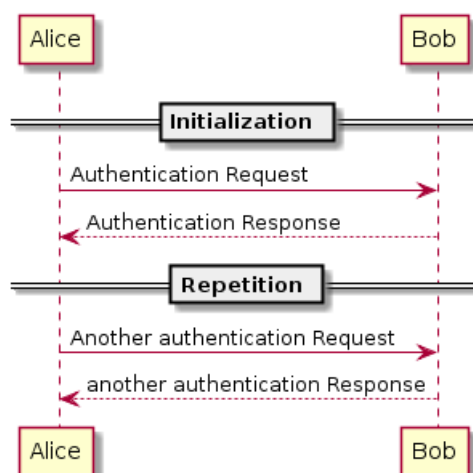
```
== Initialization ==
```

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

```
== Repetition ==
```

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

```
@enduml
```



1.15 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。

```

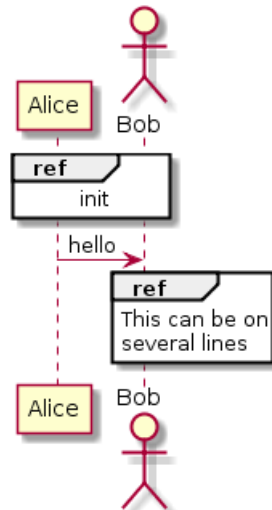
@startuml
participant Alice
actor Bob

ref over Alice, Bob : init

Alice -> Bob : hello

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

```



1.16 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。

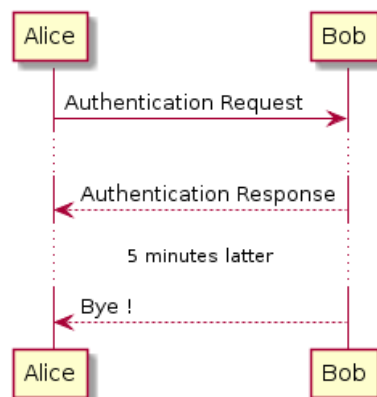
```

@startuml

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

@enduml

```



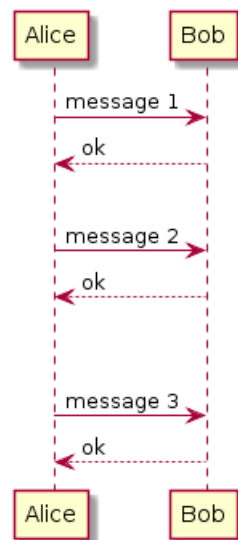
1.17 間隔

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

```



1.18 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

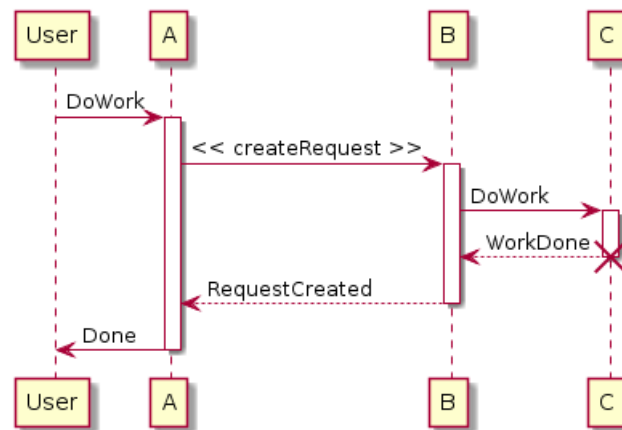
A -> B: << createRequest >>
activate B

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

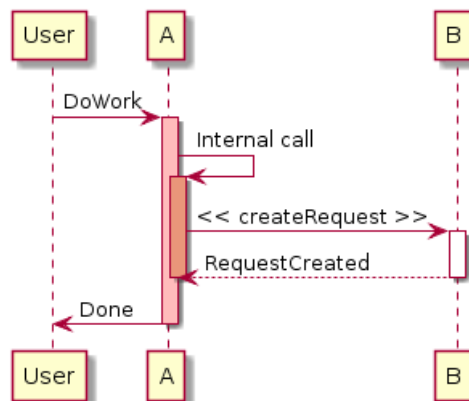
A -> A: Internal call
activate A #DarkSalmon

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

B --> A: RequestCreated
deactivate B
deactivate A
A -> User: Done
deactivate A

@enduml

```



1.19 分類子の生成

キーワード `create` 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!

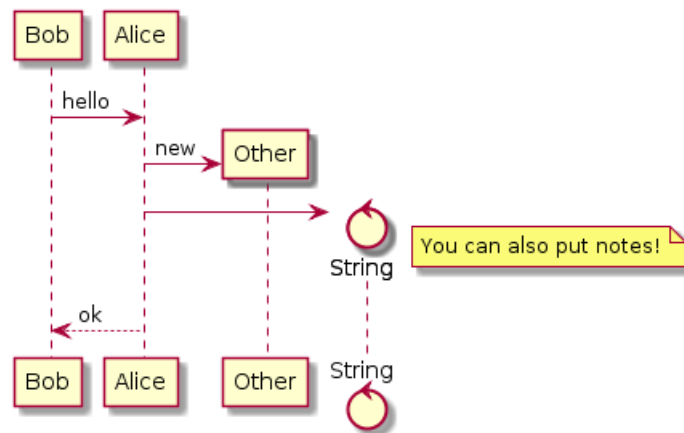
```



```

Alice --> Bob : ok
@enduml

```



1.20 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

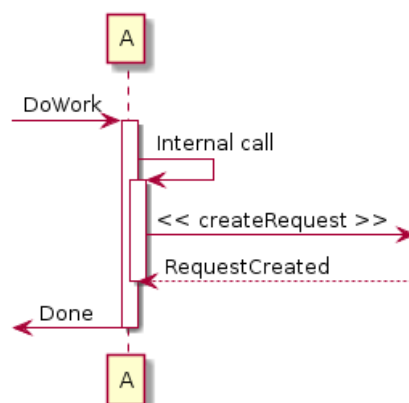
activate A

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->> Bob
[x-> Bob

[<- Bob
[x<- Bob

```



```

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

Bob <-]
Bob x<-]
@enduml

```



1.21 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。

```

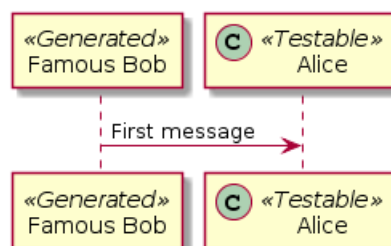
@startuml

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：

```

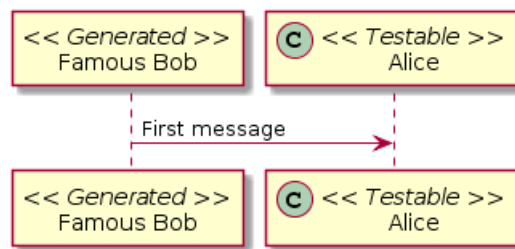
@startuml

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

Bob->>Alice: First message

@enduml

```



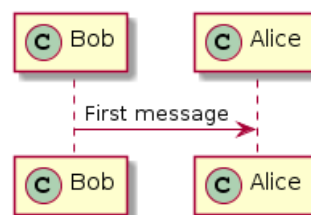
```

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

Bob->>Alice: First message

@enduml

```



1.22 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。

```

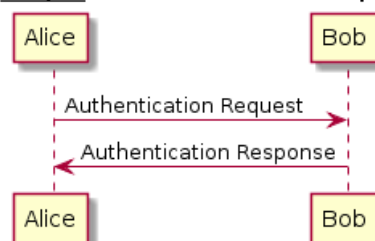
@startuml
title __Simple__ **communication** example

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

@enduml

```

Simple communication example



タイトルの説明では \n で改行を表せます。

```

@startuml
title __Simple__ communication example\nnon several lines

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

@enduml

```

Simple communication example on several lines



また、キーワード `title` と `end title` 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。

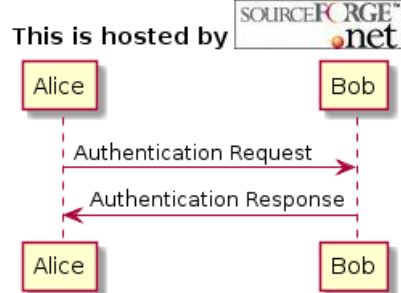
```
@startuml

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
```

Simple communication example on several lines and using html



1.23 分類子の囲み

キーワード `box` と `end box` 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。

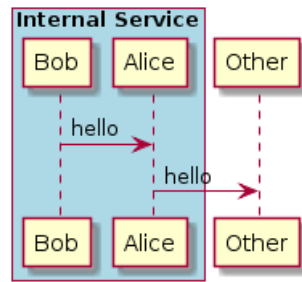
タイトルや背景色をキーワード `box` に続けて任意で追加できます。

```
@startuml

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

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

@enduml
```



1.24 フッターの除去

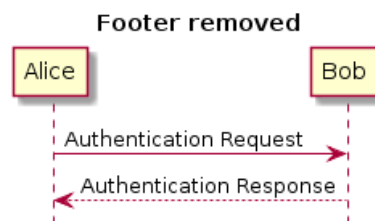
図のフッターを削除するにはキーワード `hide footbox` を使います。

```

@startuml
hide footbox
title Footer removed

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

@enduml
  
```



1.25 スキンパラメータ

図の色やフォントを変えるにはキーワード `skinparam` を使います。

キーワー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
  
```



```

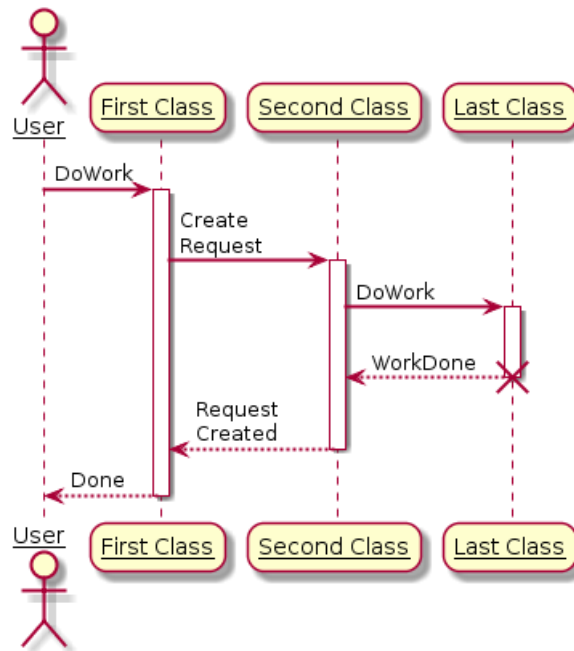
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



```

@startuml
skinparam backgroundColor #EEEEDC
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
deactivate C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```

```

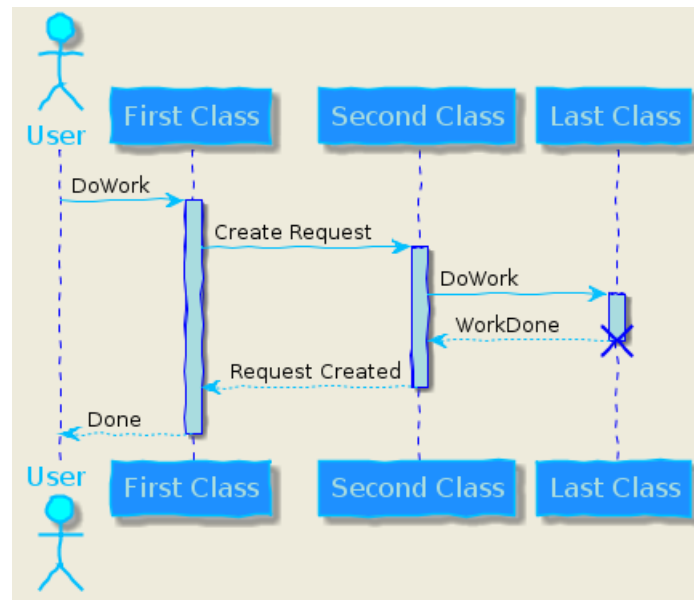
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



1.26 パディングの変更

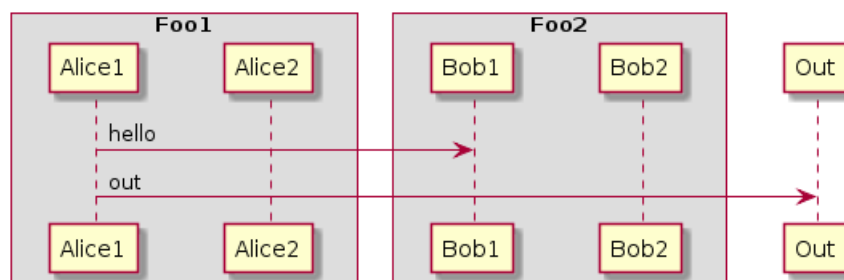
パディングの設定を変更することができます。

```

@startuml
skinparam ParticipantPadding 20
skinparam BoxPadding 10

box "Foo1"
participant Alice1
participant Alice2
end box
box "Foo2"
participant Bob1
participant Bob2
end box
Alice1 -> Bob1 : hello
Alice1 -> Out : out
@enduml

```



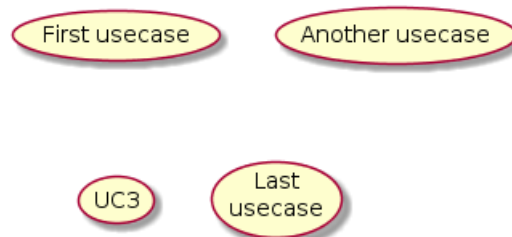
2 ユースケース図

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
```



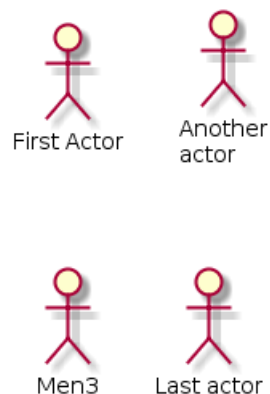
2.2 アクター

アクターは2つのコロンので囲まれます。

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。

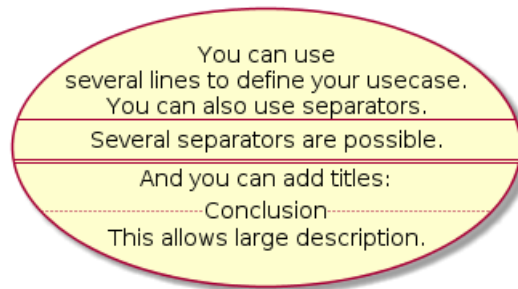
```

@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。

```

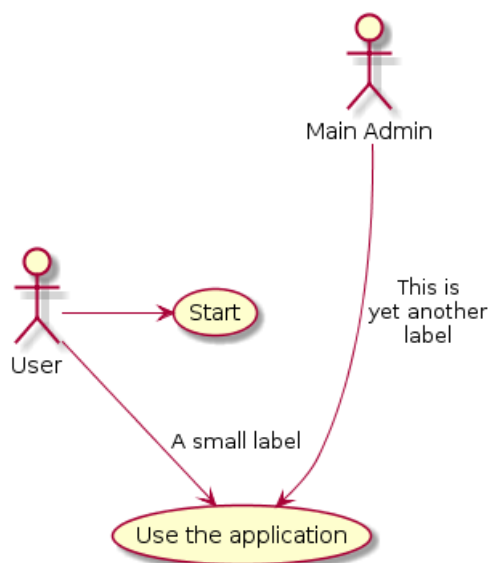
@startuml

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


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

@enduml

```



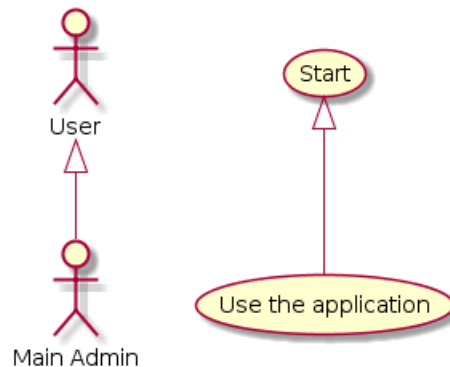
2.5 継承

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)

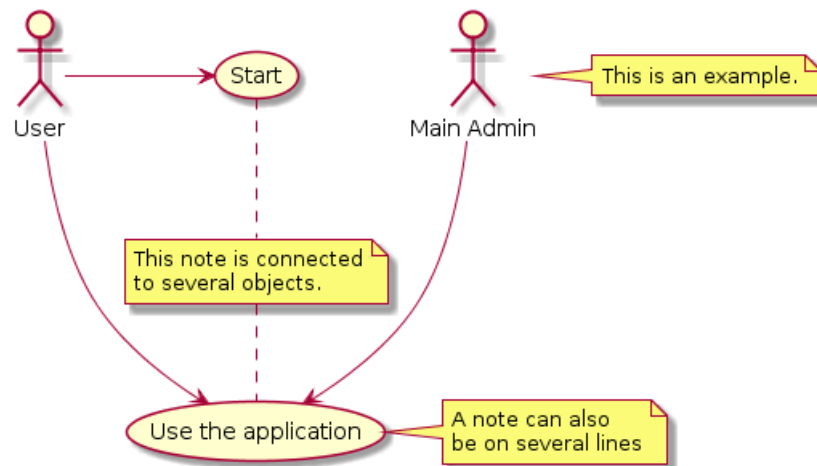
Admin ---> (Use)

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

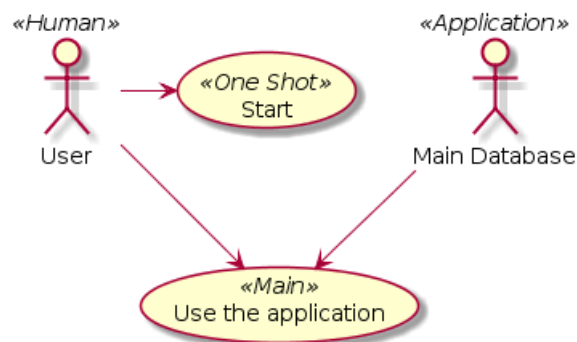
” << ” と ” >> ” を使い、アクターとユースケースを定義中にステレオタイプを追加できます。

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

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

MySql --> (Use)

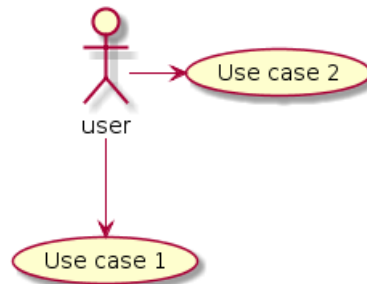
@enduml
```



