# Drawing UML with PlantUML

## Language Reference Guide

**PlantUML** is an Open Source project that allows to quickly write:

- sequence diagram,
- usecase diagram,
- class diagram,
- activity diagram,
- component diagram,
- state diagram.

Diagrams are defined using a simple and intuitive language.

# 1- Sequence Diagram

## Basic examples

Every UML description must start by `@startuml` and must finish by `@enduml`.

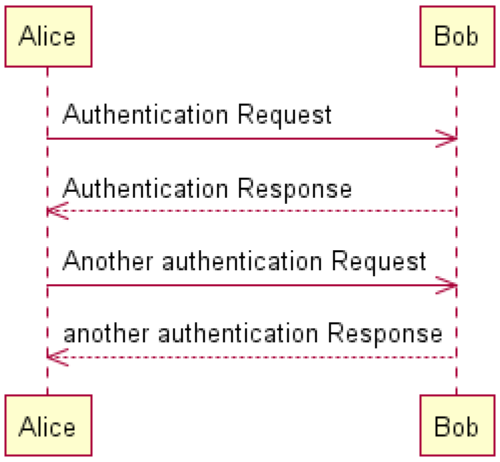
The sequence `"->"` is used to draw a message between two participants. Participants do not have to be explicitly declared.

To have a dotted arrow, you use `-->`

It is also possible to use `<-` et `<--`. That does not change the drawing, but may improve lisibility.

Example:

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

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


Alice
Bob
Authentication Request
Authentication Response
Another authentication Request
another authentication Response
Alice
Bob

## Declaring participant

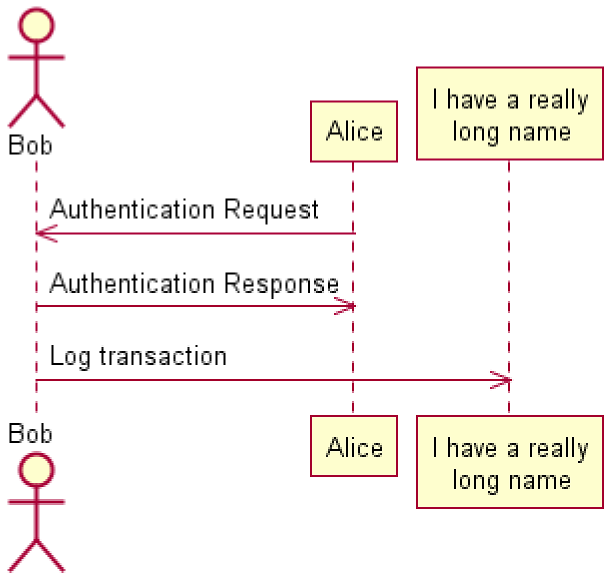
It is possible to change participant order using the `participant` keyword. .

It is also possible to use the `actor` keyword to use a stickman instead of a box for the participant.

You can rename a participant using the `as` keyword.

Everything that starts with simple quote '  is a comment.

```
@startuml
actor Bob
' The only difference between actor and participant is the drawing
participant Alice
participant "I have a really\nlong name" as L

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

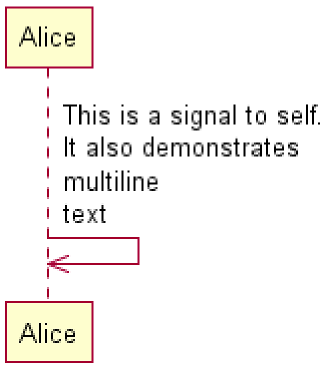


## Message to Self

A participant can send a message to itself.

It is also possible to have multilines using `\n`.

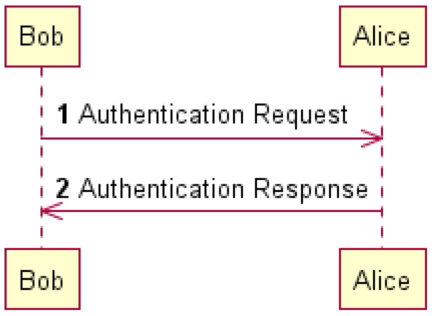
```
@startuml

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

@enduml
```



## Message sequence numbering

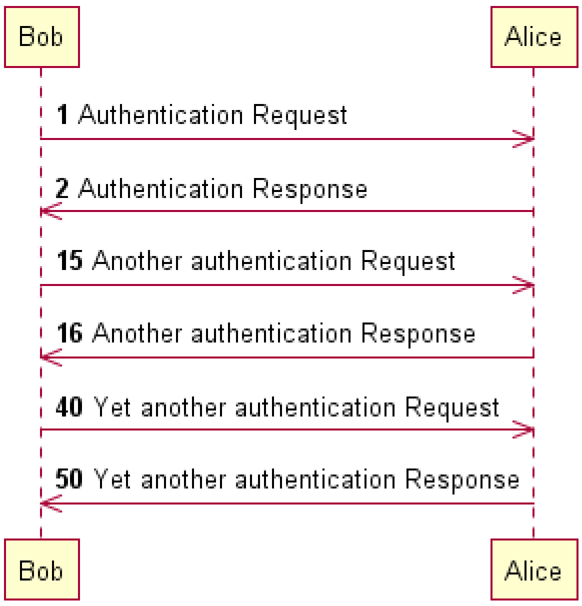
The keyword `autonumber` is used to automatically add number to messages.

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



You can specify a startnumber with `autonumber 'start'` , and also an increment with `autonumber 'start' 'increment'`

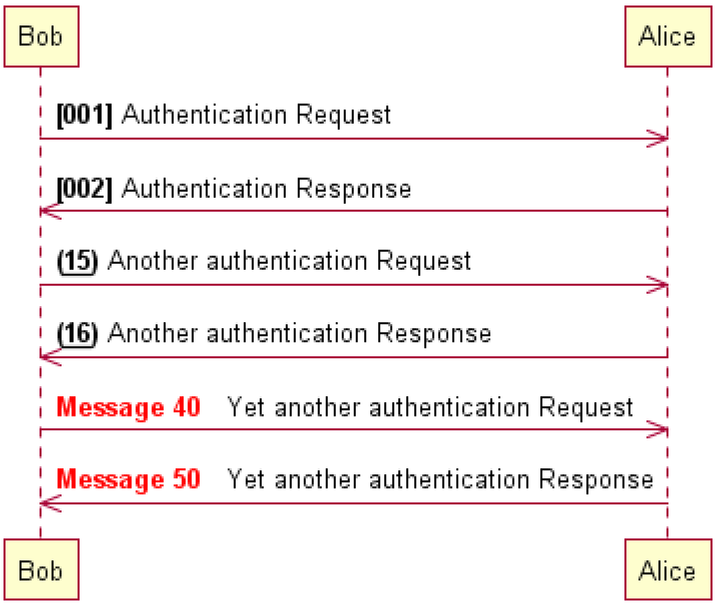
```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

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

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

@enduml
```


Bob
Alice
1 Authentication Request
2 Authentication Response
15 Another authentication Request
16 Another authentication Response
40 Yet another authentication Request
50 Yet another authentication Response
Bob
Alice

You can specify a format for your number by using between double-quote.
The formatting is done with the Java class `DecimalFormat` ('0' means digit, '#' means digit and zero if absent).

You can also use some html tags in the format.

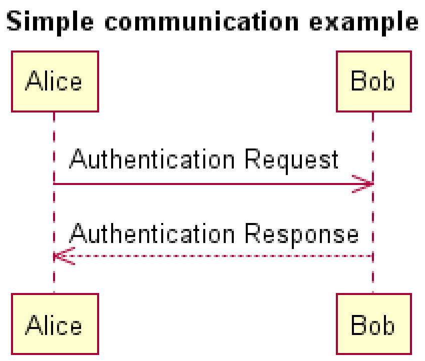
```
@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

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

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

@enduml
```


Bob
Alice
[001] Authentication Request
[002] Authentication Response
(15) Another authentication Request
(16) Another authentication Response
Message 40 Yet another authentication Request
Message 50 Yet another authentication Response
Bob
Alice

## Title

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

```
@startuml

title Simple communication example

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

@enduml
```



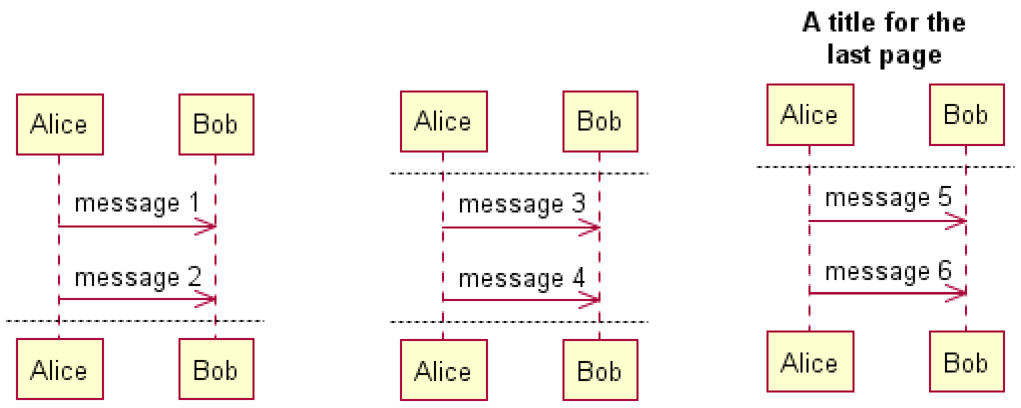
## Splitting diagrams

The `newpage` keyword is used to split a diagram into several images.

You can put a title for the new page just after the `newpage` keyword.

```
@startuml

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

newpage

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

newpage A title for the\nlast page

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

@enduml
```

This is very handy to print long diagram on several pages.



## Grouping message

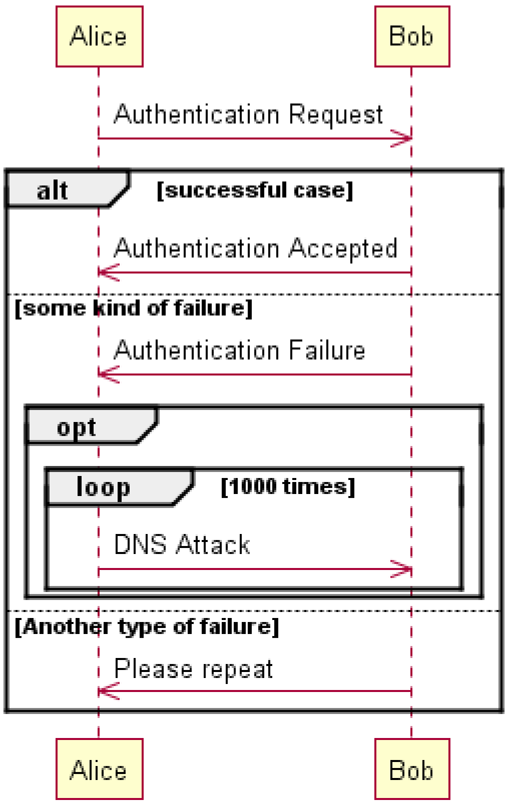
It is possible to group messages together using `alt/else`, `opt` or `loop` keywords.

It is possible a add a text that will be displayed into the header.

The `end` keyword is used to close the group.

Note that it is possible to nest groups.

```
@startuml
Alice -> Bob: Authentication Request
alt successful case
    Bob -> Alice: Authentication Accepted
else some kind of failure
    Bob -> Alice: Authentication Failure
    opt
        loop 1000 times
            Alice -> Bob: DNS Attack
        end
    end
else Another type of failure
   Bob -> Alice: Please repeat
end
@enduml
```

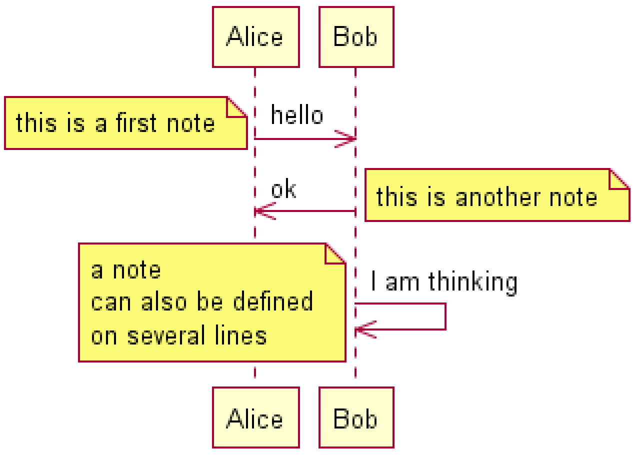


## Notes on messages

It is possible to put notes on message using the `note left` or `note right` keywords *just after the message* .

You can have multilines note using the `end note` keywords.

```
@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
```

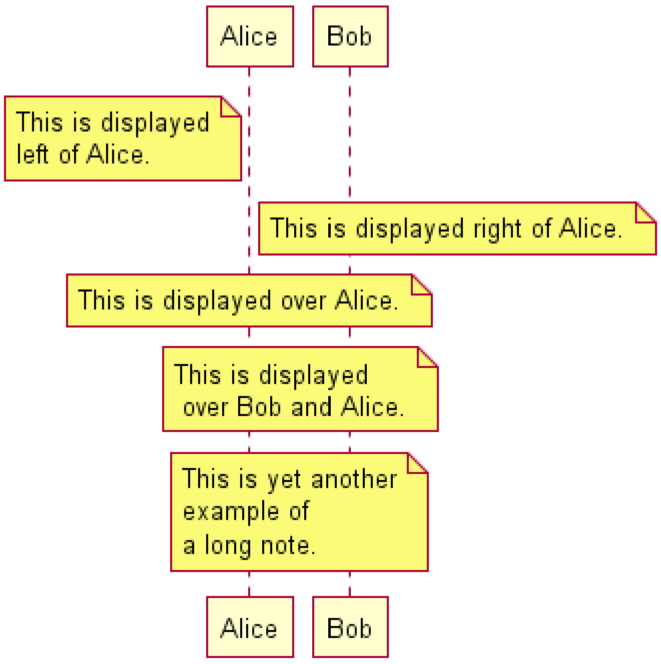


## Some other notes

It is also possible to place notes relative to participant with `note left of` , `note right of` or `note over` keywords.

You can also have multilines note using the `end note` keywords.

```
@startuml
participant Alice
participant Bob
note left of Alice
 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: This is displayed\n over Bob and Alice.

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

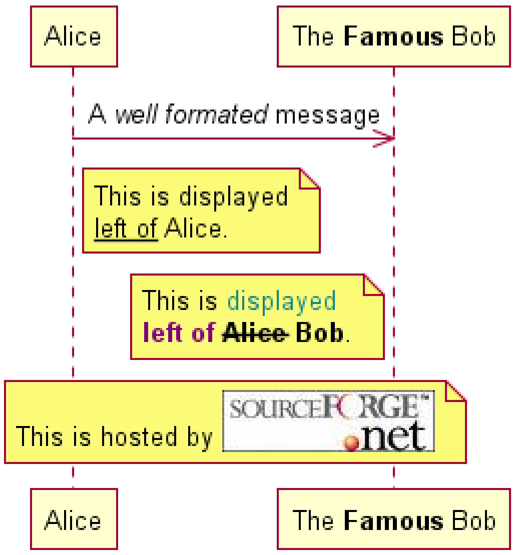

Alice
Bob
This is displayed left of Alice.
This is displayed right of Alice.
This is displayed over Alice.
This is displayed over Bob and Alice.
This is yet another example of a long note.
Alice
Bob

## Formatting using HTML

It is also possible to use few html tags like :

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

```
@startuml
participant Alice
participant "The <b>Famous</b> Bob" as Bob

Alice -> Bob : A <i>well formated</i> message
note right of Alice
        This is displayed
        <u>left of</u> Alice.
end note
note left of Bob
        This is <font color=#118888>displayed</font>
        <b><font color=purple>left of</font> <s>Alice</s> Bob</b>.
end note
note over Alice, Bob
        This is hosted by <img src=sourceforge.jpg>
end note

@enduml
```



## Lifeline Activation and Destruction

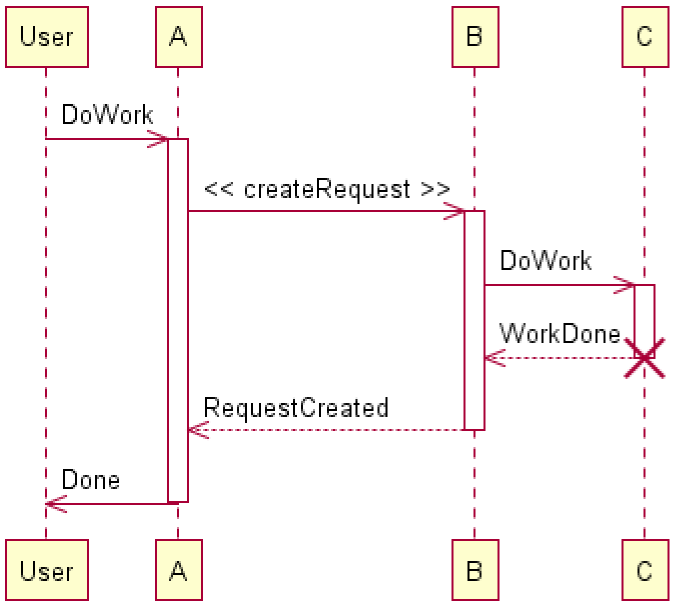
The `activate` and `deactivate` are used to denote participant activation.

Once a participant is activated, its lifeline appears.

The `activate` et `deactivate` apply on the previous message.

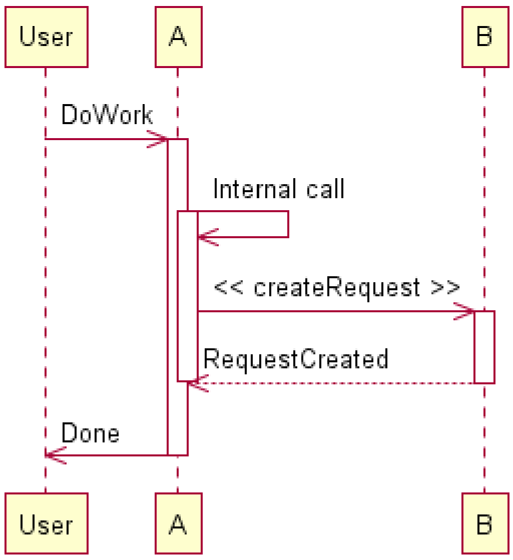
The `destroy` denote the end of the lifeline of a participant.

```
@startuml
participant User

User -> A: DoWork
activate A

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

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

B --> A: RequestCreated
deactivate B

A -> User: Done
deactivate A

@enduml
```



Nested lifeline can be used

```
@startuml
participant User
User -> A: DoWork
activate A

A -> A: Internal call
activate A

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

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

@enduml
```

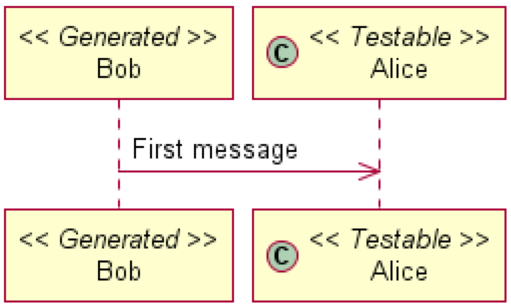


## Stereotypes and Spots

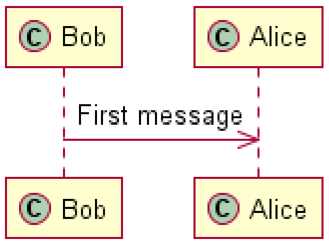
It is possible to add stereotypes to participants using << and >>.