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。

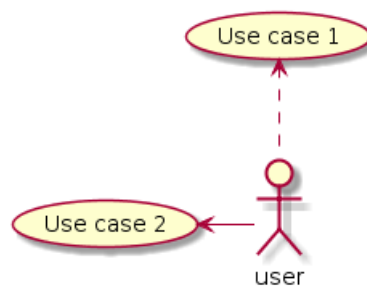
```
@startuml
:user: --> (Use case 1)
:user: -> (Use case 2)
@enduml
```



線を反対にすることも方向を変えることができます。

```

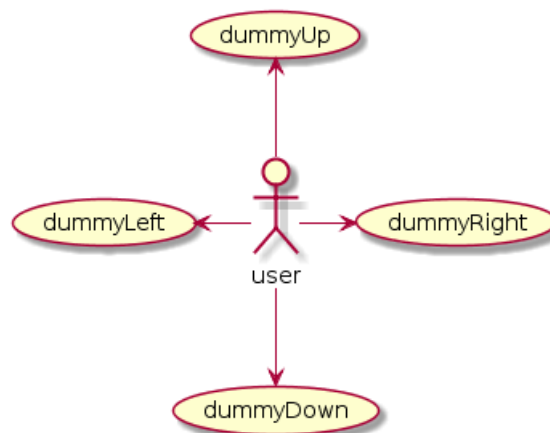
@startuml
  (Use case 1) <.. :user:
  (Use case 2) <- :user:
@enduml
  
```



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

```

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



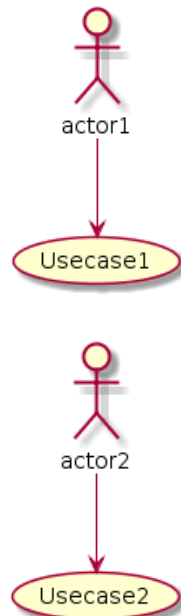
例えば、-down- ではなく -d- など、各方向の頭文字、または頭 2 文字 (-do-) だけ使って矢印を短くすることも出来ます。

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。

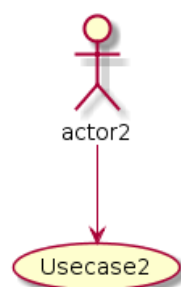
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



2.10 左から右に描画する

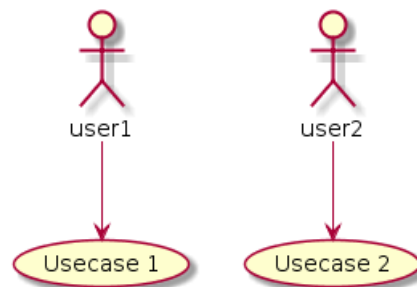
デフォルトの作図方向は top to bottom となっています。

```
@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
```



2.11 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

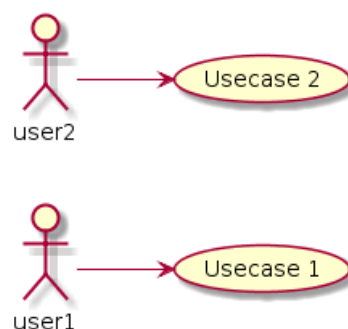
ActorBackgroundColor<< Human >> Gold
}

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

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

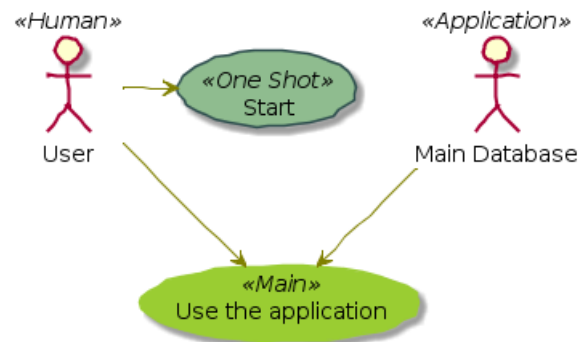
MySql --> (Use)

@enduml
```



2.12 完全な例




```
@startuml
left to right direction
skinparam packageStyle rectangle
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

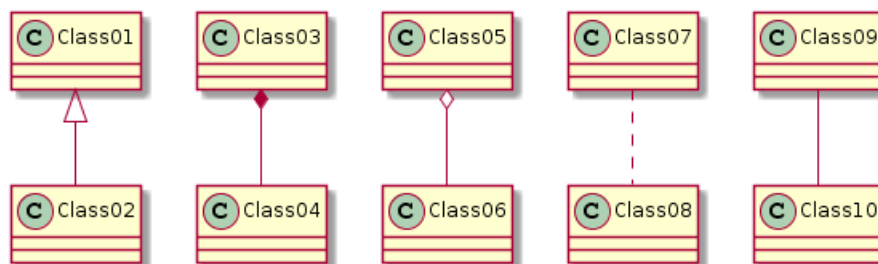
クラス間の関係は次の記号を使用して定義されています:

継承	< --	
合成	*--	
集約	o--	

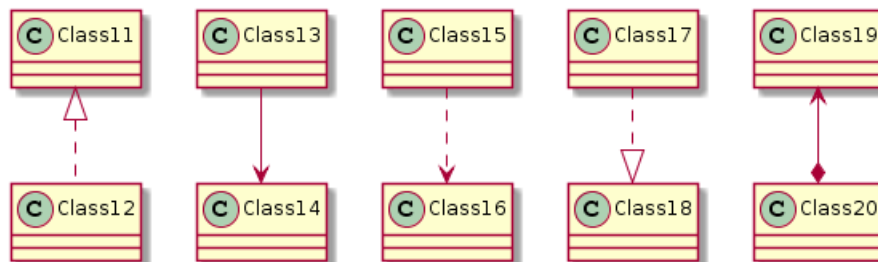
”--”を”..”に置き換えると点線にできます。

これらのルールを知ることによって、以下の図面を描くことができます:

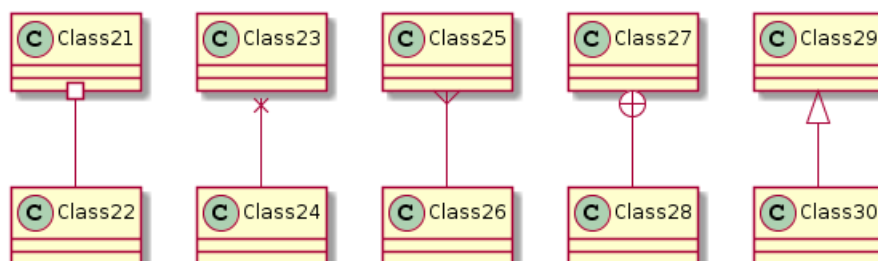
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```



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



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

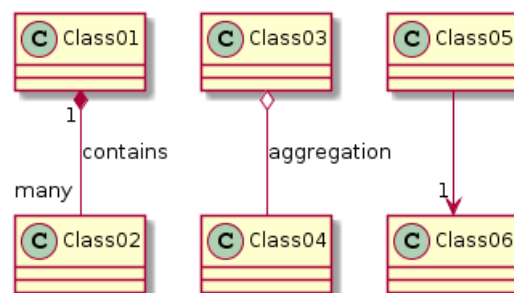


3.2 関係のラベル

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

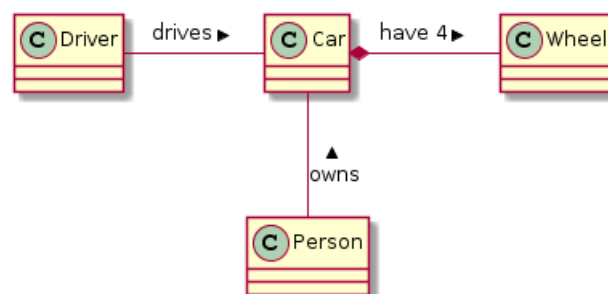
多重度を示す為に関係のそれぞれの側にダブルクォーテーション"\" を使うことができます。

```
@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係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

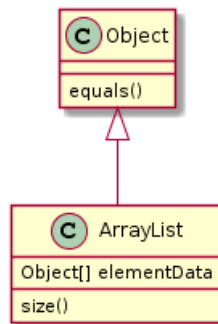
":" に続けてフィールド名やメソッド名を記述すると、フィールドやメソッドを宣言できます。

システムは括弧をチェックしてメソッドとフィールドのどちらなのかを選択します。

```
@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ができます。

構文はタイプや名前の順番について非常に柔軟であることに注意してください。

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

class Flight {
flightNumber : Integer
departureTime : Date
}

@enduml
```

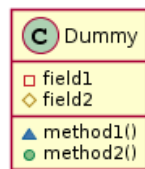


3.4 可視性の定義

メソッドやフィールドを定義するときに対応する項目の可視性を定義する記号を使用することができます。

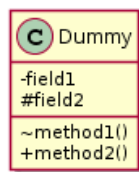
-	□	private (クラス外からは非可視)
#	◇	protected (サブクラスからも可視)
~	△	package private (パッケージ外からは非可視)
+	○	public (どこからでも可視)

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



コマンド `skinparam classAttributeIconSize 0` を使用してこの機能を切ることができます。

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

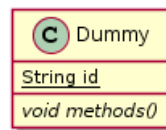


3.5 Abstract と Static

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。

これらの修飾子は行の始めまたは終りに使用することができます。{static} の代わりに {classifier} もまた使用できます。

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



3.6 高等なクラス本体

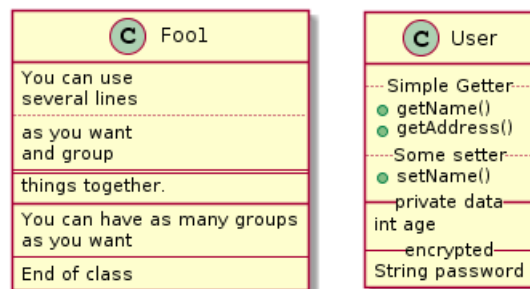
デフォルトでは、メソッドやフィールドは PlantUML によって自動再編成されます。メソッドやフィールドに独自の順序付けを定義するためのセパレータを使用できます。以下のセパレータが使用できます：-- .. == __

セパレータ内でタイトルを使用することもできます：

```
@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 注釈とステレオタイプ

ステレオタイプは、キーワード `class` に” << ” と” >> ” で定義されます。

注釈の定義には、キーワード `note left of`, `note right of`, `note top of`, `note bottom of` も使用できます。

クラス定義の最後には `note left`, `note right`, `note top`, `note bottom` も使用できます。

注釈は、キーワード `note` とで単独に定義することができ、記号 `..` を使用して他のオブジェクトとリンクすることもできます。

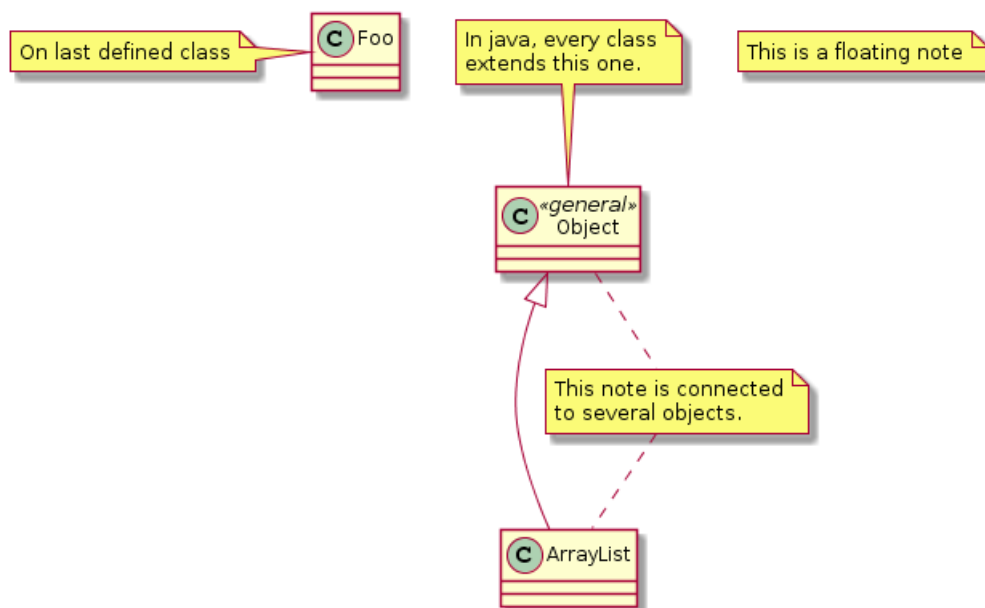
```
@startuml
class Object << general >>
Object <|--- ArrayList

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 注釈の詳細

次のようないくつかの HTML タグを使用することも可能です：

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

また、複数行にまたがる注釈も可能です。

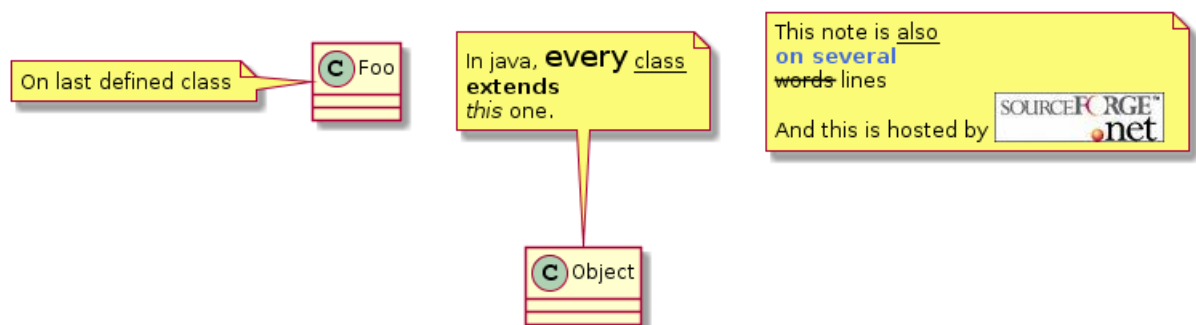
クラス定義の最後には `note left`, `note right`, `note top`, `note bottom` も使用できます。

```
@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
```



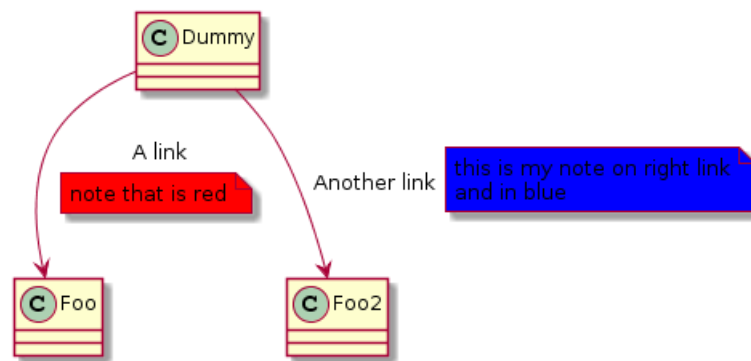
3.9 リンクへの注釈

リンク定義の直後に `note on link` を使用して、リンクに注釈を加えることが可能です。

もし注釈の相対位置を変えたい場合には、ラベル `note left on link`, `note right on link`, `note top on link`, `note bottom on link` も使用できます。

```
@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 抽象クラスとインタフェース

抽象クラスは、キーワード "abstract" または "abstract class" を使用して宣言できます。

そのクラスはイタリック体で印字されます。

キーワード interface, annotation と enum も使用できます。

```
@startuml
```

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

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

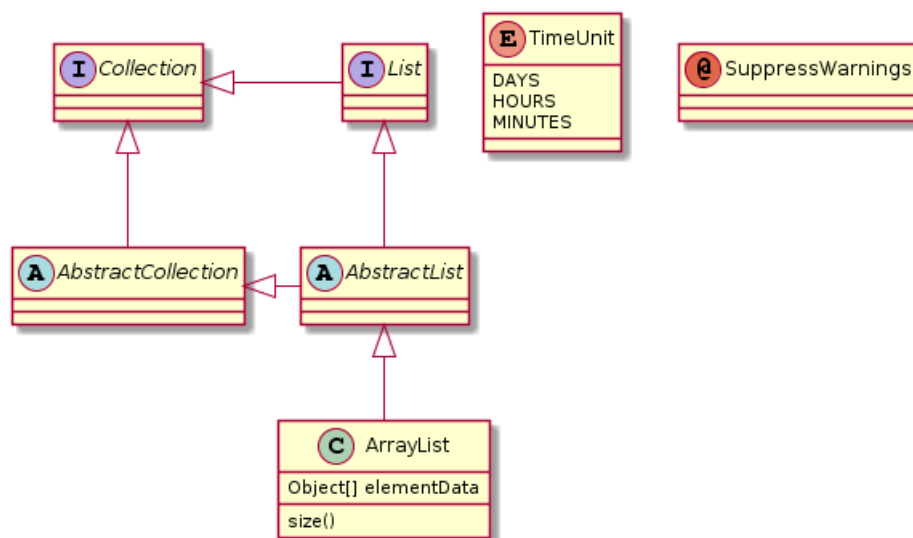
```
Collection <|-- List
AbstractCollection <|-- AbstractList
AbstractList <|-- ArrayList
```

```
class ArrayList {
Object[] elementData
size()
}
```

```
enum TimeUnit {
DAYS
HOURS
MINUTES
}
```

```
annotation SuppressWarnings
```

```
@enduml
```



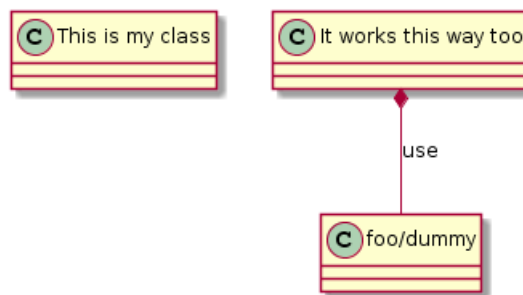
3.11 非文字の使用

クラス（または列挙型...）の表示に文字以外を使用したい場合は、次のいずれかの方法ですることができます：

- クラス定義にはキーワード `as` を使用する
- クラス名の前後に引用符 `"` を入れる

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

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



3.12 属性、メソッド等の非表示

コマンド `hide/show` を使用して、クラスを表示をパラメータ化できます。

基本のコマンドは `hide empty members` です。このコマンドは属性やメソッドが空の場合に非表示にします。

`empty members` の代わりに使用することができます：

- `empty fields` または `empty attributes` は空のフィールドに、
- `empty methods` は空のメソッドに、
- `fields` または `attributes` は、それらが記述されていても非表示になります、
- `methods` はメソッドが記述されていても非表示になります、
- `members` はフィールドとメソッドが記述されていても非表示になります、
- `circle` はクラス名の前の丸で囲んだ文字に、
- `stereotype` はステレオタイプに。

キーワード `hide` または `show` のすぐ後ろに提供することもできます：

- `class` は全てのクラスに、
- `interface` は全てのインタフェースに、
- `enum` は全ての列挙型に、
- `<<foo1>>` は `foo1` でステレオタイプ化されたクラスに、
- 既存のクラス名。

コマンド `show/hide` をルールや例外の定義にそれぞれ使用することができます。

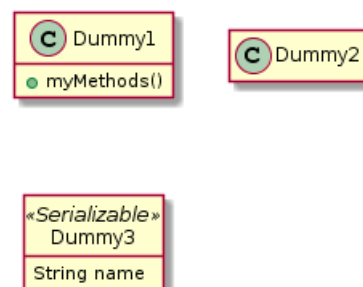
```
@startuml
class Dummy1 {
+myMethods()
}

class Dummy2 {
+hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

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

@enduml
```



3.13 非表示クラス

コマンド `show/hide` でクラスを非表示にすることができます。

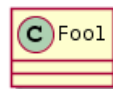
これは大規模なインクルードファイルを定義する場合で、ファイルのインクルードの後でいくつかのクラスを非表示にしたい場合に有用である可能性が有ります。

```
@startuml
class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```

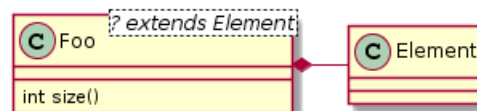


3.14 ジェネリクスの使用

括弧 `<` と `>` を使用してジェネリクスの使用をクラスに定義できます。

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

@enduml
```



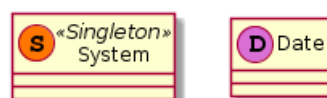
3.15 特殊な目印

通常、目印文字 (C,I,E,A) は、クラス、インターフェイス、列挙型と抽象クラスのために使用されます。

しかし、つぎの例のように単一の文字と色を追加し、ステレオタイプを定義するクラスに独自の目印を作成することができます：

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

@enduml
```



3.16 パッケージ

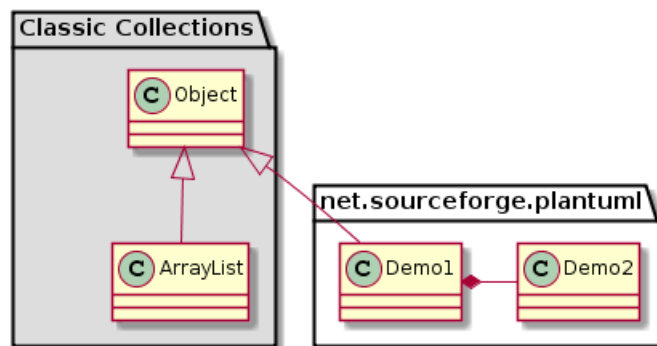
キーワード `package` を使用してパッケージを定義でき、必要に応じてパッケージの背景色 (HTML カラーコードまたは名前) を宣言します。

パッケージ定義は入れ子にできることに注意してください。

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

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

@enduml
```



3.17 パッケージスタイル

パッケージに利用可能なさまざまなスタイルがあります。

コマンド `skinparam packageStyle` を使用してデフォルトのスタイルを設定する、またはパッケージのステレオタイプを使用する、のどちらかで指定することができます。or by using a stereotype on the package:

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

package foo2 <<Rectangle>> {
class Class2
}

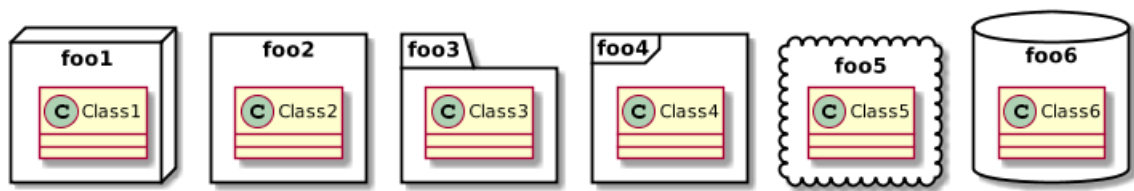
package foo3 <<Folder>> {
class Class3
}

package foo4 <<Frame>> {
class Class4
}

package foo5 <<Cloud>> {
class Class5
}

package foo6 <<Database>> {
class Class6
}

@enduml
```



次の例のように、パッケージ間のリンクを定義することもできます：

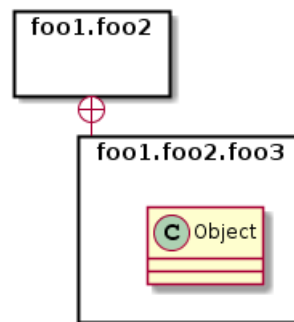
```
@startuml
skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 名前空間

パッケージ内では、クラスの名前はこのクラスの一意的識別子です。それは、全く同じ名前の2つのクラスを異なるパッケージに持つことができないことを意味します。

そのような場合、パッケージの代わりに名前空間を使用したらいいでしょう。

名前空間からの完全修飾名によりクラスを参照することができます。デフォルトの名前空間からのクラスは、一つのドットで修飾します。

明示的に名前空間を作成する必要はないことに注意してください：完全修飾されたクラスは自動的に適切な名前空間に置かれています。

```
@startuml
class BaseClass

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

.BaseClass <|-- Meeting
}

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



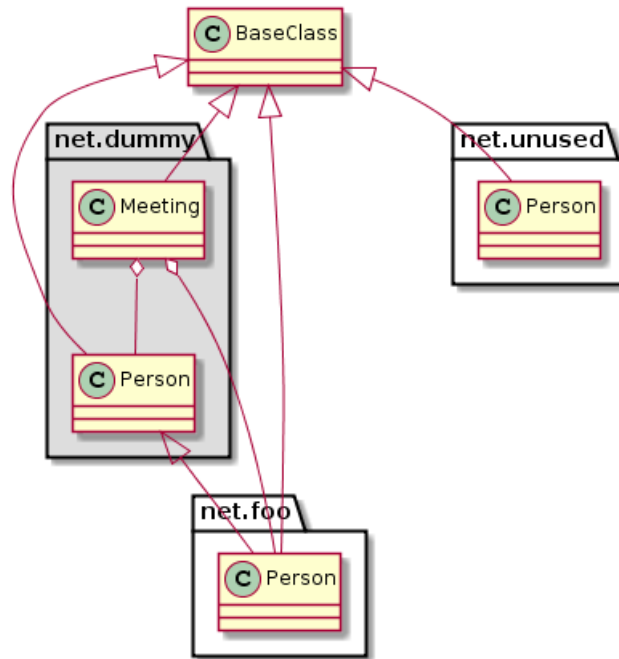
```

net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml

```



3.19 自動的に名前空間を作成する

コマンド `set namespaceSeparator ???` を使用して、(ドット以外の) 別の区切り文字を定義することができます。

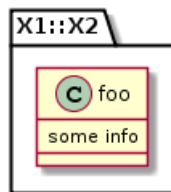
```

@startuml

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

@enduml

```



コマンド `set namespaceSeparator none` を使用して、自動的に名前空間を作成する機能を無効にすることができます。

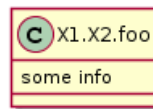
```

@startuml

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

@enduml

```



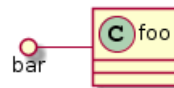
3.20 ロリポップ（棒付きキャンディー）インタフェース

次の構文を使用して、クラスにロリポップインタフェースを定義することもできます：

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

```

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

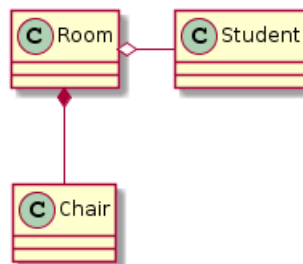


3.21 矢印の向きを変える

デフォルトではクラス間のリンクは2つのダッシューを持っており、垂直に配向されています。次のように単一のダッシュ（またはドット）を置くことによって水平方向にリンクを使用することが可能です。

```

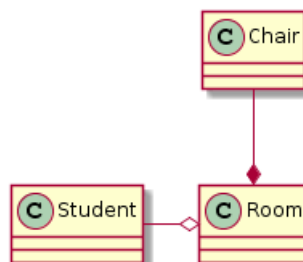
@startuml
Room o- Student
Room *-- Chair
@enduml
  
```



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：

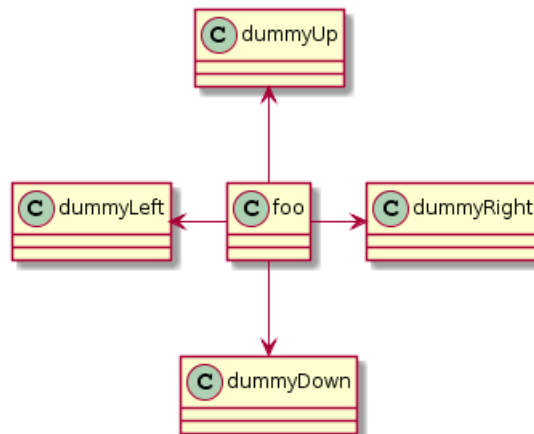
```

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



キーワード `left`, `right`, `up`, `down` を矢印の内側に置くことにより、矢印の方向を変えることも可能です：

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



方向の最初の文字を使用して矢印を短縮することができます（例えば、`-d-` を `-down-` の代わりに、または、最初の 2 文字 (`-do-`)。

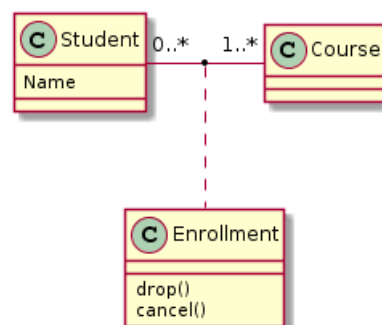
この機能を悪用してはならないことに注意してください。 *Graphviz* は微調整のいない良い結果を通常は与えてくれます。

3.22 関連クラス

この例のように、2 つのクラスの関係を定義した後で 関連クラスを定義することができます。

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

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



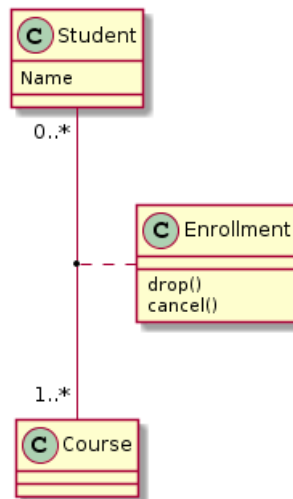
別の方向にそれを定義することができます：


```

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

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

```



3.23 化粧をする

色やフォントを変えるために skinparam コマンドを使用することができます。

次のようにコマンドは使用できます：

- 図の定義では、他のコマンドのように、
- インクルードファイルの中に、
- 設定ファイルの中に、コマンドラインまたは ANT タスクで提供。

```

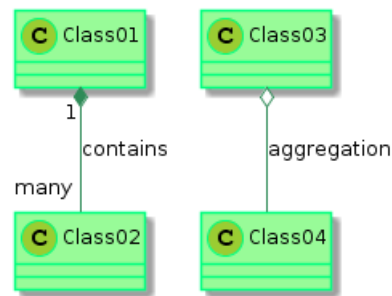
@startuml
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

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

Class03 o-- Class04 : aggregation

@enduml

```



3.24 ステレオタイプの化粧

ステレオタイプクラスに特定の色やフォントを定義することができます。

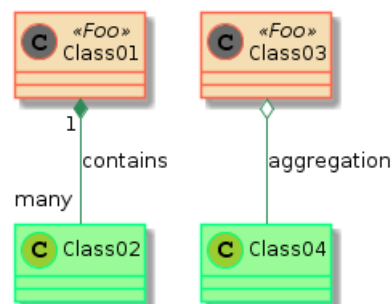
```

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

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

Class03 o-- Class04 : aggregation

@enduml
  
```



3.25 色のグラデーション

表記を使用して、クラスや注釈の個々の色を宣言することが可能です。

標準的な色の名前または RGB コードのいずれかを使用することができます。

次の構文で背景に色のグラデーションをつけることもできます。2 つの色の名前を次のいずれかで分割：

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

グラデーションの方向に依存します。

例えば、こんなふうに行けるかも：

```

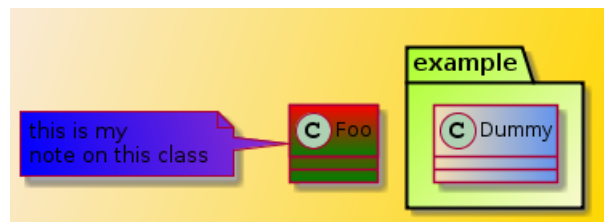
@startuml
skinparam backgroundcolor AntiqueWhite/Gold
skinparam classBackgroundColor Wheat|CornflowerBlue

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

package example #GreenYellow/LightGoldenRodYellow {
class Dummy
}

@enduml

```



3.26 レイアウトの手助け

Sometimes, the default layout is not perfect...

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

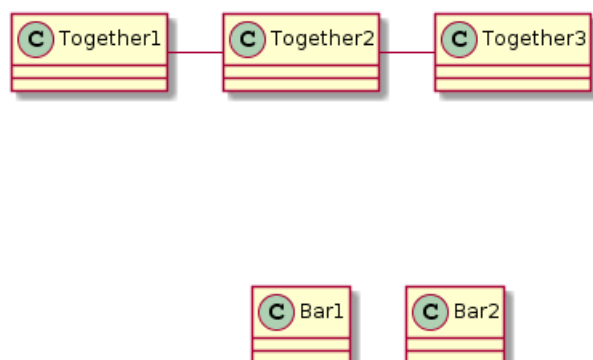
You can also use **hidden** links to force the layout.

```

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

@enduml

```



3.27 大きなファイルの分割

時には、ある非常に大きな画像ファイルを受け取ることがあるでしょう。

生成された画像を複数のファイルに分割するコマンド”page (hpages)x(vpages)”を使用することができます：

hpages は横方向のページ数を示すコマンドであり、そして vpages は縦方向のページ数を示すコマンドです。

特定のスキンパラメータ設定を使用して、分割されたページに罫線を配置することもできます(例を参照)。

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

class BaseClass

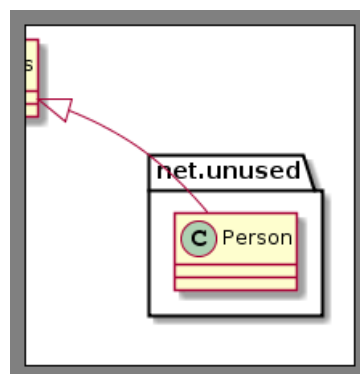
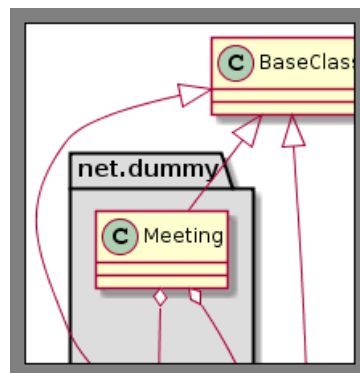
namespace net.dummy #DDDDDD {
.BaseClass <|-- Person
.Meeting o-- Person