In the stereotype, you can add a spotted character in a colored circle using the syntax `(X,color)`.

```
@startuml

participant 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
```

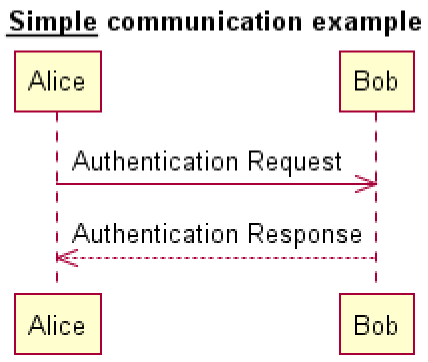


## More information on titles

You can use some HTML tags in the title like :

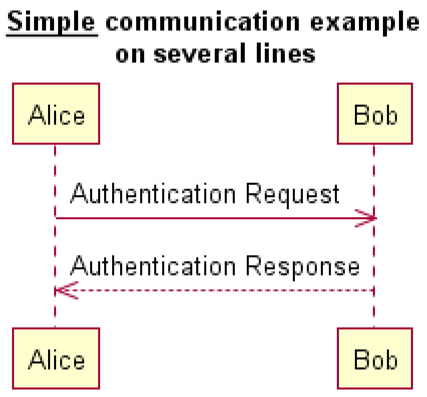
- `<b>`
- `<u>`
- `<i>`
- `<s>`, `<strike>` or `<del>`
- `<font color="#AAAAAA">` or `<font color="colorName">`
- `<img src="file">` : the file must be accessible by the filesystem

```
@startuml
title <u>Simple</u> communication example

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



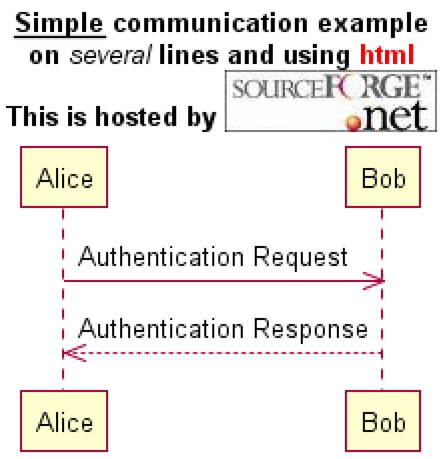
You can add newline using `\n` in the title description.

```
@startuml
title <u>Simple</u> communication example\non several lines

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



You can also define title on several lines using `title` and `end title` keywords.

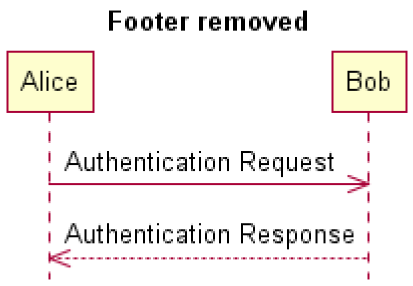
```
@startuml
title
 <u>Simple</u> communication example
 on <i>several</i> lines and using <font color=red>html</font>
 This is hosted by <img src=sourceforge.jpg>
end title

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

@enduml
```



## Removing Footer

You can use the `footbox off` keywords to remove the footer of the diagram.

```
@startuml

footbox off
title Footer removed

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

@enduml
```

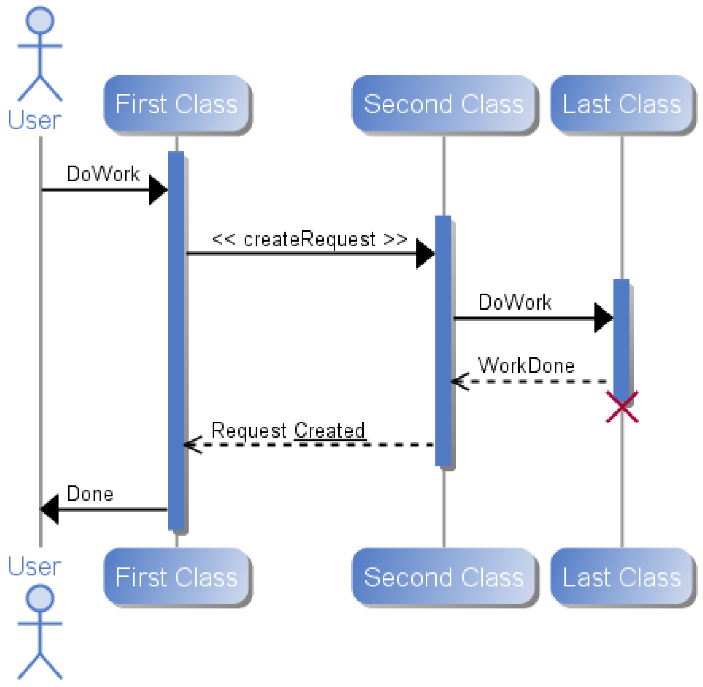


## Skin

Use the keyword `skin` to change the look of the generated diagram.

There are only two skins available today (`Rose`, which is the default, and `BlueModern`), but it is possible to write your own skin.

```
@startuml
skin BlueModern

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

User -> A: DoWork
activate A

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

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

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

A -> User: Done
deactivate A

@enduml
```



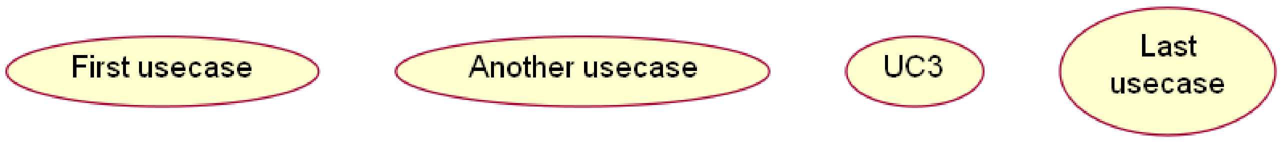
# 2- Use Case Diagram

### Usecases

Use cases are enclosed using between parentheses (because two parentheses looks like an oval).

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

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

@enduml
```



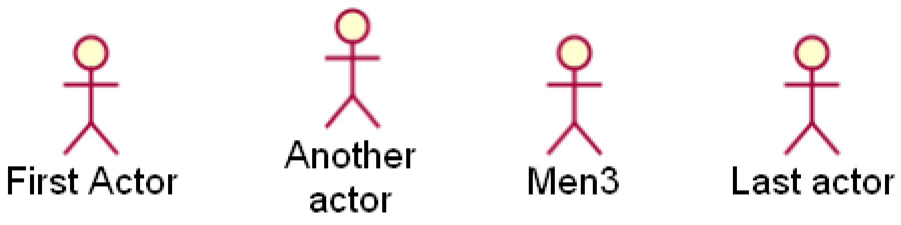
### Actors

Actor are enclosed using between two points.

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

We will see latter than actor definitions is optional.

```
@startuml

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

@enduml
```



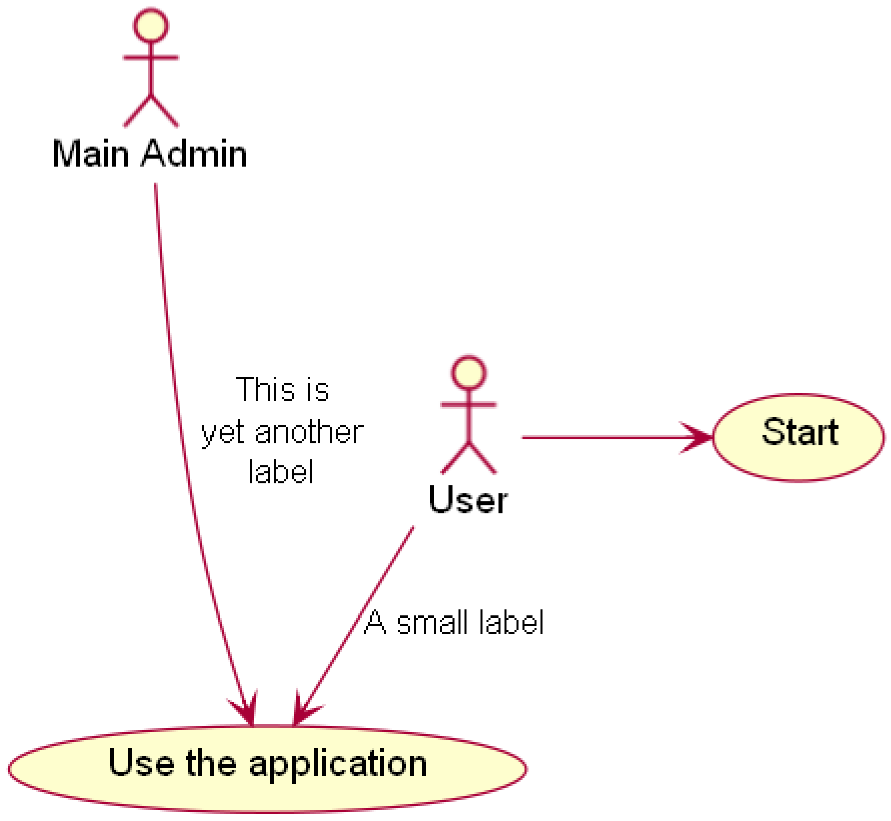
## Basic example

To link actors and use cases, the arrow `-->` is used.

The more dashes `"-"` in the arrow, the longer the arrow. You can add a label on the arrow, by adding a `":"` character in the arrow definition.

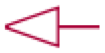
In this example, you see that *User* has not been defined before, and is used as an actor.

```
@startuml

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

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

@enduml
```

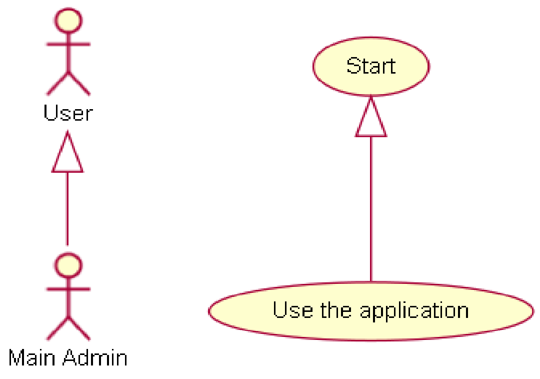


## Extension

If one actor/use case extends another one, you can use the symbol `<|--`.

This symbol stands stands for  .

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

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

@enduml
```