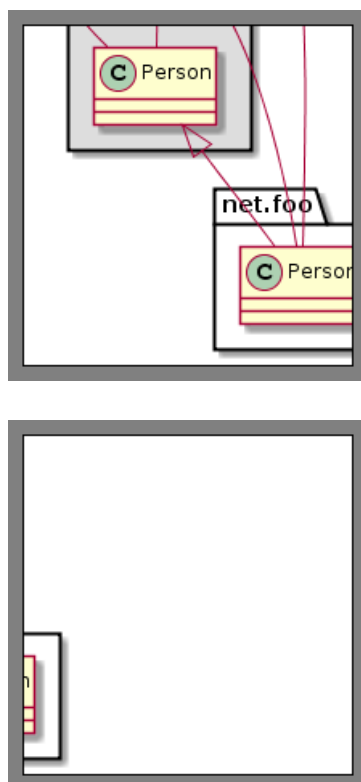
.BaseClass <|-- Meeting
}

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

net.dummy.Meeting o-- Person
}

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





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用します。

場合によっては、(*top) を使用して開始点を図の一番上に置くこともできます。

--> で矢印を表します。

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

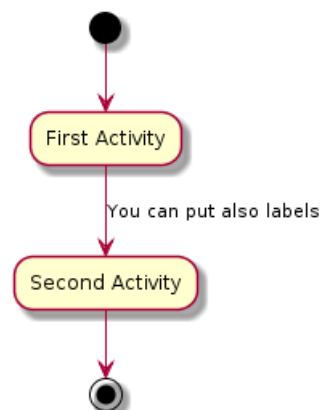


4.2 矢印のラベル

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

水平矢印には -> を使用できます。次の構文を使用して矢印の方向を強制することができます。

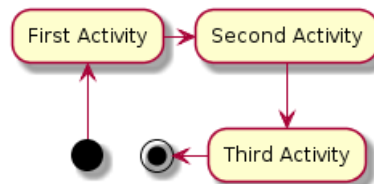
- -down-> (default arrow)
- -right-> or ->

```

• -left->
• -up->

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

```



4.4 分岐

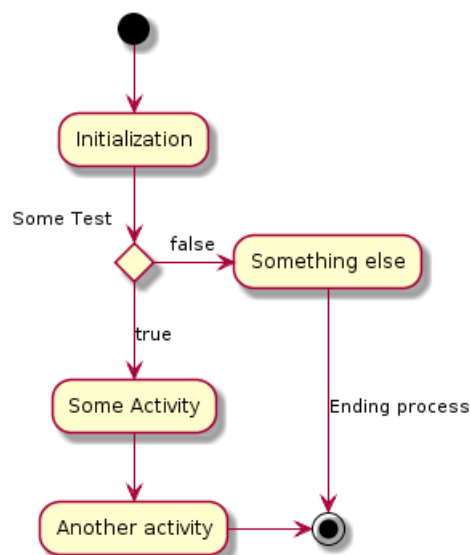
キーワード `if/then/else` を使用してブランチを定義することができます。

```

@startuml
(*) --> "Initialization"

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

```



残念ながら、図のテキストで同じアクティビティを繰り返すことがあります：

```

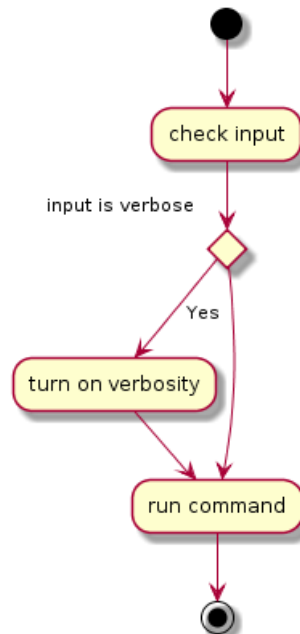
@startuml
(*) --> "check input"
If "input is verbose" then
--> [Yes] "turn on verbosity"

```

```

--> "run command"
else
--> "run command"
Endif
-->(*)
@enduml

```



4.5 もっと分岐

デフォルトでは、分岐は最後に定義されたアクティビティに接続されますが、これを上書きしてキーワード `if` でリンクを定義することは可能です。

分岐をネストすることも可能です。

```

@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

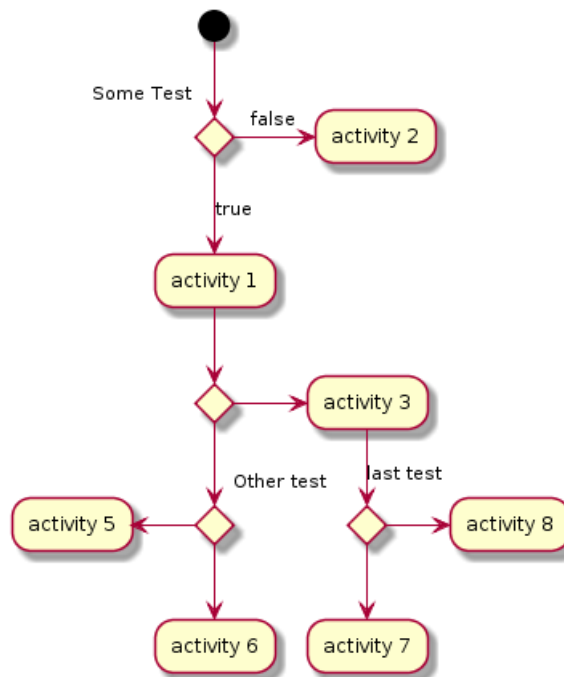
else
->[false] "activity 2"

endif

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

@enduml

```

4.6 同期

=== code === を使用して同期バーを表示できます。

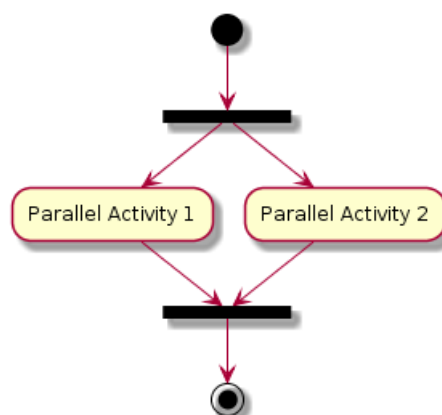
```

@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

アクティビティを宣言するとき、説明文を複数の行にまたがせることができます。説明に \n を追加することもできます。

キーワード as を使ってアクティビティに短いコードを与えることもできます。このコードは、図の説明の後半で使用できます。

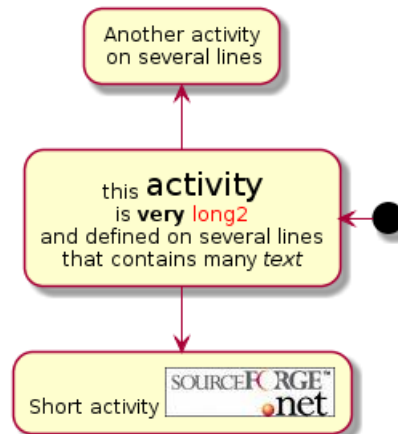
```

@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 注釈

注釈をつけるアクティビティの説明の直後にあるコマンド `note left`, `note right`, `note top` or `note bottom`, を使用して、アクティビティに注釈を追加することができます。

開始点に注釈を付ける場合は、図の説明の最初に注釈を定義します。

キーワード `endnote` を使用して、複数の行に注釈を付けることもできます。

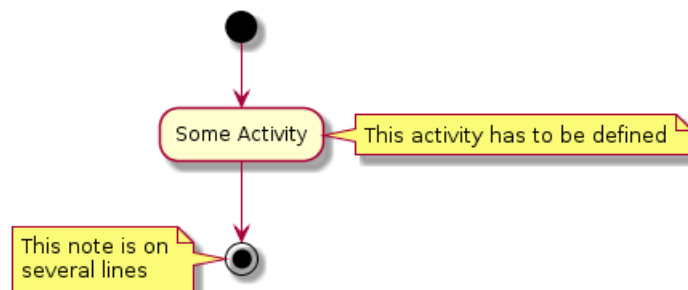
```

@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 パーティション

キーワード `partition` を使用してパーティションを定義し、必要に応じてパーティションの背景色を宣言することができます (HTML カラーコードまたは名前を使用)。

アクティビティを宣言すると、自動的に最後に使用されたパーティションに配置されます。
 閉じ括弧 } を使用してパーティション定義を閉じることができます。

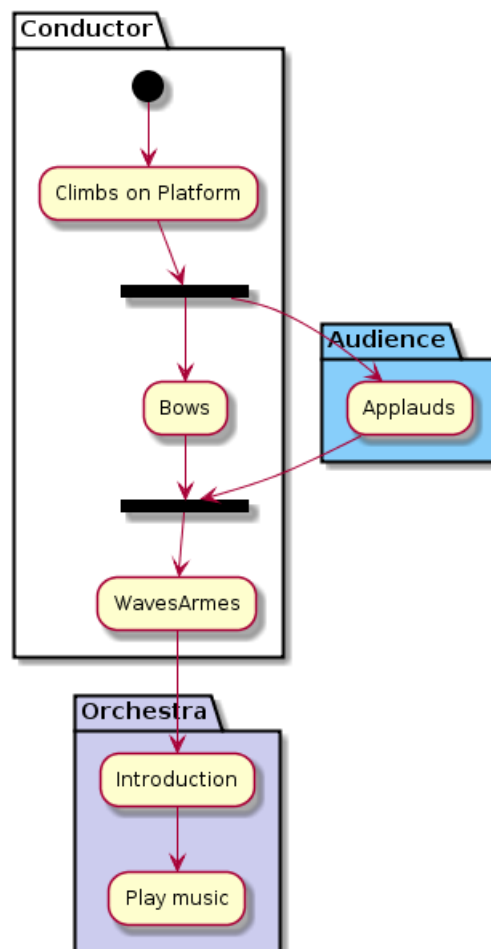
```
@startuml
partition Conductor {
(*) --> "Climbs on Platform"
--> == S1 ==
--> Bows
}

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

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

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

@enduml
```



4.10 スキンパラメータ

コマンド `skinparam` を使用して、図面の色とフォントを変更することができます。

このコマンドを使用することができます：

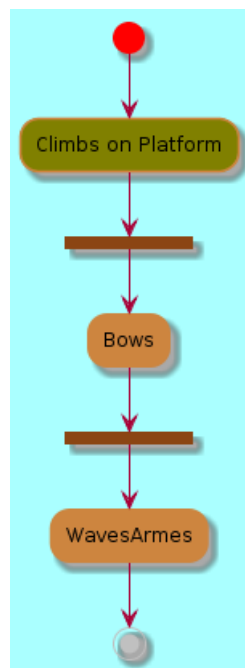
- 図の定義中では、他のコマンドと同様に、
- インクルードされたファイルの中で、
- コマンドラインまたは ANT タスクで提供される構成ファイルの中で。

定型アクティビティには、特定の色とフォントを定義できます。

```
@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
```



4.11 八角形

コマンド `skinparam activityShape octagon` を使用して、アクティビティの形状を八角形に変更できます。

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

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

@enduml
```



4.12 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

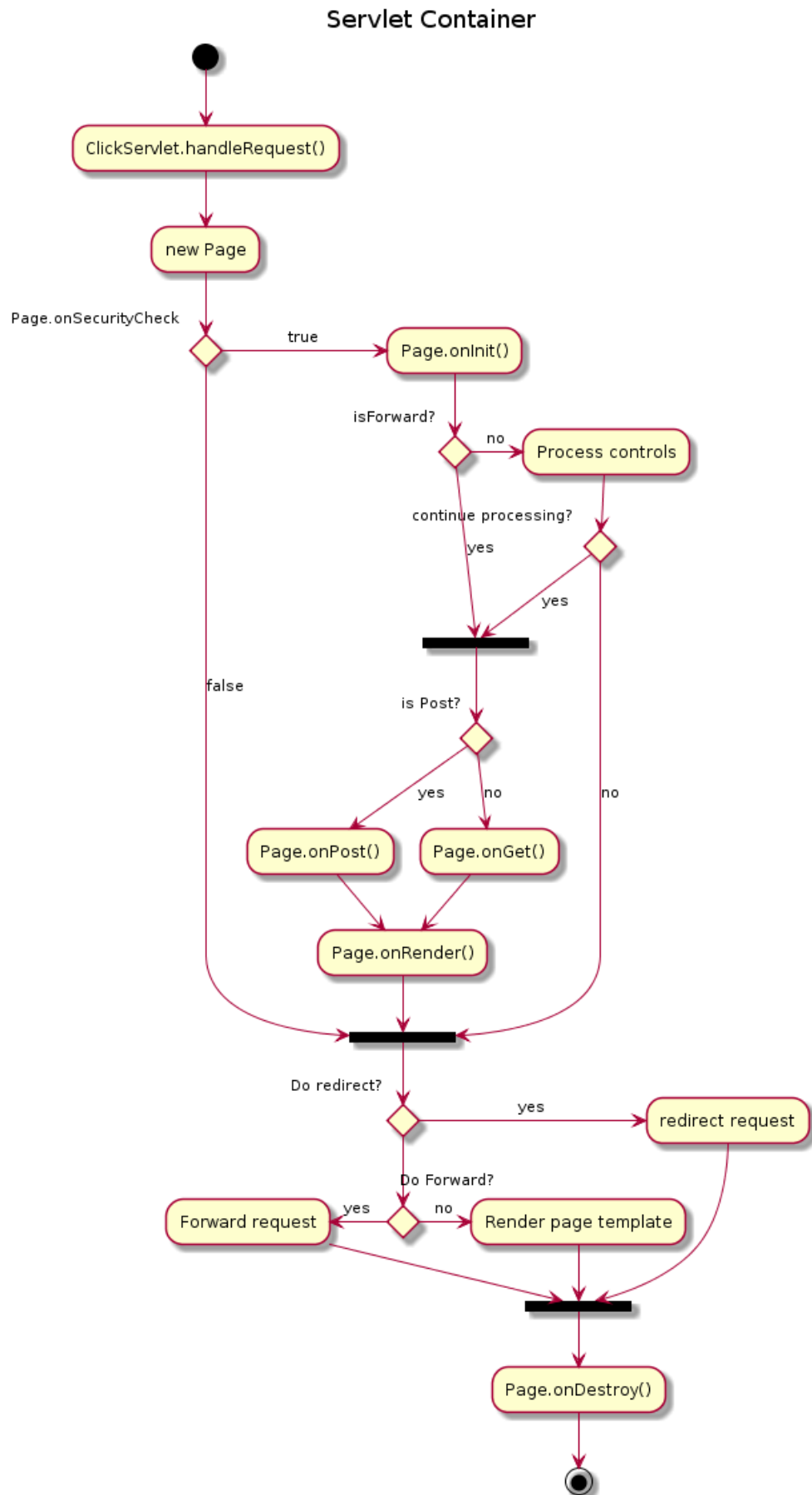
else
-->[false] ===REDIRECT_CHECK===
endif

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

```



5 アクティビティ図 (ベータ版)

アクティビティ図の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と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。

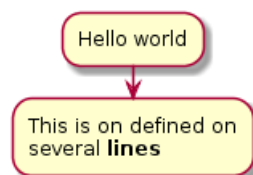
5.1 単純なアクティビティ

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。

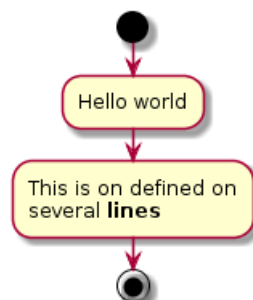
```
@startuml
:Hello world;
:This is on defined on
several **lines**;
@enduml
```



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
```

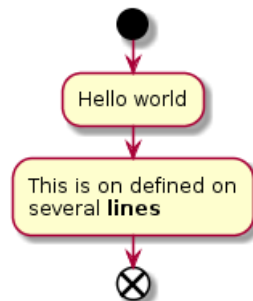


キーワード end もまた使用できます。

```

@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。

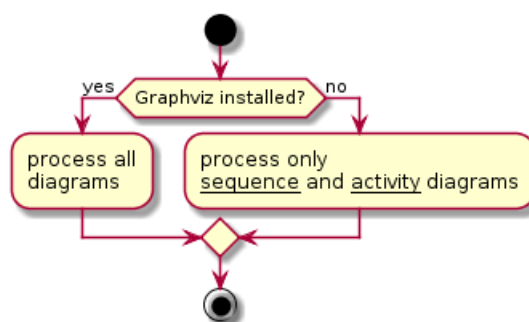
```

@startuml
start

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

stop
@enduml

```



いくつか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;
endif

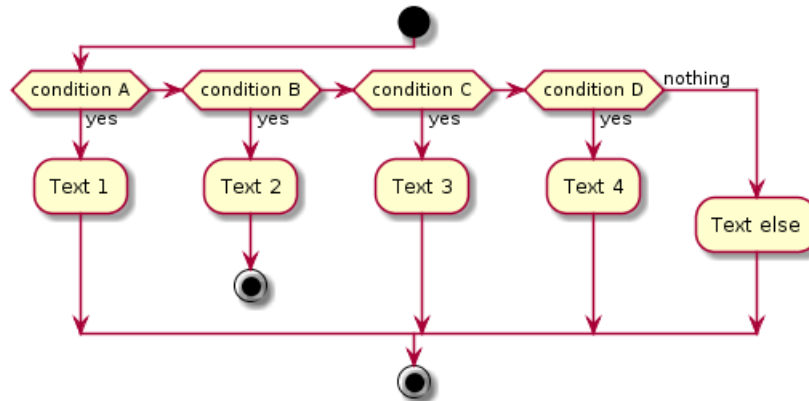
```



```

else (nothing)
:Text else;
endif
stop
@enduml

```



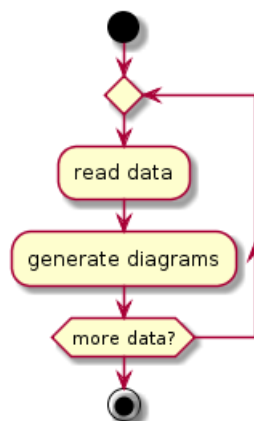
5.4 繰り返し（後判定）

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?)

```

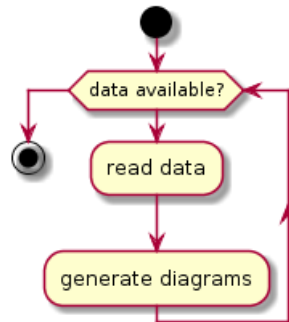
```

:read data;
:generate diagrams;
endwhile

stop

@enduml

```

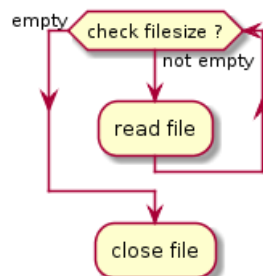


キーワード `endwhile` の後ろ、または、キーワード `is` を使用することで、ラベルを与えることができます。

```

@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。

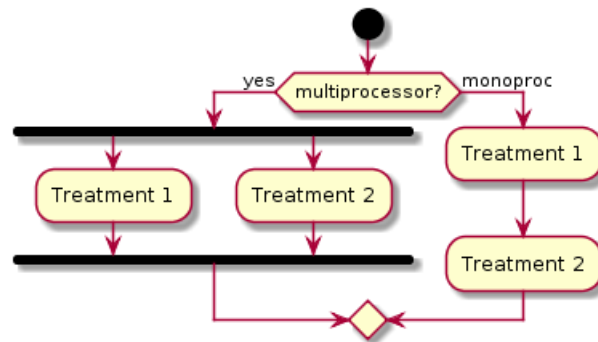
```

@startuml
start

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

@enduml

```



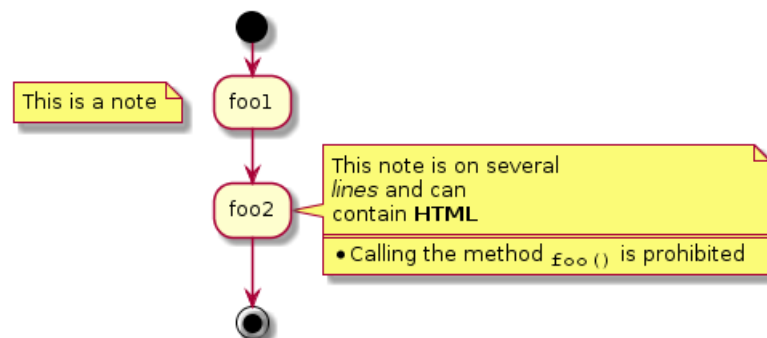
5.7 注釈

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

```



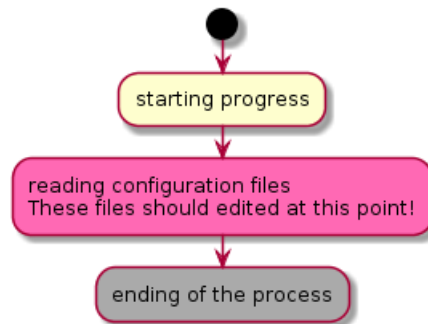
5.8 色指定

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

```

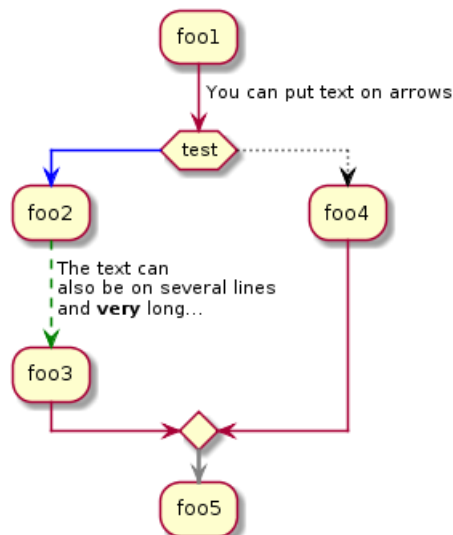


5.9 矢印

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
  
```



5.10 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;
}
  
```

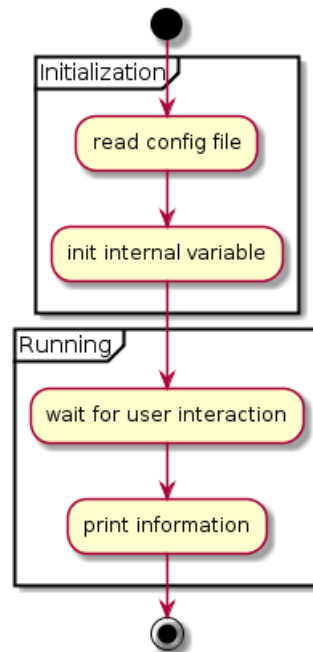


```

: init internal variable;
}
partition Running {
: wait for user interaction;
: print information;
}

stop
@enduml

```



5.11 動線

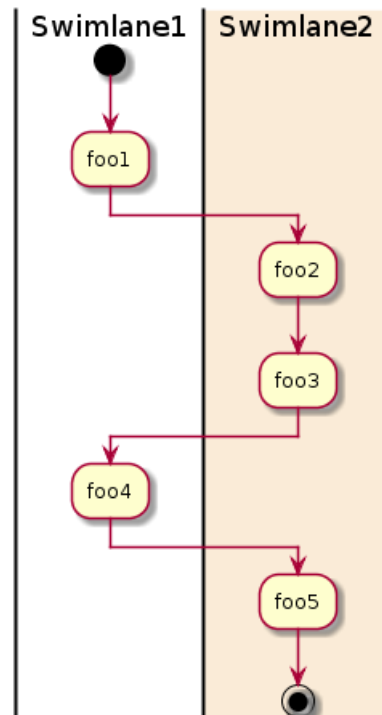
パイプ記号 | を用いて、複数の動線を定義することができます。

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

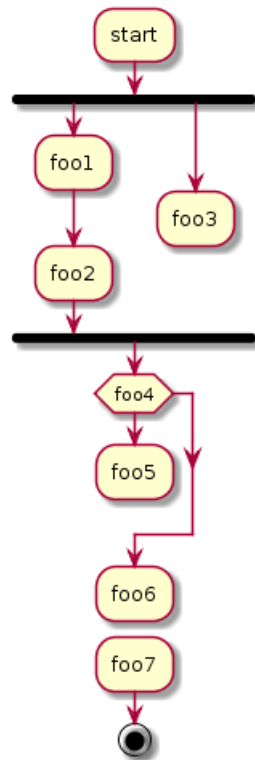
```



5.12 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
```



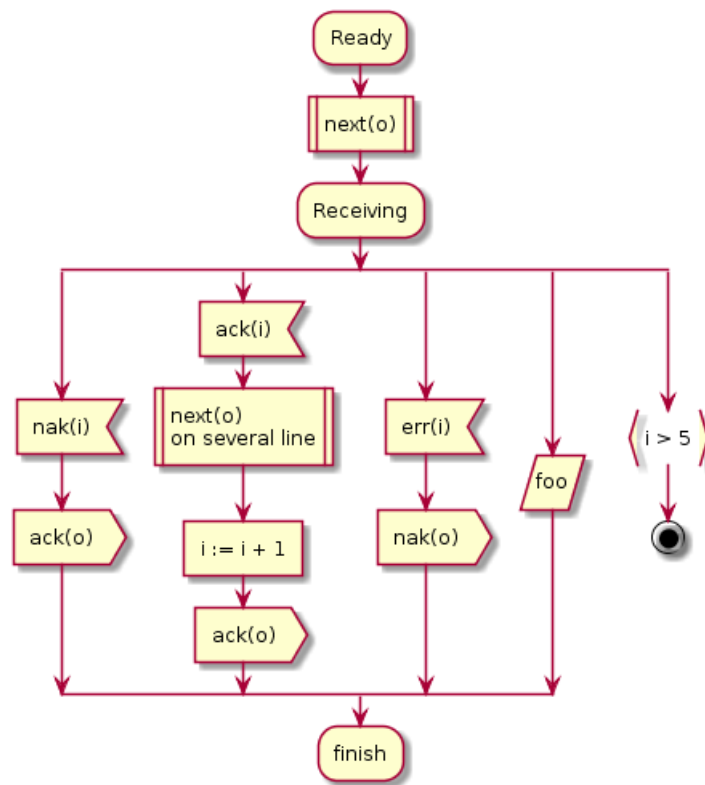
5.13 SDL 図

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
  
```



5.14 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
endif

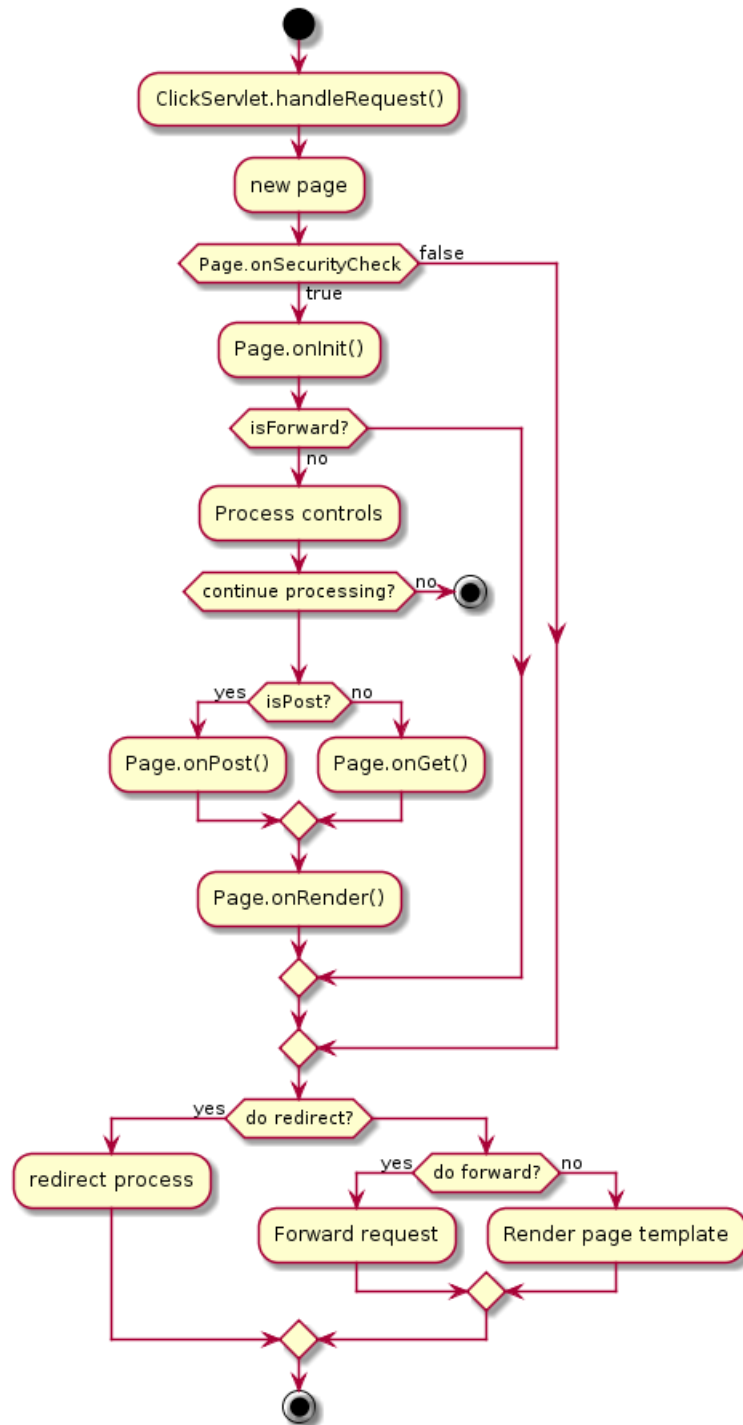
if (isPost?) then (yes)
:Page.onPost();
else (no)
:Page.onGet();
endif
:Page.onRender();
endif
else (false)
endif

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

stop

@enduml

```

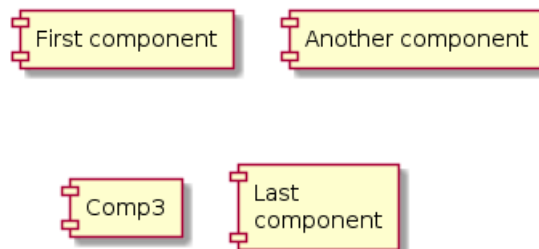
6 コンポーネント図

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
```



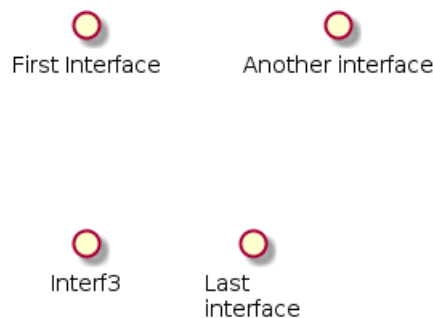
6.2 インタフェース

インタフェースは丸括弧 () でシンボルを囲うことで定義できます。(何故なら見た目が丸いからです。)

もちろん `interface` キーワードを使って定義することもできます。 `as` キーワードでエイリアスを定義できます。このエイリアスは後で、関係を定義する時に使えます。

後で説明されますが、インタフェースの定義は省略可能です。

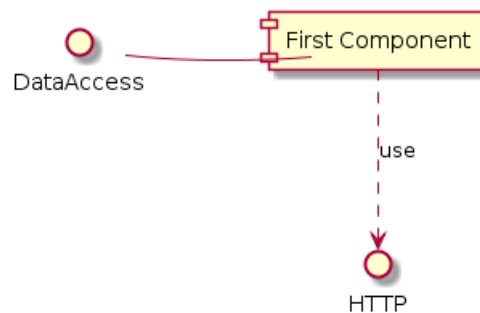
```
@startuml
() "First Interface"
() "Another interface" as Interf2
interface Interf3
interface "Last\ninterface" as Interf4
@enduml
```



6.3 基本的な例

要素間の関係は、破線 (..)、直線 (--), 矢印 (-->) の組合せで構成されます。

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



6.4 ノートの使用方法

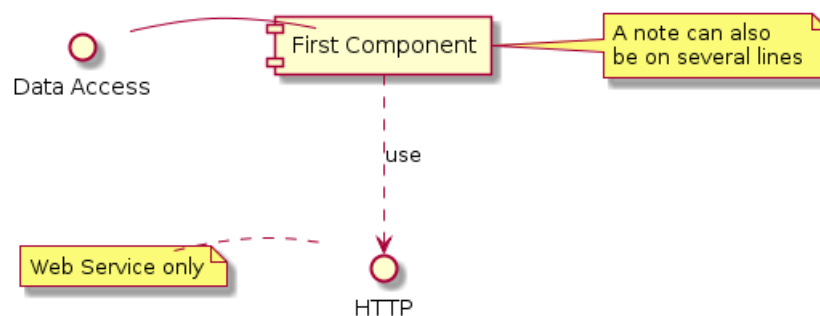
オブジェクトに関連のあるノートを作成するには `note left of`、`note right of`、`note top of`、`note bottom of` キーワードを使います。note left of、note right of、note top of、note bottom of

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
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 コンポーネントのグループ化

いくつかのキーワードをグループコンポーネントやインタフェースに使用することができます：

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

```
@startuml

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

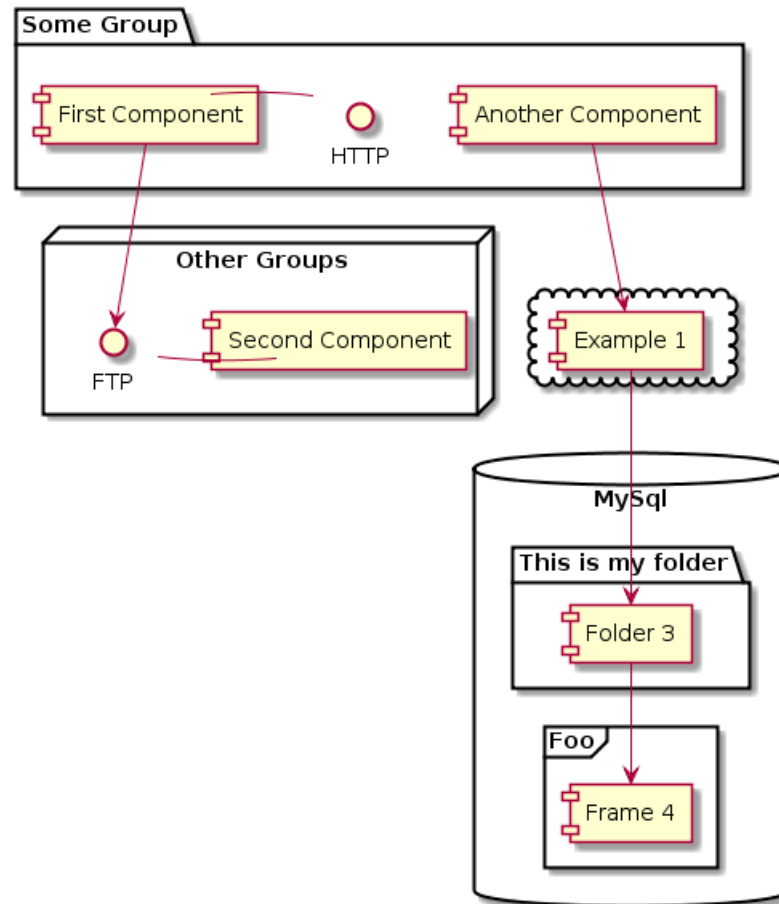
node "Other Groups" {
    FTP - [Second Component]
    [First Component] --> FTP
}

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]