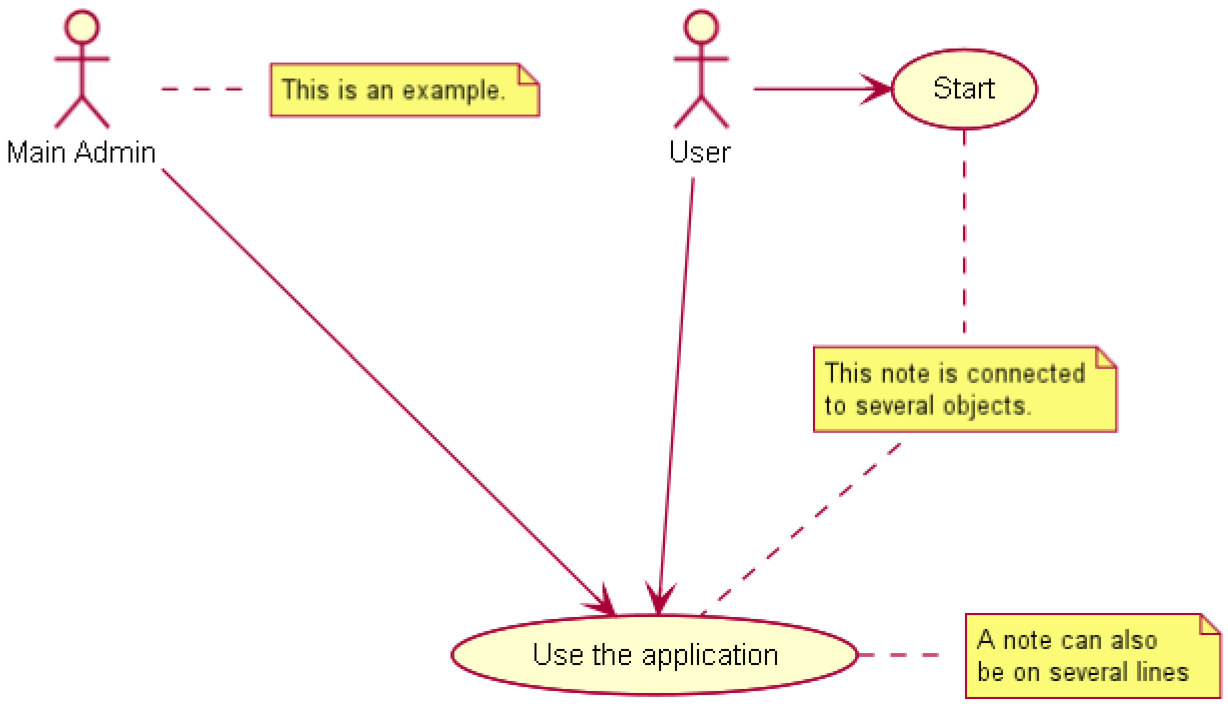


## Using notes

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

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

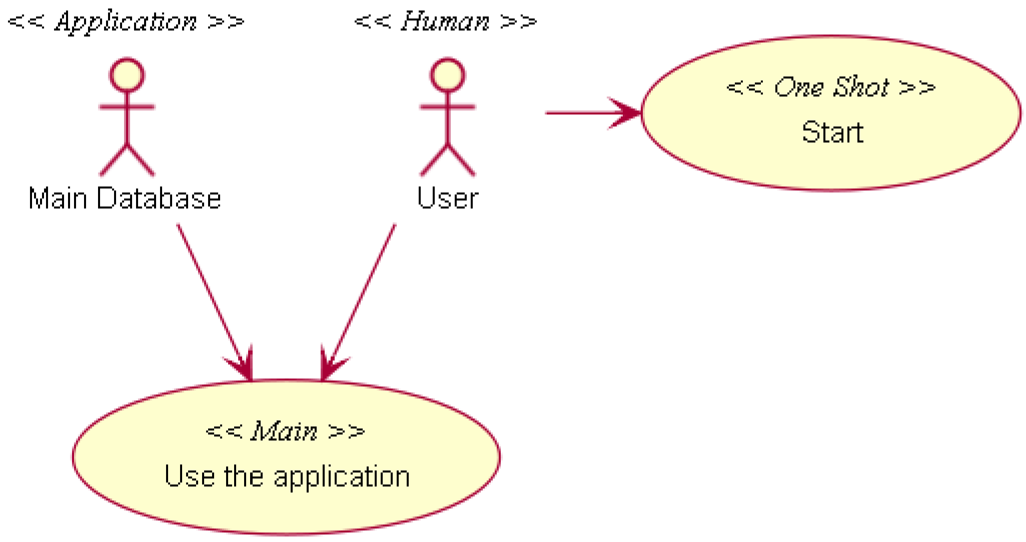
```
@startuml
:Main Admin: as Admin
(Use the application) as (Use)

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

Admin ---> (Use)

note right of Admin : This is an example.

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

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



## Stereotypes

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

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

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

MySql --> (Use)

@enduml
```

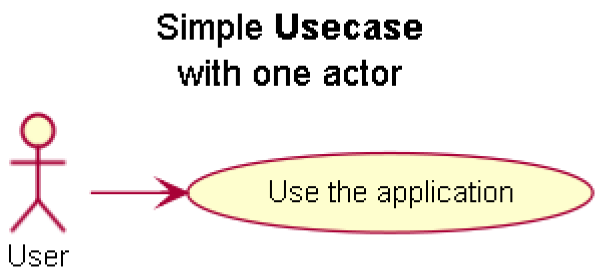


## Title the diagram

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

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

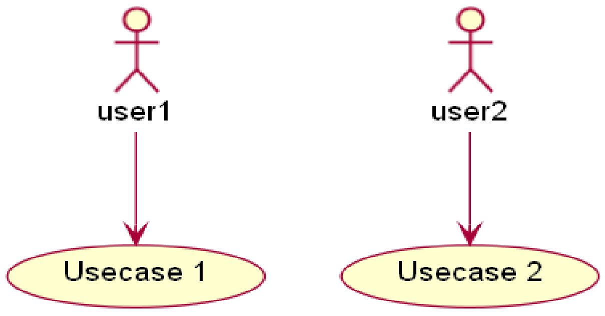
```
@startuml
title Simple <b>Usecase</b>\nwith one actor

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

@enduml
```



## Left to right direction

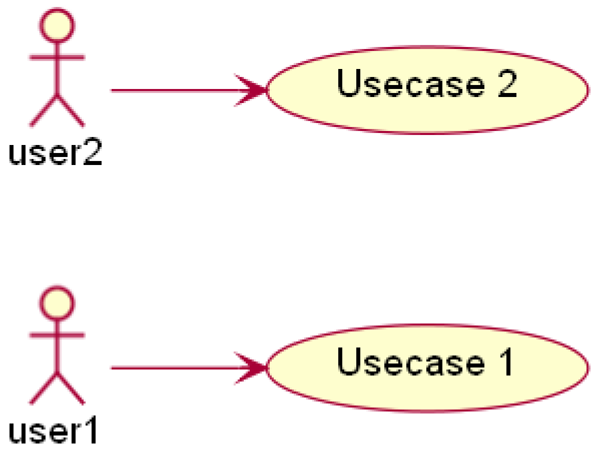
The general default behaviour when building diagram is **top to bottom**.

```
@startuml img/usecase_img09.png

#default
top to bottom direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)

@enduml
```



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

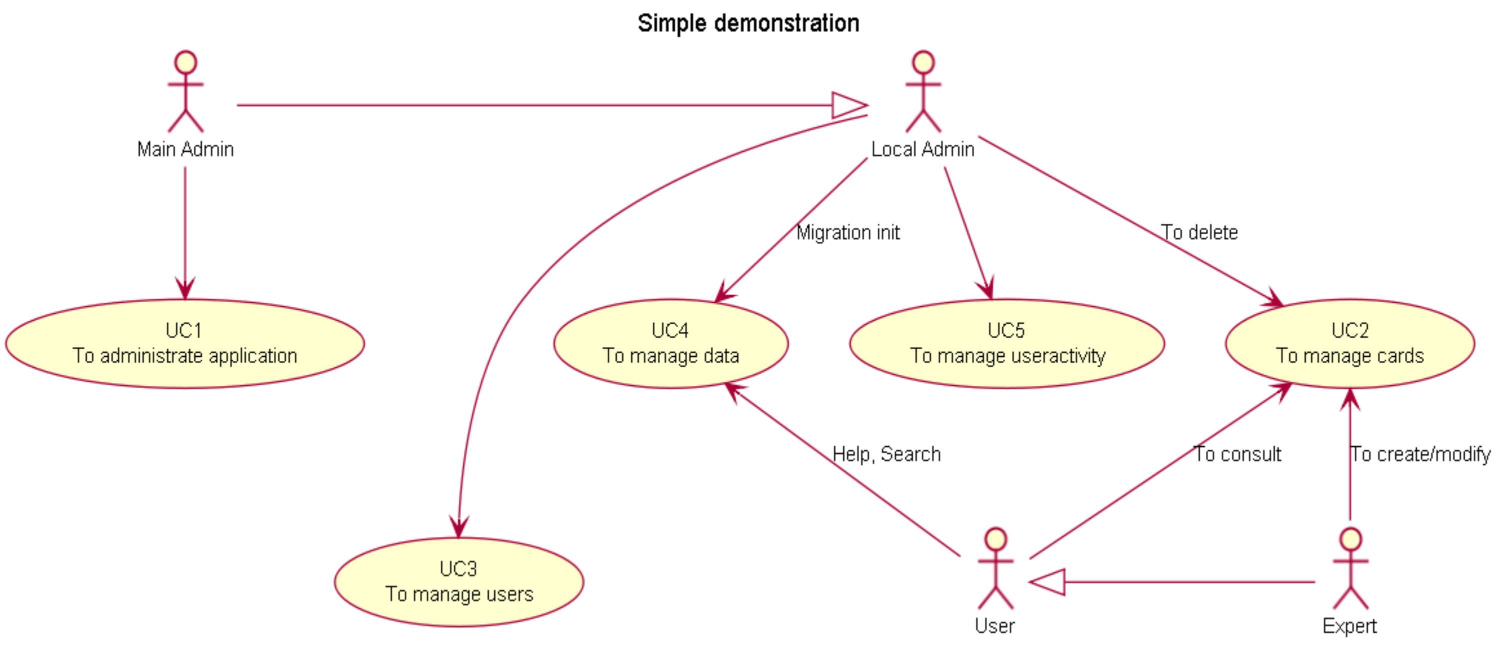
```
@startuml img/usecase_img09.png

left to right direction

user1 --> (Usecase 1)
user2 --> (Usecase 2)

@enduml
```



## A complete example

You can define the direction of any arrow as <|-- or --|> .