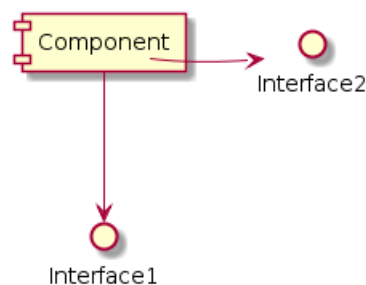
@enduml
```



6.6 矢印の方向を変える

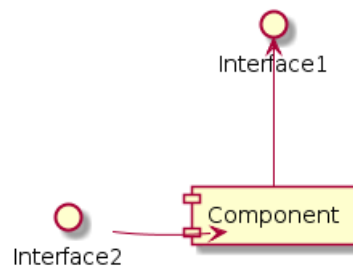
デフォルトではクラス間のリンクは2つのダッシュ--を持っており垂直方向に配向されています。次のように単一のダッシュ(またはドット)を置くことによって水平方向のリンクを使用することが可能です:

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



リンクを反対にすることで方向を変更することもできます。

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

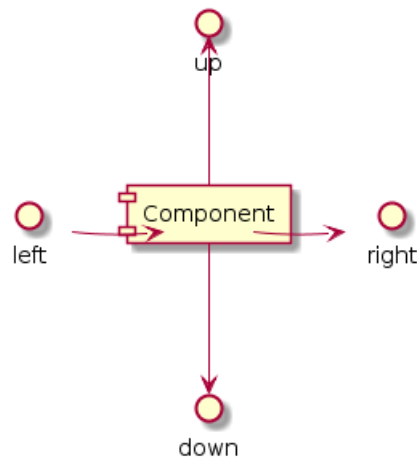


また、left を加えることで矢印の向きを変更することもできます。right, up, down などのキーワードを矢印の間に記述します。:

```

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

```



方向の最初の文字のみを使用して矢印を短くすることができます（例えば、-down- の代わりに -d-）、または最初の 2 文字（-do-）。

この機能を悪用してはならないことに注意してください: *Graphviz* は微調整の必要がない良い結果を通常は与えてくれます。

6.7 UML2 表記の使用

コマンド `skinparam componentStyle uml2` は、UML2 表記に切り替えるために使用されます。

```

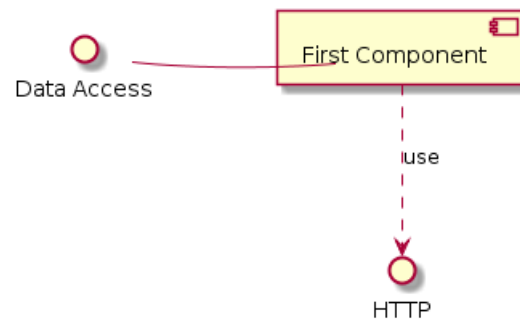
@startuml
skinparam componentStyle uml2

interface "Data Access" as DA

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

@enduml

```

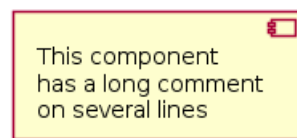


6.8 長い説明

角括弧を使用して説明を複数行で記述することができます。

```

@startuml
component comp1 [
This component
has a long comment
on several lines
]
@enduml
  
```



6.9 個々の色

コンポーネント定義のあとに色を指定することができます。

```

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



6.10 ステレオタイプでスプライトを使用

ステレオタイプのコンポーネント内にスプライトを使用することができます。

```

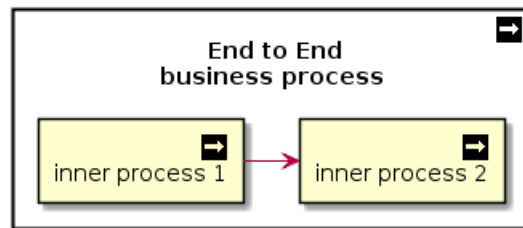
@startuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFF0FFFF
FFFFFFFFF0FFFF
FF0000000000FF
FF0000000000FF
FF0000000000FF
FFFFFFFFF0FFFF
FFFFFFFFF0FFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
}
  
```

```

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 見かけを変える

描画色とフォントを変えるためにコマンド `skinparam` を使用することができます。
このコマンドを次のように使用することができるのは：

- 図の定義では、他のコマンドのように、
- インクルードファイルで、
- 設定ファイルでは、コマンドラインまたは ANT タスクで提供。

ステレオタイプのコンポーネントおよびインタフェースのための特定の色とフォントを定義することができます。

```

@startuml

skinparam interface {
backgroundColor RosyBrown
borderColor orange
}

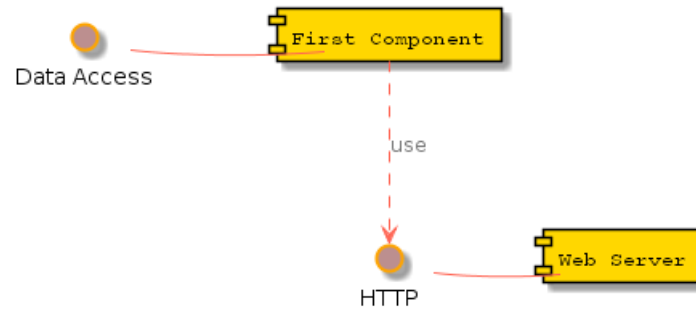
skinparam component {
FontSize 13
BackgroundColor<<Apache>> Red
BorderColor<<Apache>> #FF6655
FontName Courier
BorderColor black
BackgroundColor gold
ArrowFontName Impact
ArrowColor #FF6655
ArrowFontColor #777777
}

() "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>>

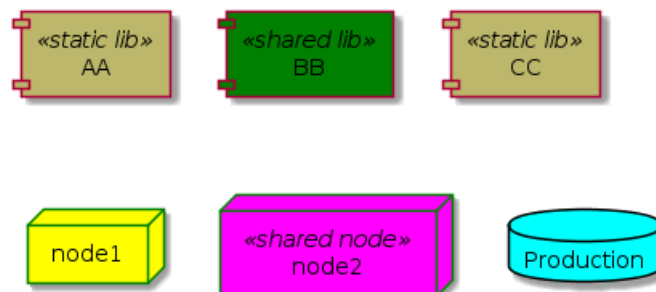
node node1
node node2 <<shared node>>
database Production

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

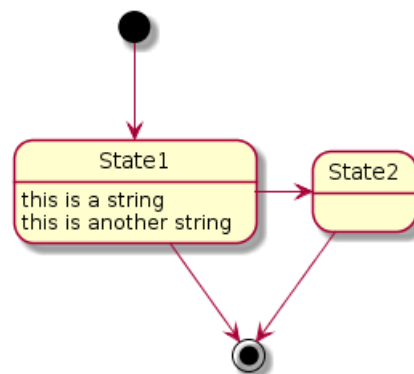
ステート図の始点と終点は、[*] で示します。

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

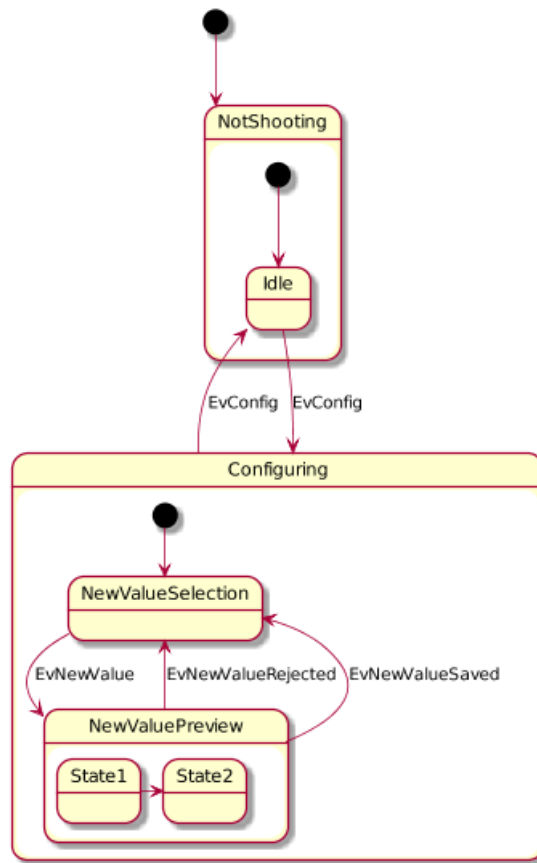
state NotShooting {
    [*] --> Idle
    Idle --> Configuring : EvConfig
    Configuring --> Idle : EvConfig
}

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。

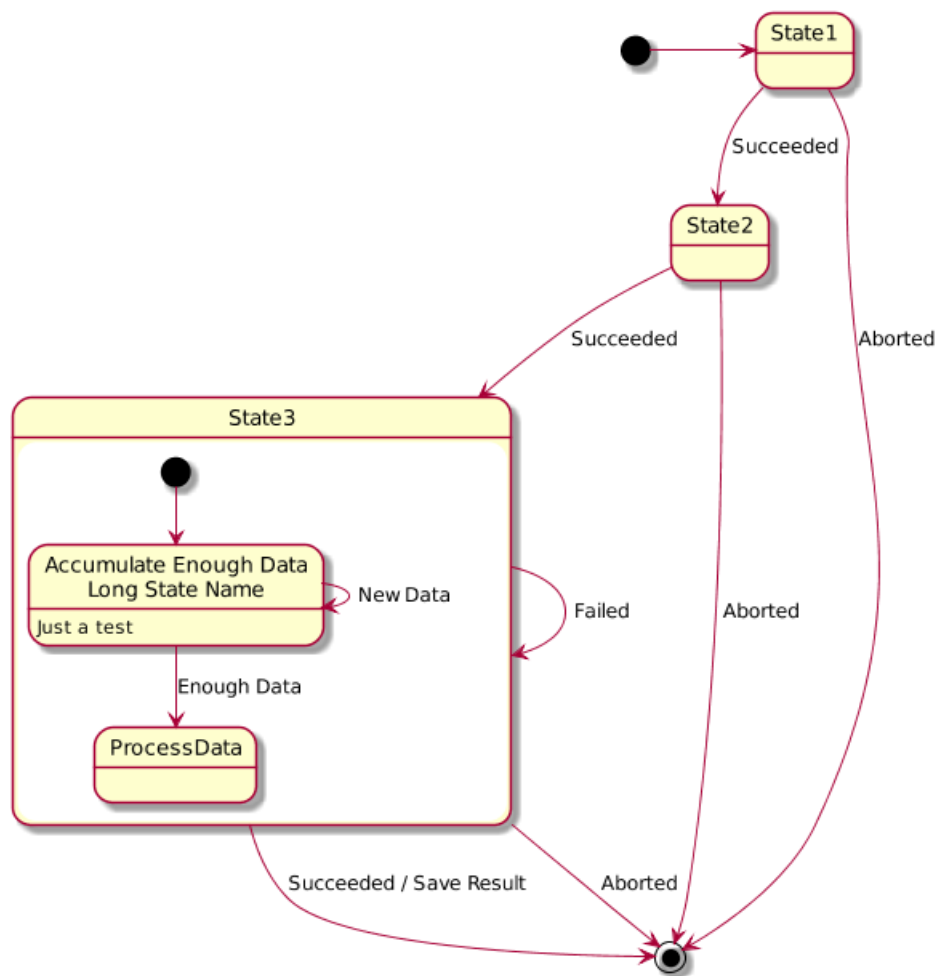
```

@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

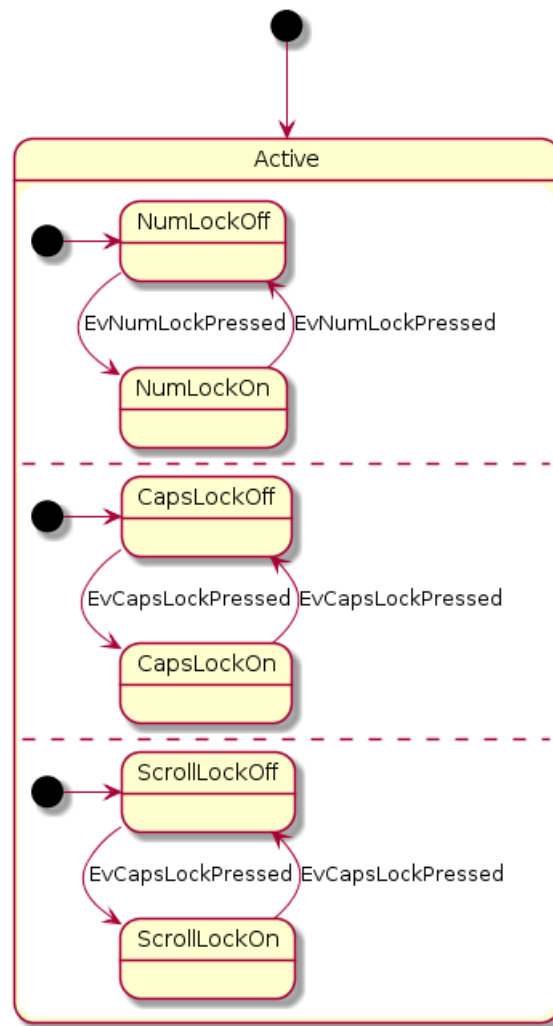
記号 `--` で分離することで、同時状態となる複合状態を定義することができます。

```

@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
  
```



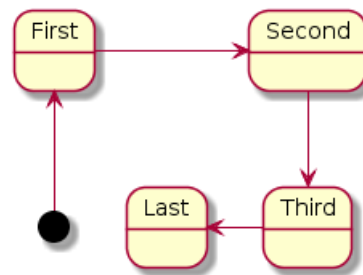
7.5 矢印の方向

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なりません：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。

```

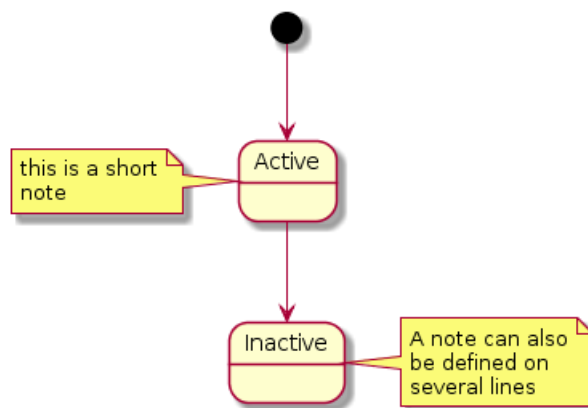
@startuml

[*] --> 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
  
```

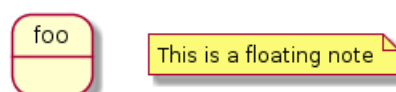


さらに、状態にひもづかない注釈を定義できます。

```

@startuml

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



7.7 もっと注釈

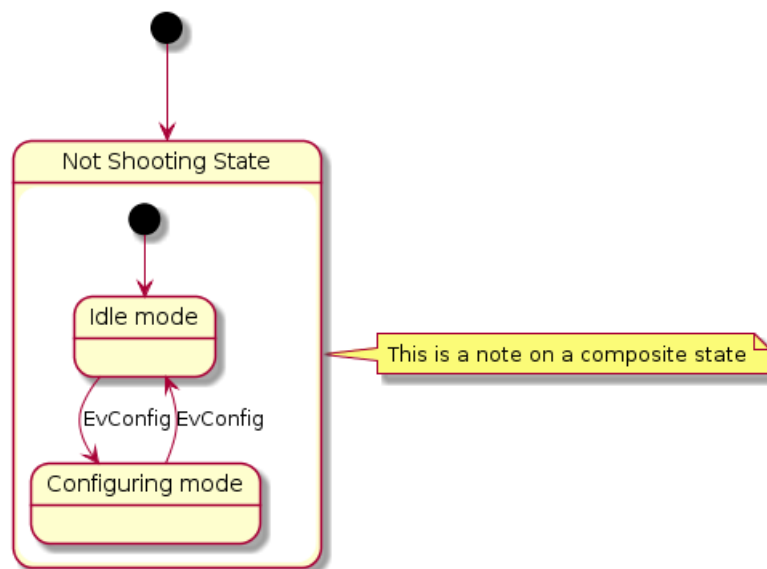
複合状態にも注釈をつけることができます。

```
@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。

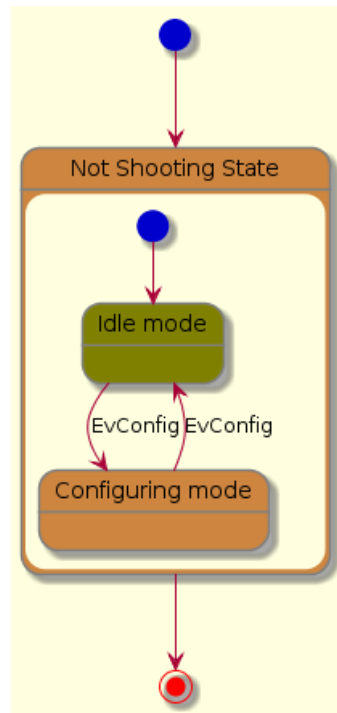
```
@startuml
skinparam backgroundColor LightYellow
skinparam state {
StartColor MediumBlue
EndColor Red
BackgroundColor Peru
BackgroundColor<<Warning>> Olive
BorderColor Gray
FontName Impact
}

[*] --> 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
```

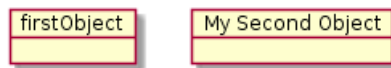


8 オブジェクト図

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。

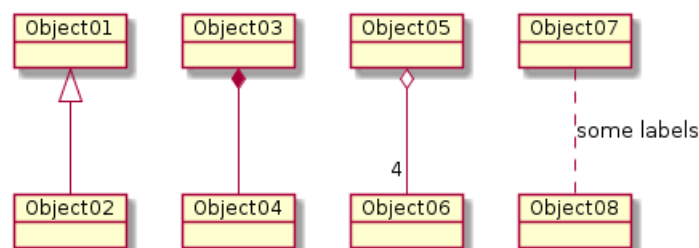
これらのルールを知ることで、以下の図を描くことができます。

関係にラベルをつけることができ、” : ” を使い、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
```