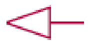
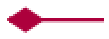
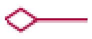
```
@startuml
title Simple demonstration
:Main Admin: as Admin
:Local Admin: as LAdmin
Admin -|> LAdmin
User <|- Expert

(UC1\nTo administrate application) as (UC1)
(UC2\nTo manage cards) as (UC2)
(UC3\nTo manage users) as (UC3)
(UC4\nTo manage data) as (UC4)
(UC5\nTo manage useractivity) as (UC5)

Admin --> (UC1)
LAdmin --> (UC2) : To delete
LAdmin ---> (UC3)
LAdmin --> (UC4) : Migration init
LAdmin --> (UC5)

(UC2) <-- Expert : To create/modify
(UC2) <-- User : To consult
(UC4) <-- User : Help, Search

@enduml
```



# 3- Class Diagram

## Relations between classes

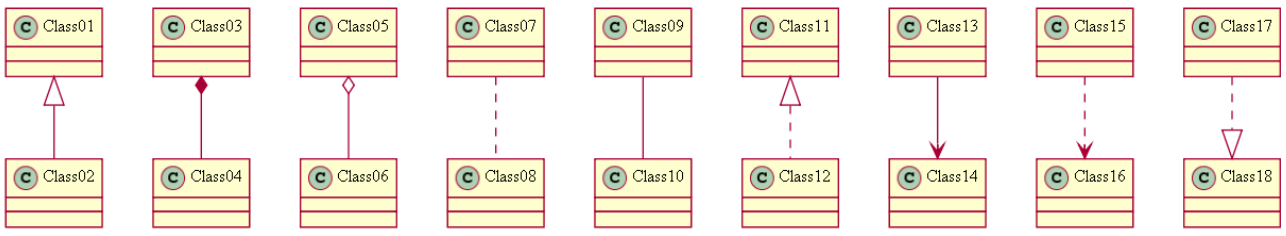
Relations between classes are defined using the following symbols :

| | | |
|---|---|---|
| Extension | `<\|--` |  |
| Composition | `*--` |  |
| Agregation | `o--` |  |

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

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

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

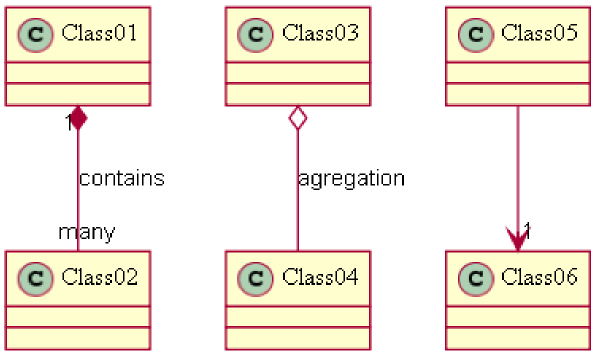


## Label on relations

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

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

```
@startuml

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

Class03 o-- Class04 : agregation

Class05 --> "1" Class06

@enduml
```

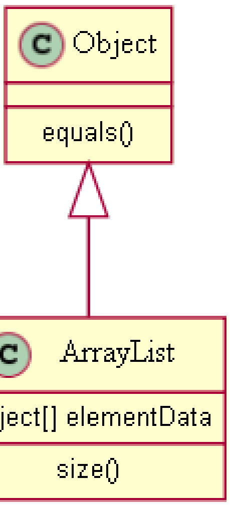


### Adding methods

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

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

```
@startuml
Object <|-- ArrayList

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

@enduml
```



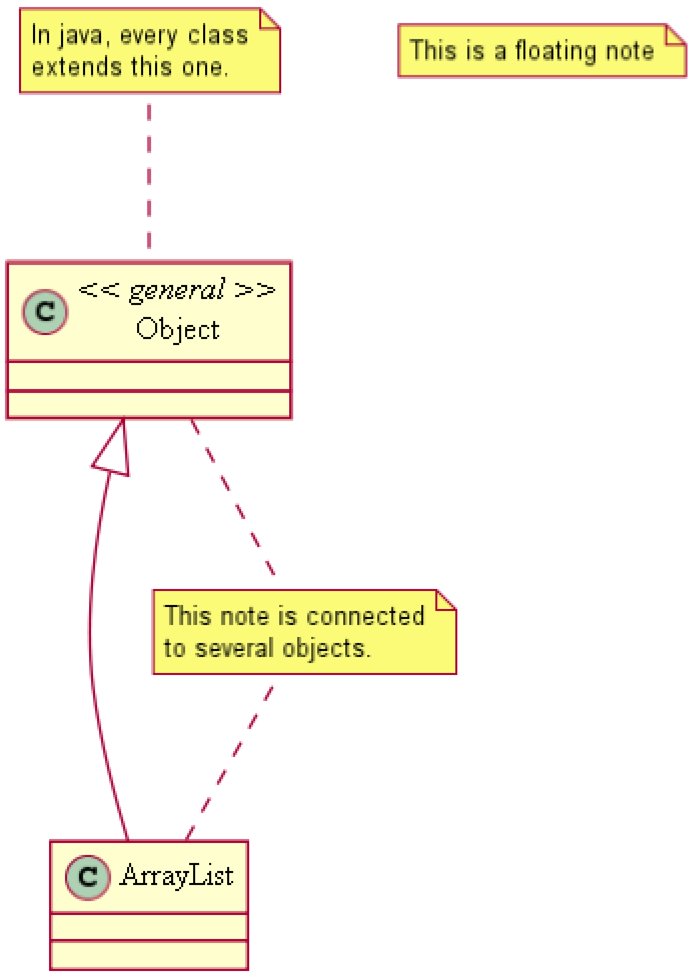
## Notes and stereotypes

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

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

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

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

note top of Object : 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

@enduml
```



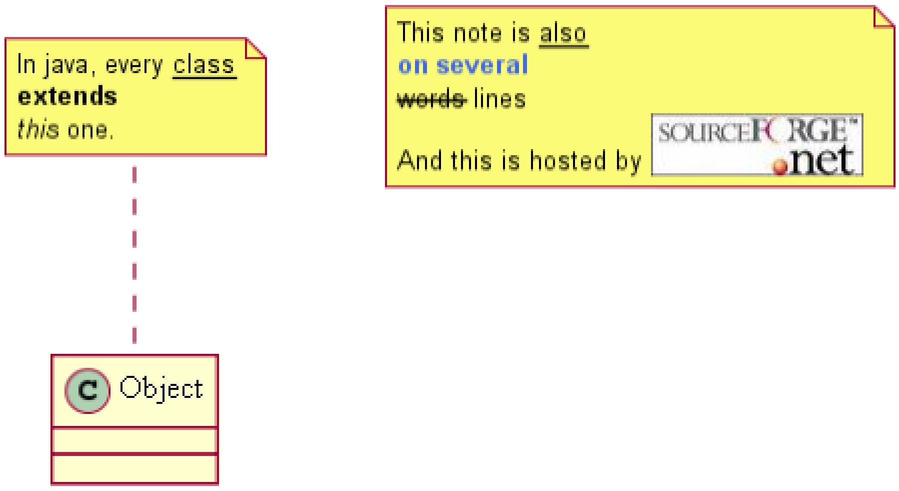
## More on notes

It is also possible to use few html tags like :

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

You can also have a note on several lines.

```
@startuml

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

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

@enduml
```

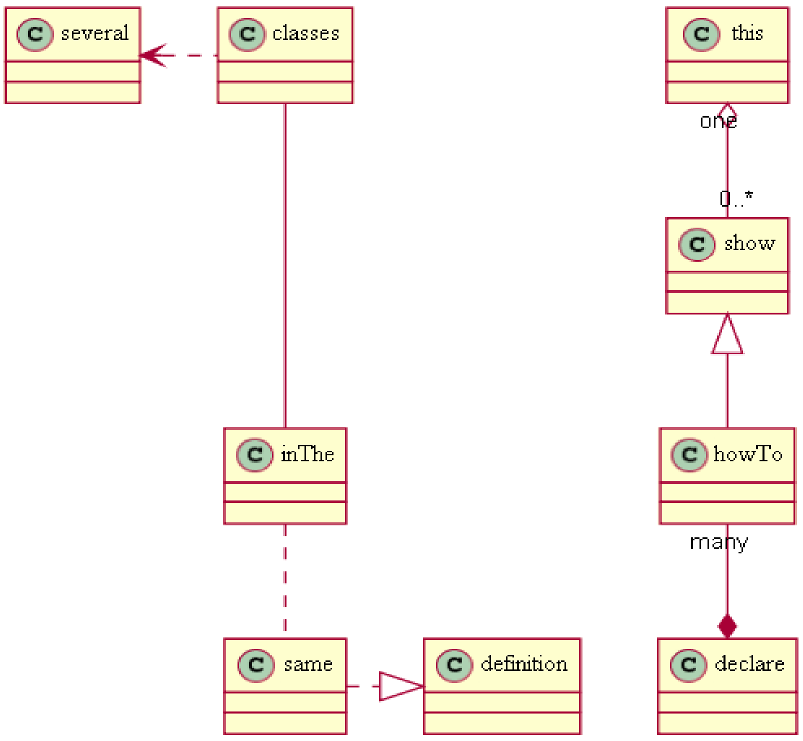


## Chained relations

It is possible to chain relation definition on the same line.

Please note that the number of dashes - or dot . in a relation does change arrow's length.