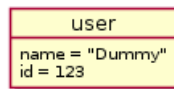
8.3 フィールドの追加

フィールドを宣言するには、シンボル ":" にフィールド名を続けます。

```
@startuml
object user

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

@enduml
```

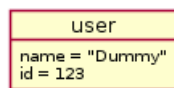


全てのフィールドを括弧 {} で括って範囲を示すことも可能です。

```
@startuml
```

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

```
@enduml
```



8.4 クラス図と共通の機能

- 可視性
- 注釈の定義
- パッケージの使用
- 出力スキム

9 共通コマンド

9.1 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、/ ' で始まり /' で終わる複数行にまたがるコメントも書けます。

9.2 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。

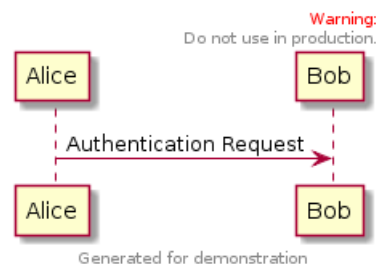
ヘッダー／フッターを HTML で記述することも可能です。

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

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

center footer Generated for demonstration

@enduml
```



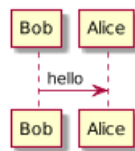
9.3 ズーム

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。

拡大率を定義するために、数値か分数のいずれかを使用することができます。幅または高さ（ピクセル数）のいずれかでも指定することができます。幅と高さの両方を与えることもできます：イメージは指定した範囲に収まるようにスケーリングされます。

- scale 1.5
- scale 2/3
- scale 200 width
- scale 200 height
- scale 200*100
- scale max 300*200
- scale max 1024 width
- scale max 800 height

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



9.4 タイトル

タイトルの指定にキーワード `title` を使います。タイトルの記述では `\n` を使用して新しい行を追加することができます。

いくつかの `skinparam` 設定が、タイトルに境界をつけるために利用可能です。

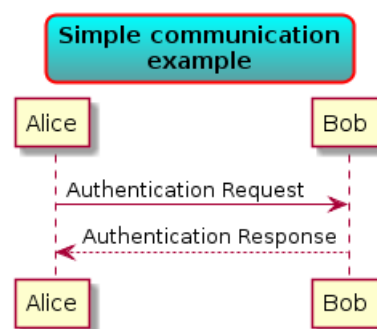
```

@startuml
skinparam titleBorderRoundCorner 15
skinparam titleBorderThickness 2
skinparam titleBorderColor red
skinparam titleBackgroundColor Aqua-CadetBlue

title Simple communication\nexample

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

@enduml
  
```



タイトルには creole フォーマットが使用できます。

また、キーワード `title` と `end title` 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。

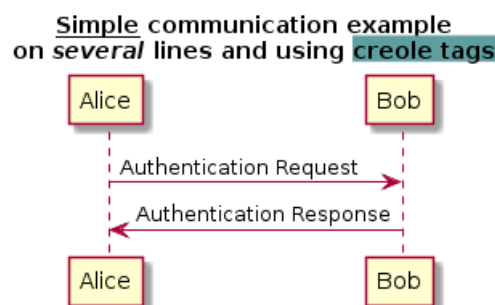
```

@startuml

title
<u>Simple</u> communication example
on <i>several</i> lines and using <back:cadetblue>creole tags</back>
end title

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

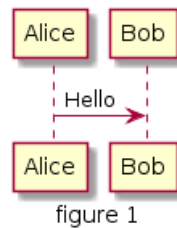
@enduml
  
```



9.5 見出し

図の下に見出しを置くキーワード `caption` もあります。

```
@startuml
caption figure 1
Alice -> Bob: Hello
@enduml
```

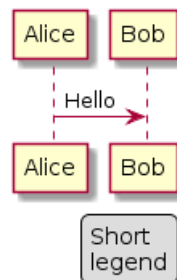


9.6 図の説明文

`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
```



10 Salt (GUI 設計ツール)

Salt はグラフィカルインタフェースの設計を助ける PlantUML のサブプロジェクトです。

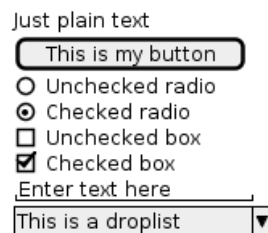
キーワード `@startsalt`、または、`@startuml` と次の行に続くキーワード `salt` の、いずれかを使用することができます。

10.1 基本のウィジェット

ウィンドウは中括弧で始めて中括弧で閉じなければなりません。次のように定義できます。

- ボタンは `[と]` で括ります。
- ラジオボタンは `(と)` で括ります。
- チェックボックスは `[と]` で括ります。
- テキスト領域は `"` で括ります。

```
@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
```



このツールの目標は簡単な見本でウィンドウについて議論することです。

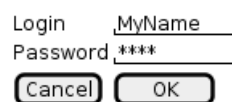
10.2 罫線の使用

表は括弧 `{` で開始すれば自動的に作成されます。

そして `|` で列を分割する必要があります。

例：

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



括弧で表を開始したら、行や列の罫線を表示したいがために定義された文字を使用することができます：

全ての縦横の罫線を表示する

! 全ての縦線を表示する

- 全ての横線を表示する

+ 枠線を表示する

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

Login	MyName
Password	****
Cancel	OK

10.3 セパレータの使用

いくつかの横線をセパレータとして使用することができます。

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

Text1
<u>Some field</u>
<u>Note on usage</u>
Another text
Ok

10.4 木構造ウィジェット

木構造があるなら、{T で開始して階層を示すために + を使用する必要があります。

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

```

++ Africa
}
}
@endsalt

```



10.5 括弧で括る

定義中に、新しい括弧で括ることによりサブ要素を定義することができます。

```

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

```

Name: _____
 Modifiers: ☒ public ☐ default ☐ private ☐ protected
 ☐ abstract ☐ final ☐ static
 Superclass: java.lang.Object Browse...

10.6 タブの追加

{/ 表記を使用してタブを追加することができます。太字のテキストを設けるように、HTML コードを使用できることに注意してください。

```

@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

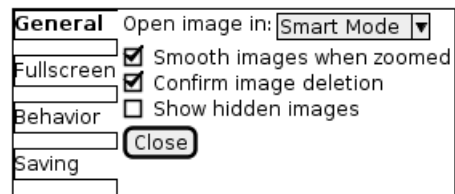
```

タブは垂直方向にも配向できます：


```

@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

```



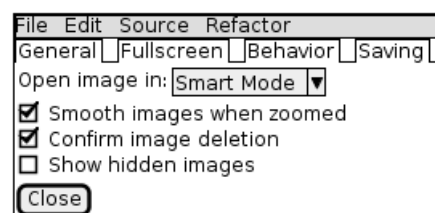
10.7 メニューの使用

{* 表記でメニューを追加することができます。

```

@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

```

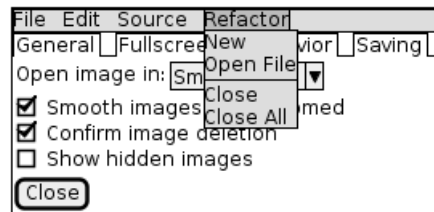


メニューを開くことも可能です：

```

@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 テーブル（上級）

テーブルのための2つの特別な表記を使用することができます。

- * は残りのセルとの範囲を示すために
- . は空のセルを示すために

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

	Column 2	Column 3
Row header 1	value 1	value 2
Row header 2	A long cell	

11 Creole (マークアップ言語)

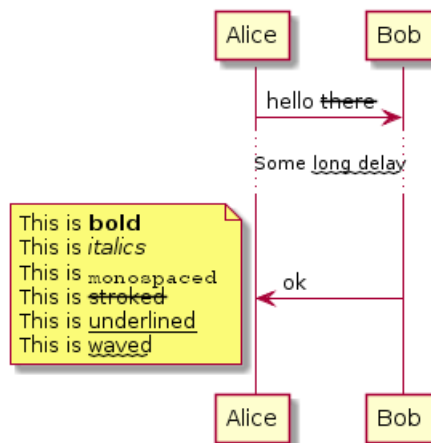
軽量版 Creole エンジン、文書の様式を定義する標準化された方法として PlantUML に統合されました。

現在、全ての図がこの構文をサポートしています。

HTML 構文との上位互換性が保たれていることに注意してください。

11.1 テキストの強調

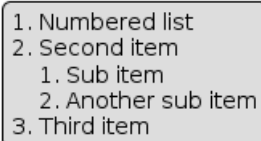
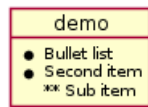
```
@startuml
Alice -> Bob : hello --there--
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
This is bold
This is italics
This is "monospaced"
This is --stroked--
This is __underlined__
This is ~~waved~~
end note
@enduml
```



11.2 リスト

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

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



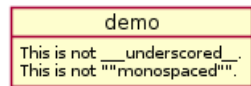
11.3 エスケープ文字

特別なクレオール文字をエスケープするためにチルダ ~ を使用することができます。

```

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

```

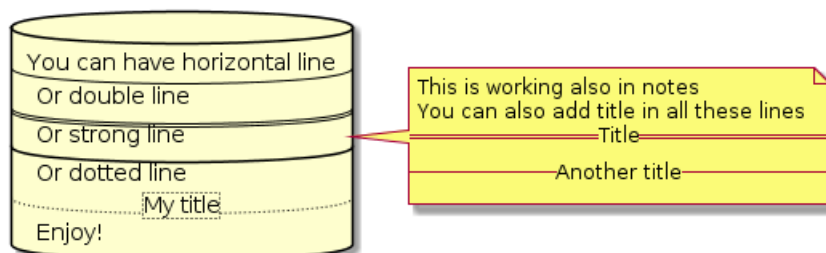


11.4 横線

```

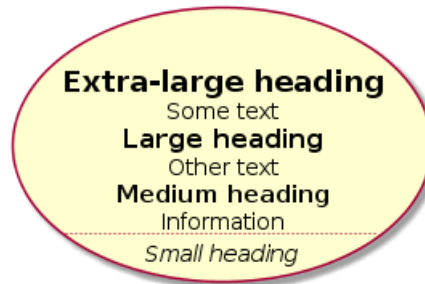
@startuml
database DB1 as "
You can have horizontal line
----
Or double line
====
Or strong line
----
Or dotted line
..My title..
Enjoy!
"
note right
This is working also in notes
You can also add title in all these lines
==Title==
--Another title--
end note
@enduml

```



11.5 見出し

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

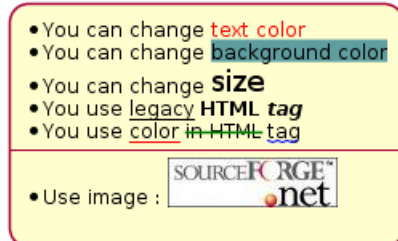


11.6 伝統的な HTML タグ

Some HTML tags are also working:

- `` は太字に
- `<u>` や `<u:colorName>` は下線に
- `<i>` は斜体 (イタリック) に
- `<s>` や `<s:colorName>` は打ち消し線に
- `<w>` や `<w:colorName>` は波線 (下線) に
- `<color:colorName>` や `<color:colorName>`
- `<back:colorName>` や `<back:colorName>` は背景色に
- `<size:nn>` はフォントサイズを変更するために
- `<img:file>` : そのファイルはファイルシステムによりアクセスできる必要があります
- `<img:http://url>` : その URL はインターネットによりアクセスできる必要があります

```
@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
```



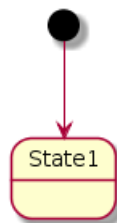
11.7 表

表を構築することが可能です。

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

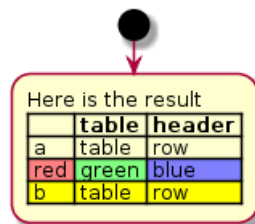
Example of simple table

	table	header
a	table	row
b	table	row



セルの背景色と線を指定することができます。

```
@startuml
start
:Here is the result
|= |= table |= header |
| a | table | row |
|<#FF8080> red |<#80FF80> green |<#8080FF> blue |
<#yellow>| b | table | row |;
@enduml
```

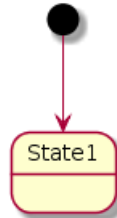
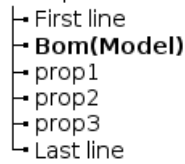


11.8 木構造

文字列 `|_` を木構造を構築するために使用できます。

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

Example of Tree



11.9 特殊文字

&# 構文や <U+XXXX> でユニコード文字を使用することが可能です。

```

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

```



11.10 Open Iconic

Open Iconic は非常に素晴らしいオープンソースのアイコンセットです。これらのアイコンはクレオールパーザに統合されているので、図の外部領域にそれらを使用することができます。

次の構文を使用できます：<&ICON_NAME>

```

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

```

♥Use of OpenIconic♥



完全なリストは Open Iconic ウェブサイト上で利用可能ですが、このような特殊な図形を使用することができます。

```

@startuml
listopeniconic
@enduml

```

List Open Iconic

Credit to

<https://useiconic.com/open>

account-login	bell	cloud	excerpt	justify-right	musical-note	star
account-logout	bluetooth	cloudy	expand-down	key	paperclip	sun
action-redo	bolt	code	expand-left	laptop	pencil	tablet
action-undo	book	cog	expand-right	layers	people	tag
align-center	bookmark	collapse-down	expand-up	lightbulb	person	tags
align-left	box	collapse-left	external-link	link-broken	phone	target
align-right	briefcase	collapse-right	eye	link-intact	pie-chart	task
aperture	british-pound	collapse-up	eyedropper	list	pin	terminal
arrow-bottom	browser	comment	file	list-rich	play-circle	text
arrow-circle-bottom	brush	comment-square	fire	location	plus	thumb-down
arrow-circle-left	bug	compass	flag	lock-locked	power-standby	thumb-up
arrow-circle-right	bullhorn	contrast	flash	lock-unlocked	print	timer
arrow-circle-top	calendar	copywriting	folder	loop-circular	project	transfer
arrow-left	camera-slr	credit-card	fork	loop-square	pulse	trash
arrow-right	caret-bottom	crop	fullscreen-enter	loop	puzzle-piece	underline
arrow-thick-bottom	caret-left	dashboard	fullscreen-exit	magnifying-glass	question-mark	vertical-align-bottom
arrow-thick-left	caret-right	data-transfer-download	globe	map-marker	rain	vertical-align-center
arrow-thick-right	caret-top	data-transfer-upload	graph	map	random	vertical-align-top
arrow-thick-top	chat	delete	grid-four-up	media-pause	reload	video
audio-spectrum	check	dial	grid-three-up	media-play	resize-both	volume-high
badge	chevron-bottom	document	grid-two-up	media-record	resize-height	volume-low
ban	chevron-left	dollar	grid-two-up	media-skip-backward	resize-width	volume-off
ban-chart	chevron-right	double-quote-sans-left	hard-drive	media-skip-forward	rss	warning
basket	chevron-top	double-quote-sans-right	headphones	media-step-backward	rss-alt	wifi
battery-empty	checkbox	double-quote-serif-left	heart	media-step-forward	script	wrench
battery-full	circle-check	double-quote-serif-right	home	media-stop	share-boxed	x
beaker	circle-x	droplet	image	medical-cross	share	yen
	clock	eject	inbox	menu	shield	zoom-in
	cloud-download	elevator	info	microphone	signal	zoom-out
	cloud-upload	envelope-closed	italic	minus	signpost	
		envelope-open	justify-center	monitor	sort-ascending	
		euro	justify-left	moon	sort-descending	
				move	spreadsheet	



11.11 スプライトの定義と使い方

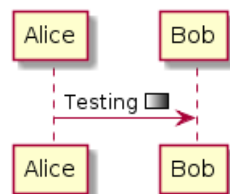
Sprite は、図で利用できる小さなグラフィック要素です。

PlantUML では、スプライトはモノクロで 4、8 または 16 のグレースケールのいずれかを持つことができます。

スプライトを定義するには、ピクセルごとに 0~F の間の 16 進数を使用する必要があります。

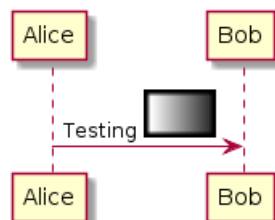
次に、<\$XXX> を使用してスプライトを使用できます。ここで XXX はスプライトの名前です。

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



You can scale the sprite.

```
@startuml
sprite $foo1 {
FFFFFFFFFFFFFFFF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
FFFFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1{scale=3}>
@enduml
```



11.12 スプライトのエンコード

スプライトをエンコードするには、コマンドラインを次のように使用します：

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

foo.png は使用したい画像ファイルです (自動的にグレイに変換されます)。

-encodesprite の後で、フォーマットを指定する必要があります: 4, 8, 16, 4z, 8z または 16z。

数値はグレイレベルを示し、オプションの z はスプライト定義で圧縮を有効にするために使用されます。

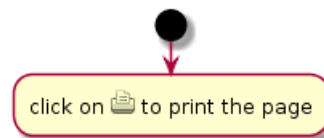
11.13 スプライトのインポート

GUI を起動して、既存のイメージからスプライトを生成することもできます。

メニューバーをクリックし File/Open Sprite Window をクリックします。

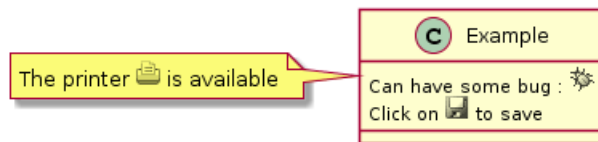
イメージをクリップボードにコピーした後、対応するスプライトの定義がいくつか表示されます: 必要なものをピックアップするだけです。

11.14 例



```
@startuml
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPE
start
:click on <$printer> to print the page;
@enduml
```

Use of sprites (🖨️, 🐛...)