```
@startuml
this "one" o-- "0..*" show <|-- howTo "many" --* declare
several <. classes --- inThe .. same .|> definition
@enduml
```



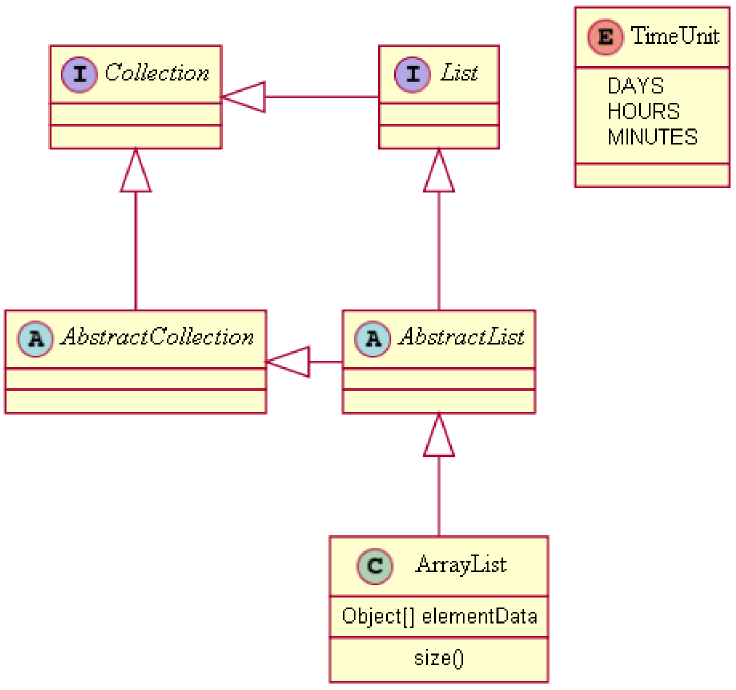
### Abstract class and interface

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

The class will be printed in *italic*.

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

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

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

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

ArrayList : Object[] elementData
ArrayList : size()

enum TimeUnit
TimeUnit : DAYS
TimeUnit : HOURS
TimeUnit : MINUTES

@enduml
```

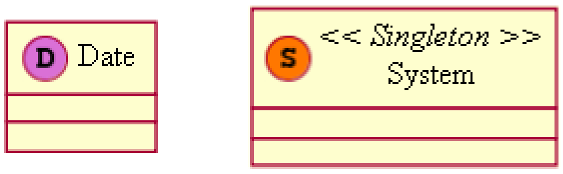


## Specific Spot

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

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

```
@startuml

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



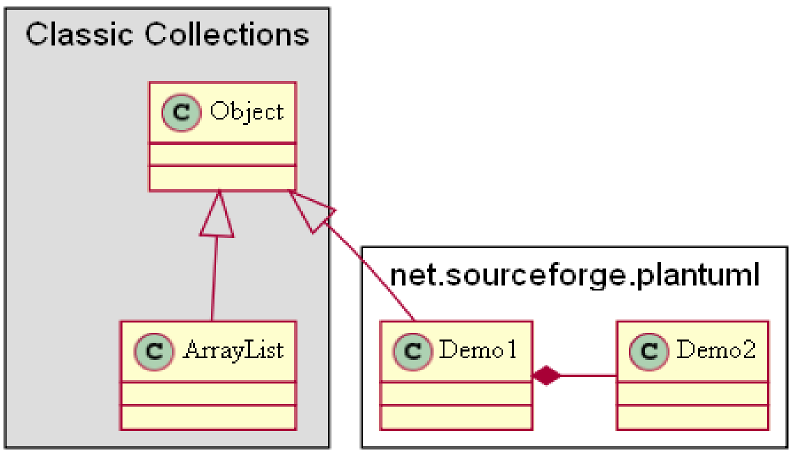
## Using packages

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

When you declare classes, they are automatically put in the last used package, and you can close the package definition using the `end package` keyword.

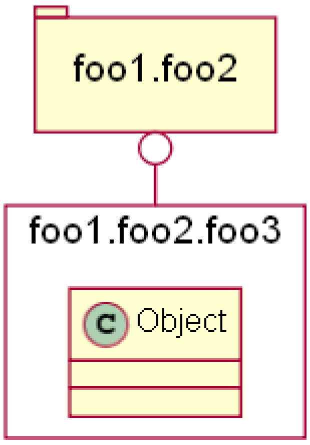
Note that package definitions can be nested.

```
@startuml

package "Classic Collections" #DDDDDD
Object <|-- ArrayList
end package

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

@enduml
```



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

```
@startuml

package foo1.foo2
end package

package foo1.foo2.foo3
  class Object
end package

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```

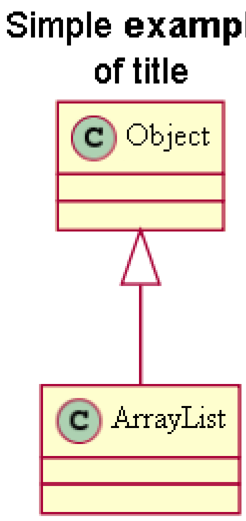


## Title the diagram

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

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

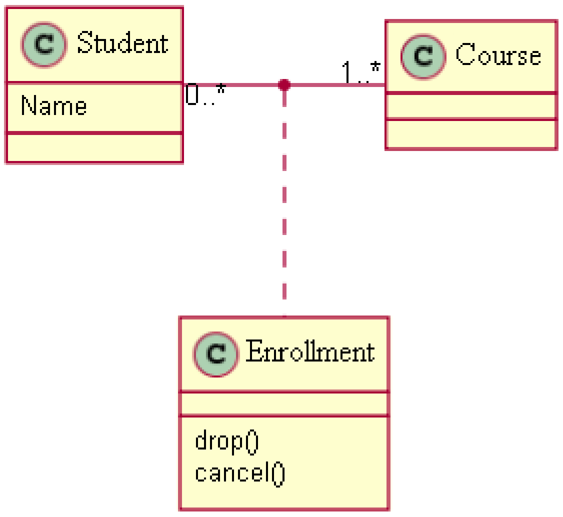
```
@startuml img/classes15.png
title Simple <b>example</b>\nof title

Object <|-- ArrayList

@enduml
```



## Association classes

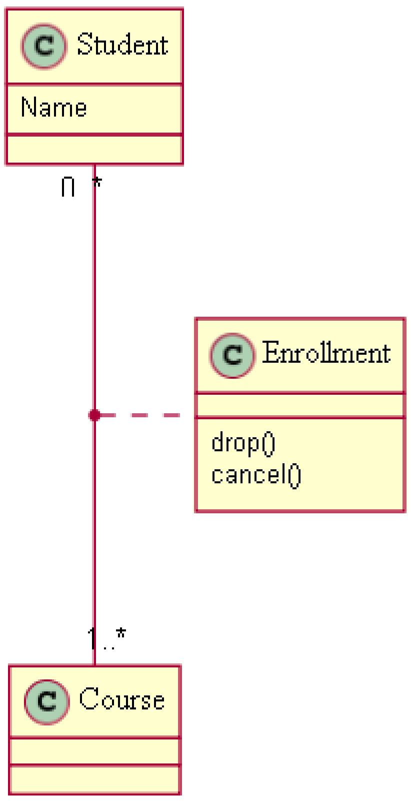
You can define *association class* after that a relation has been defined between two classes, like in this example:

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

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



You can define it in another direction:

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

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



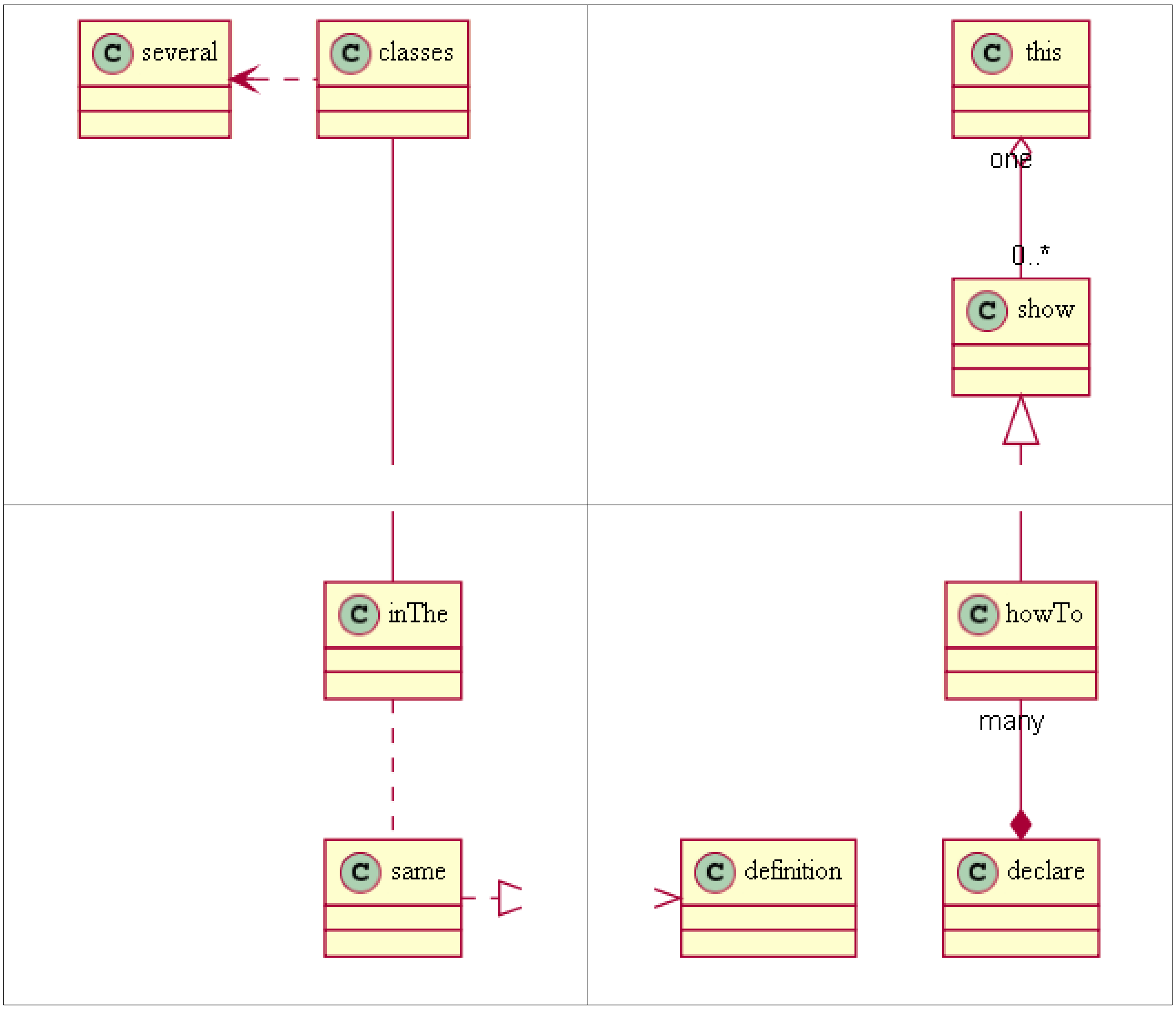
### Splitting large files

Sometimes, you will get some very large image files.

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

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

```
@startuml
' Split into 4 pages
page 2x2
this "one" o-- "0..*" show <|-- howTo "many" --* declare
several <. classes --- inThe .. same .|> definition
@enduml
```



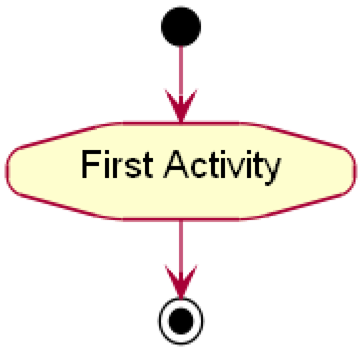
# 4- Activity Diagram

## Simple Activity

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

Use `-->` for arrows.

```
@startuml

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

@enduml
```



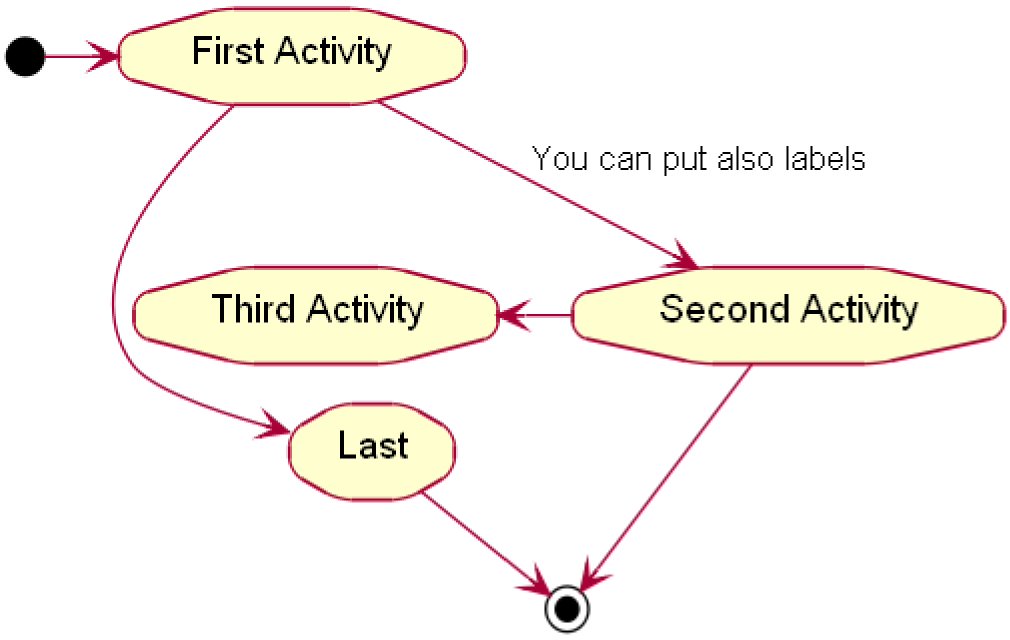
## Arrows

You can use -> for horizontal arrows.

You can use ---> for longer arrows.

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

```
@startuml

(*) -> "First Activity"
--> "Second Activity" : You can put also labels
"Third Activity" <- "Second Activity"
--> (*)

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

@enduml
```

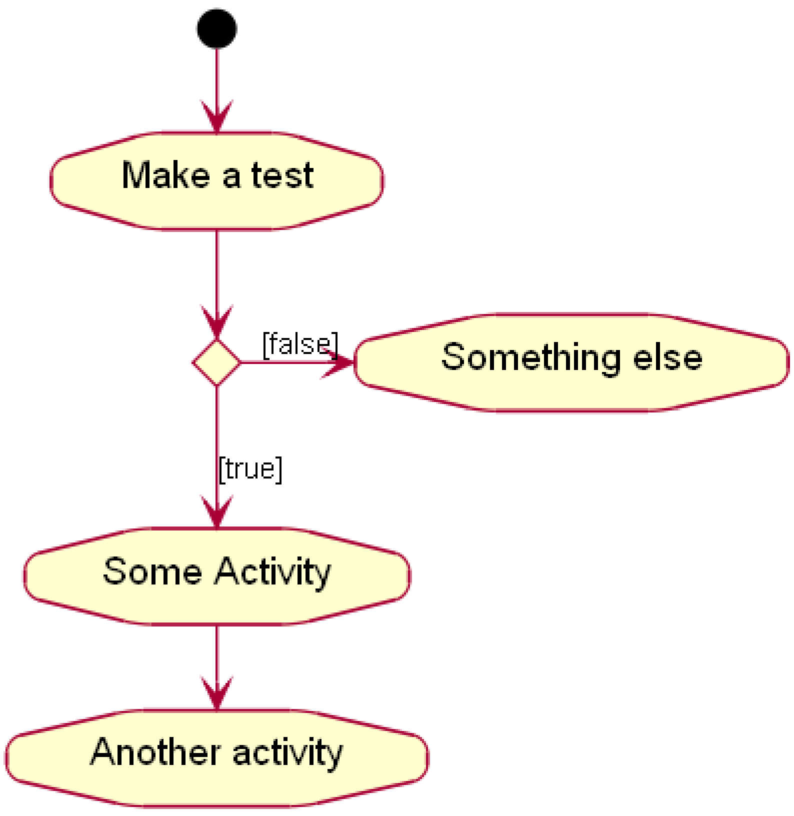


### Branches

You can use <> followed with an internal label for branches.

The notation `[some label]` always refers to the last used branche.

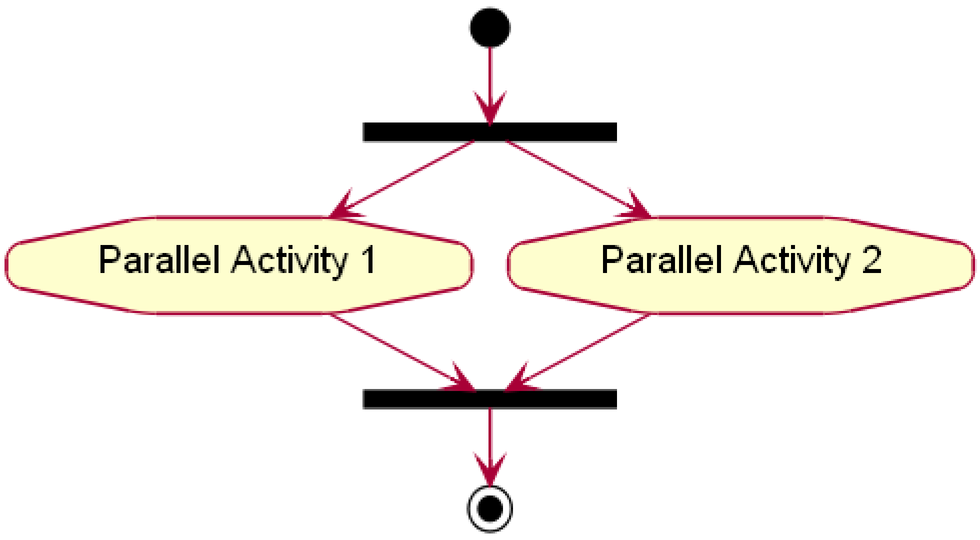
```
@startuml

(*) --> "Make a test"
--> <> B1

--> [true] "Some Activity"
--> "Another activity"

-> [false] "Something else"

@enduml
```



## Synchronization

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

```
@startuml

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

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

--> (*)

@enduml
```



## Long activity description

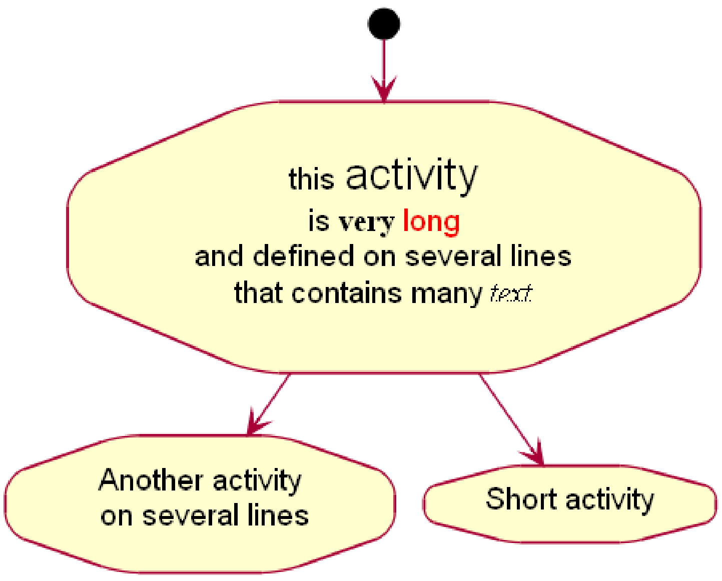
When you declare activities, you can span on several lines the description text. You can also add `\n` in the description.

It is also possible to use few html tags like :

- `<b>`
- `<i>`
- `<font size="16">`
- `<font color="#AAAAAA">` or `<font color="colorName">`

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

```
@startuml

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

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

A1 --> "Short activity"

@enduml
```



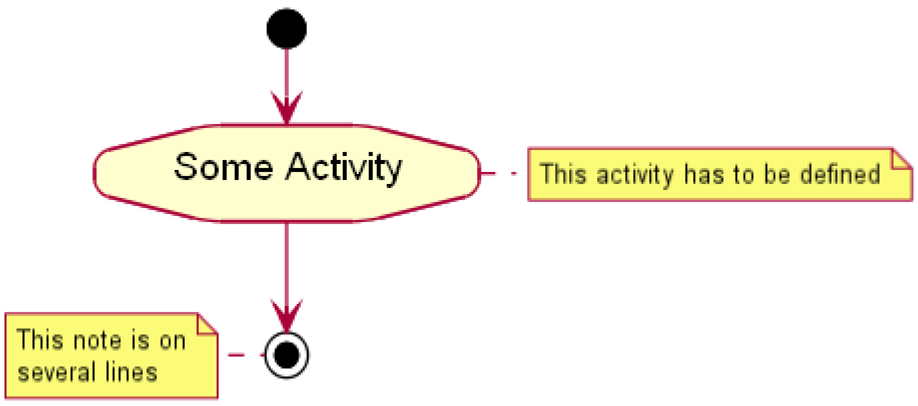
## Notes

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

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

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

```
@startuml

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

@enduml
```