```
@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S40T7C53cKpsHpaYPDJY_12MMH-BLRyywPhrrlw3qu
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPE
sprite $disk {
444445566677881
436000000009991
43600000000ACA1
537000000001A7A1
5370000000012B8A1
538000000123B8A1
638000001233C9A1
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

コマンド `skinparam` を使用して、製図の色やフォントを変えることができます。例：

```
skinparam backgroundColor yellow
```

このコマンドを使用することができます：

- 図の定義には、他のコマンドのように、
- インクルードファイルには（*Preprocessing* 参照）、
- 設定ファイルには、コマンドラインまたは ANT タスクで提供します。

12.2 入れ子

繰返しを避けるために、入れ子に定義することが可能です。次のような定義は、

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

次のとまったく等価です。

```
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	#FEFECF		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	#FEFECF		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	#FEFECF		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	#FEFECF		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	#FEFECF		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	#FEFECF		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	#FEFECF		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	#FEFECF		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	#FEFECF		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	#FEFECF		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 Apex
```

`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 Apex
```

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

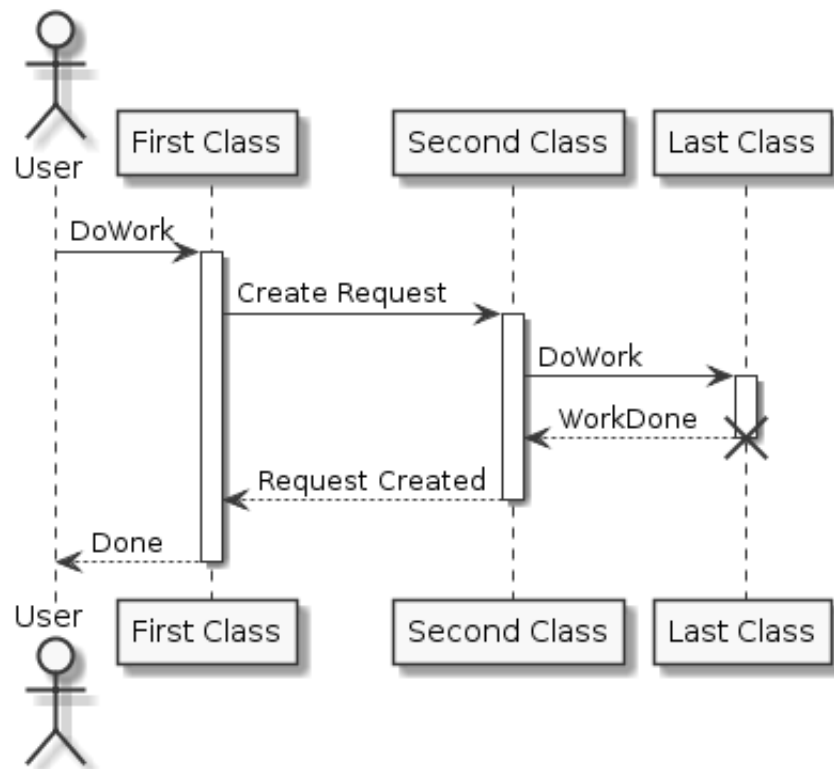
A -> B: Create Request
activate B

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。

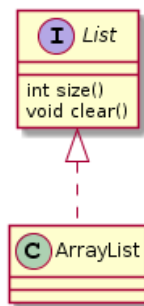
それらの機能は、“#” が、“感嘆符”!“” であることを除けば、C 言語のプリプロセッサととても良く似ています。

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。

ファイルは 1 度だけインクルードできます。全く同じファイルを何度もインクルードしたい場合は、`!include` の代わりに `!include_many` ディレクティブを使用する必要があります。

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 URL でインクルード

図にインターネット/イントラネットからファイルをインクルードする `!includeurl` コマンドを使用します。

`http://someurl.com/mypath` からインクルードしたい `@startuml/@enduml` ブロックを指定するために `!includeurl http://someurl.com/mypath!0` を使用することもできます。`!0` 表記は最初の図を示します。

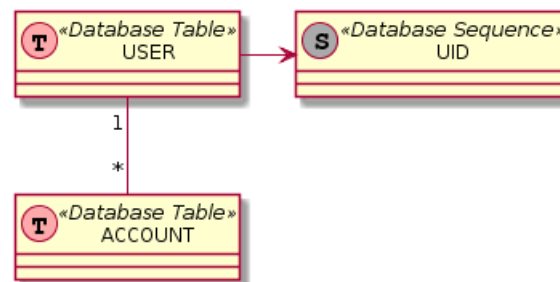


13.3 定数を定義する

定数を定義するには擬似命令 `!define` 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
```



もちろん、擬似命令 `!include` を使用して、図に含めるすべての定数を 1 つのファイルに定義することができます。

定数は擬似命令 `!undef XXX` で未定義にすることができます。

-D フラグを使用して、コマンドライン内で定数を指定することもできます。

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

-D フラグは“-jar plantuml.jar” セクションの後に置かなければならないことに注意してください。

13.4 マクロ定義

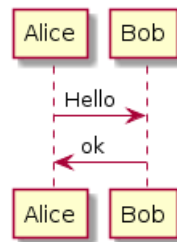
引数を持つマクロも定義できます。

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



マクロは複数の引数を持つことができます。

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



13.5 日付と時刻の追加

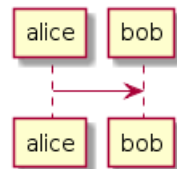
特殊な変数 `%date%` を使用して現在の日付や時刻を拡張することもできます。

日付は SimpleDataFormat 文書で規定された形式を使用して指定することができます。

```

@startuml
!define ANOTHER_DATE %date[yyyy.MM.dd 'at' HH:mm]%
Title Generated %date% or ANOTHER_DATE
alice -> bob
@enduml
  
```

Generated Mon Apr 17 19:35:48 UTC 2017 or 2017.04.17 at 19:35



13.6 その他の特殊な変数

次のような特殊な変数が利用可能です。

`%dirpath%` 現在のファイルのパス

`%filename%` 現在のファイルの名称

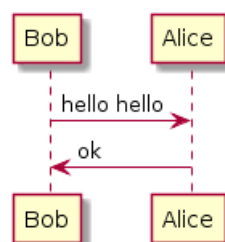
13.7 複数行のマクロ

`!definelong` と `!enddefinelong` を使用して複数行でマクロを定義することもできます。

```

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

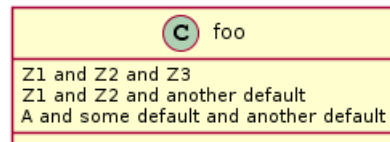
AUTHEN(Bob,Alice)
@enduml
  
```



13.8 マクロパラメータのデフォルト値

マクロパラメータにデフォルト値を割り当てることができます。

```
@startuml
!define some_macro(x, y = "some default" , z = 'another default' ) x and y and z
class foo {
some_macro(Z1, Z2, Z3)
some_macro(Z1, Z2)
some_macro(A)
}
@enduml
```



13.9 条件つき

擬似命令 `!ifdef XXX` と `!endif` を使用して、条件付き描画を行うことができます。

これら 2 つの擬似命令の間の行は、`!ifdef` の後の定数が前に定義されている場合にのみ含まれます。

定数に `not` が定義されている場合は、`!else` 部分をインクルードすることもできます。

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

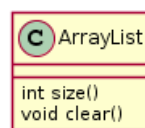


File `ArrayList.iuml`:

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

次に、`!define` 指示文を使用して図の条件部分を有効にすることができます。

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



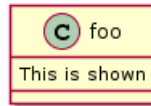
指定された定数が定義されていない場合は、行を含む擬似命令 `!ifndef` を使用することもできます。

評価式では、括弧と演算子 `||` とでブール式を使用することができます。

```

@startuml
!define SHOW_FIELDS
!undef SHOW_METHODS
class foo {
!ifdef SHOW_FIELDS || SHOW_METHODS
This is shown
!endif
!ifdef SHOW_FIELDS && SHOW_METHODS
This is NOT shown
!endif
}
@enduml

```



13.10 検索パス

コマンドラインで java プロパティ "plantuml.include.path" を指定することができます。

例えば：

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

この -D オプションは、-jar オプションの前に置く必要があることに注意してください。-r オプションの後の -D オプションは、plantuml プリプロセッサ内の定数を定義するために使用されます。

13.11 高度な機能

構文 ## を使用してマクロの引数にテキストを結びつけることが可能です。

```

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

```

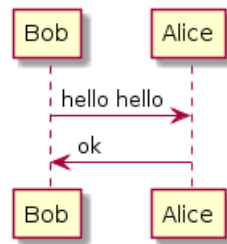


マクロは別のマクロで定義することができます。

```

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

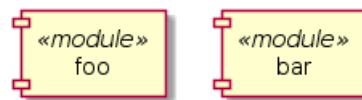
```



マクロは引数の個数で多態になることができます。

```

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



インクルードファイルを使用するときにはシステムの環境変数や定数の定義を使用することができます：

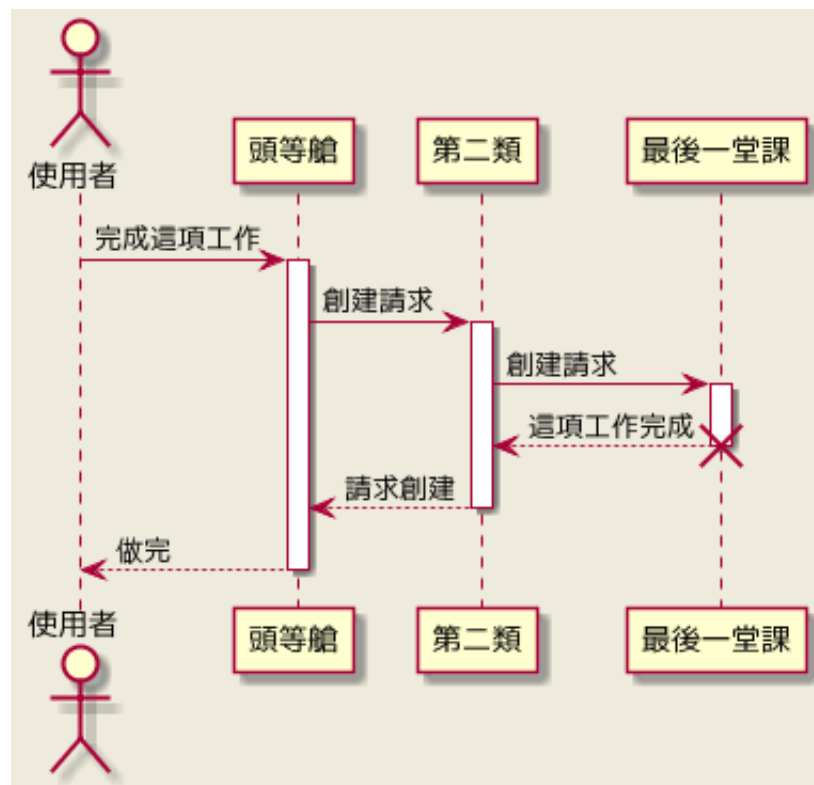
```

!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 #EEEEBD
actor 使用者
participant "頭等艙" as A
participant "第二類" as B
participant "最後一堂課" as 别的東西
使用者 -> A: 完成這項工作
activate A
A -> B: 創建請求
activate B
B -> 别的東西: 創建請求
activate 别的東西
别的東西 --> B: 這項工作完成
destroy 别的東西
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、必要になれ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

1	シーケンス図	1
1.1	基本的な例	1
1.2	分類子の宣言	1
1.3	分類子名にアルファベット以外を使う	2
1.4	自分自身へのメッセージ	3
1.5	矢印の見た目を変える	3
1.6	矢印の色を替える	4
1.7	メッセージシーケンスの番号付け	4
1.8	図の分割	6
1.9	メッセージのグループ化	7
1.10	メッセージの注釈	8
1.11	その他の注釈	8
1.12	ノートの形を変える。	9
1.13	Creole と HTML	10
1.14	境界線	11
1.15	リファレンス	11
1.16	遅延	12
1.17	間隔	12
1.18	ライフラインの活性化と破壊	13
1.19	分類子の生成	14
1.20	インとアウトのメッセージ	15
1.21	ステレオタイプとスポット	16
1.22	タイトルについての詳細	17
1.23	分類子の囲み	18
1.24	フッターの除去	19
1.25	スキンパラメータ	19
1.26	パディングの変更	21
2	ユースケース図	22
2.1	ユースケース	22
2.2	アクター	22
2.3	ユースケースの説明	22
2.4	簡単な例	23
2.5	継承	24
2.6	ノートの使用方法	24
2.7	ステレオタイプ	25
2.8	矢印の方向を変えるには	25
2.9	図を分割する	26
2.10	左から右に描画する	27
2.11	スキン設定 (Skinparam)	28
2.12	完全な例	29

3 クラス図	30
3.1 クラス間の関係	30
3.2 関係のラベル	31
3.3 メソッドの追加	32
3.4 可視性の定義	33
3.5 Abstract と Static	34
3.6 高等なクラス本体	35
3.7 注釈とステレオタイプ	36
3.8 注釈の詳細	37
3.9 リンクへの注釈	38
3.10 抽象クラスとインタフェース	39
3.11 非文字の使用	40
3.12 属性、メソッド等の非表示	41
3.13 非表示クラス	42
3.14 ジェネリクスの使用	42
3.15 特殊な目印	42
3.16 パッケージ	43
3.17 パッケージスタイル	43
3.18 名前空間	44
3.19 自動的に名前空間を作成する	45
3.20 ロリポップ（棒付きキャンディー）インタフェース	46
3.21 矢印の向きを変える	46
3.22 関連クラス	47
3.23 化粧をする	48
3.24 ステレオタイプの化粧	49
3.25 色のグラデーション	49
3.26 レイアウトの手助け	50
3.27 大きなファイルの分割	51
4 アクティビティ図	53
4.1 単純なアクティビティ	53
4.2 矢印のラベル	53
4.3 矢印の方向を変える	53
4.4 分岐	54
4.5 もっと分岐	55
4.6 同期	56
4.7 長いアクティビティの記述	56
4.8 注釈	57
4.9 パーティション	57
4.10 スキンパラメータ	58
4.11 八角形	59
4.12 完全な例	60

5	アクティビティ図（ベータ版）	62
5.1	単純なアクティビティ	62
5.2	開始／終了	62
5.3	条件文	63
5.4	繰り返し（後判定）	64
5.5	繰り返し（前判定）	64
5.6	並列処理	65
5.7	注釈	66
5.8	色指定	66
5.9	矢印	67
5.10	グループ化	67
5.11	動線	68
5.12	分離	69
5.13	SDL 図	70
5.14	完全な例	71
6	コンポーネント図	73
6.1	コンポーネント	73
6.2	インタフェース	73
6.3	基本的な例	74
6.4	ノートの使用法	74
6.5	コンポーネントのグループ化	75
6.6	矢印の方向を変える	76
6.7	UML2 表記の使用	77
6.8	長い説明	78
6.9	個々の色	78
6.10	ステレオタイプでスプライトを使用	78
6.11	見かけを変える	79
7	状態図	81
7.1	簡単な状態	81
7.2	複合状態	81
7.3	長い言葉	82
7.4	同時状態	83
7.5	矢印の方向	84
7.6	注釈	85
7.7	もっと注釈	86
7.8	見栄え	86
8	オブジェクト図	88
8.1	オブジェクトの定義	88
8.2	オブジェクト間の関係	88
8.3	フィールドの追加	88
8.4	クラス図と共通の機能	89

9 共通コマンド	90
9.1 コメント	90
9.2 フッターとヘッダー	90
9.3 ズーム	90
9.4 タイトル	91
9.5 見出し	92
9.6 図の説明文	92
10 Salt (GUI 設計ツール)	93
10.1 基本のウィジェット	93
10.2 罫線の使用	93
10.3 セパレータの使用	94
10.4 木構造ウィジェット	94
10.5 括弧で括る	95
10.6 タブの追加	95
10.7 メニューの使用	96
10.8 テーブル (上級)	97
11 Creole (マークアップ言語)	98
11.1 テキストの強調	98
11.2 リスト	98
11.3 エスケープ文字	99
11.4 横線	99
11.5 見出し	100
11.6 伝統的な HTML タグ	100
11.7 表	101
11.8 木構造	101
11.9 特殊文字	102
11.10 Open Iconic	102
11.11 スプライトの定義と使い方	104
11.12 スプライトのエンコード	104
11.13 スプライトのインポート	105
11.14 例	105
12 フォントや色の変更	107
12.1 使い方	107
12.2 入れ子	107
12.3 色	108
12.4 フォントの色、名前、サイズ	109
12.5 白黒	112

13 プリプロセッシング	113
13.1 ファイルのインクルード	113
13.2 URL でインクルード	113
13.3 定数を定義する	114
13.4 マクロ定義	114
13.5 日付と時刻の追加	115
13.6 Other special variables	115
13.7 複数行のマクロ	115
13.8 マクロパラメータのデフォルト値	116
13.9 条件つき	116
13.10 検索パス	117
13.11 高度な機能	117
14 国際化	119
14.1 キャラクターセット	119
15 色の名前	121