Some Activity
This activity has to be defined
This note is on several lines

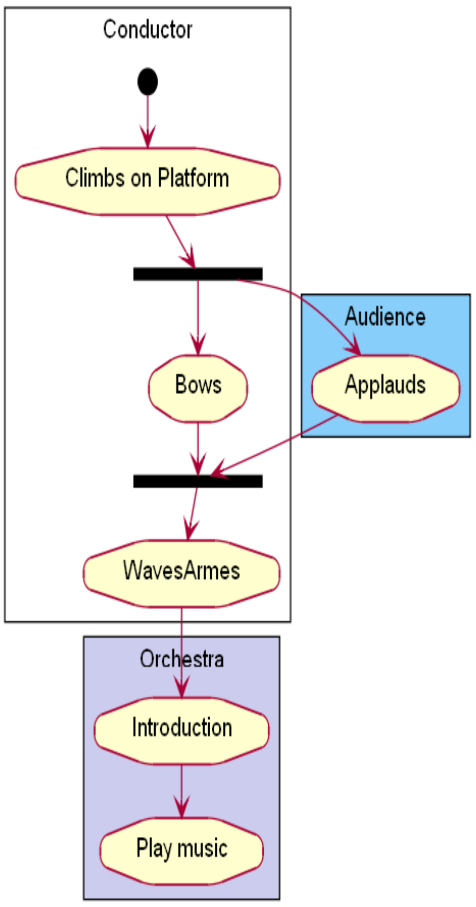
### Partition

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

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

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

```
@startuml

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

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

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

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

@enduml
```

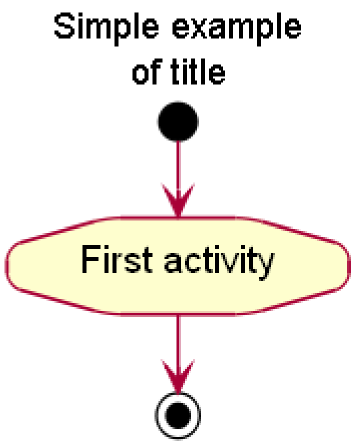


## Title the diagram

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

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

```
@startuml
title Simple example\nof title

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



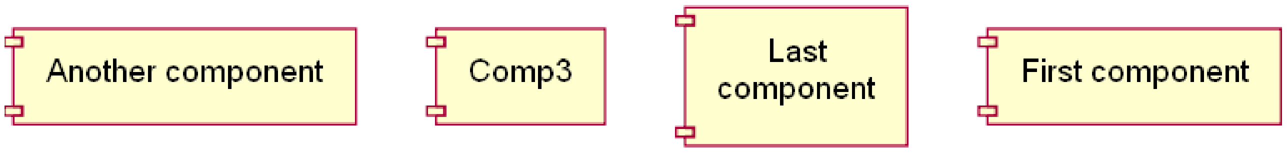
# 5- Component Diagram

## Components

Components must be bracketed.

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

```
@startuml

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

@enduml
```



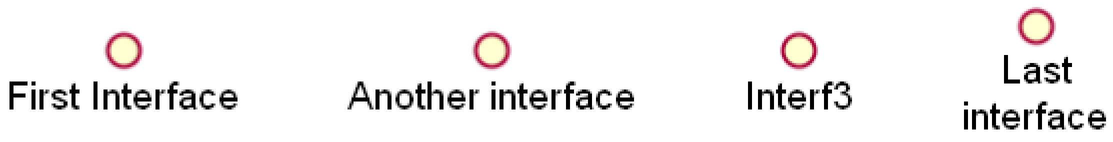
## Interfaces

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

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

We will see latter that interface definition is optional.

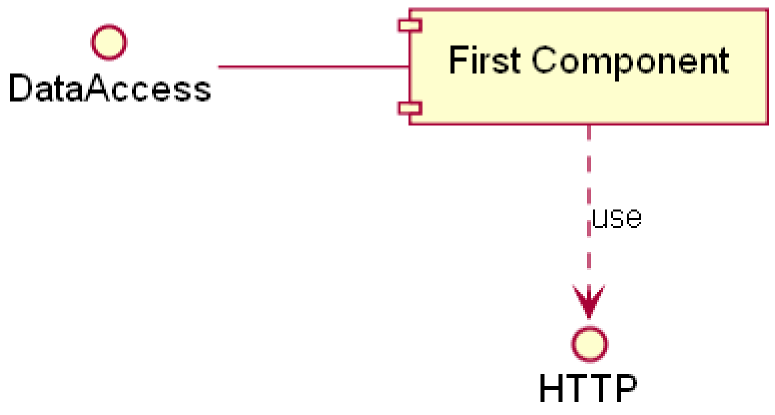
```
@startuml

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

@enduml
```



## Basic example

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

```
@startuml

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

@enduml
```

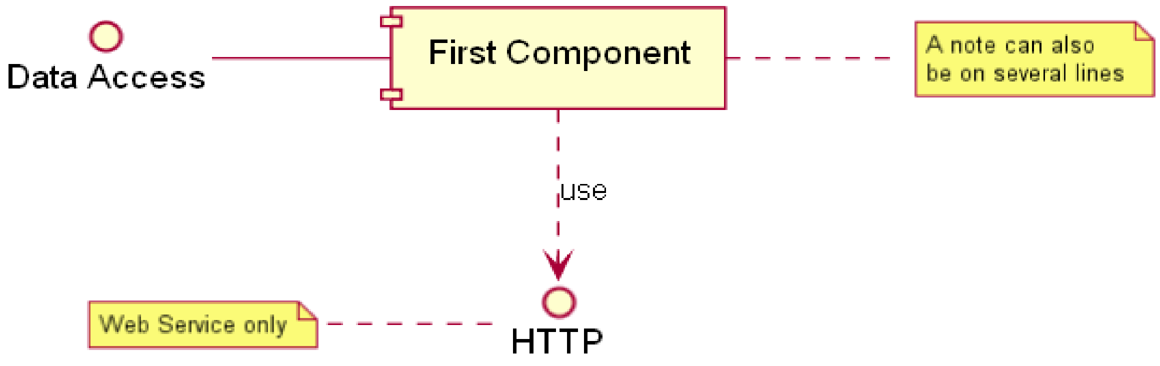


## Using notes

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

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

```
@startuml

interface "Data Access" as DA

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

note left of HTTP : Web Service only

note right of [First Component]
  A note can also
  be on several lines
end note
@enduml
```

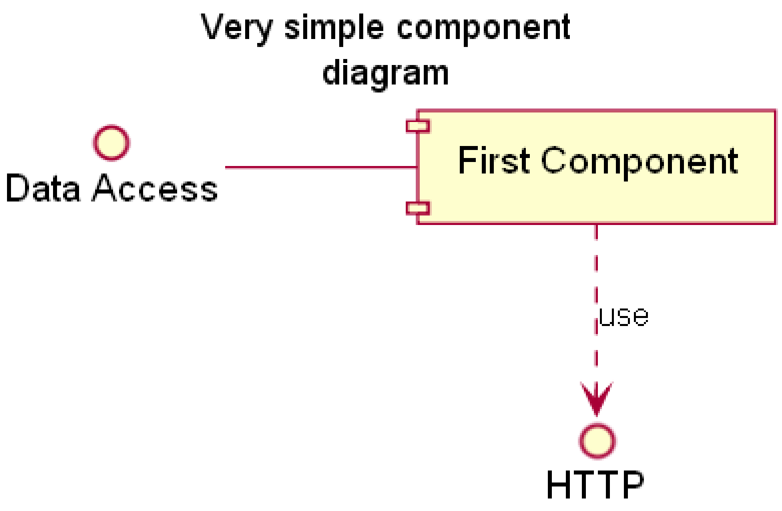


## Title the diagram

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

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

```
@startuml
title Very simple component\ndiagram

interface "Data Access" as DA

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

@enduml
```



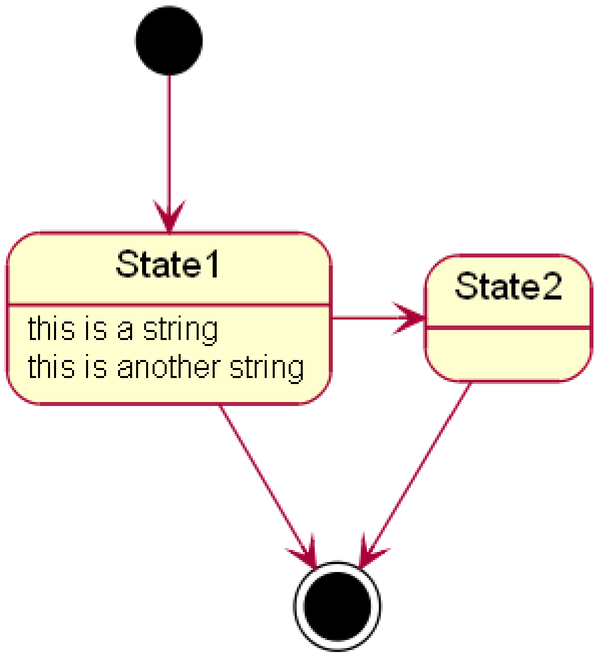
# 6- State Diagram

## Simple State

You can use [*] for the starting point and ending point of the state diagram.

Use --> for arrows.

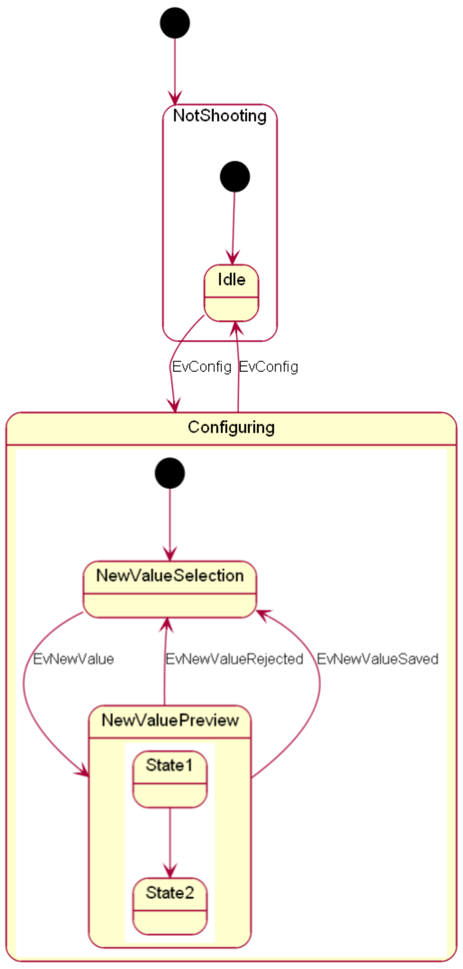
```
@startuml

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

State1 -> State2
State2 --> [*]

@enduml
```



## Composite State

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

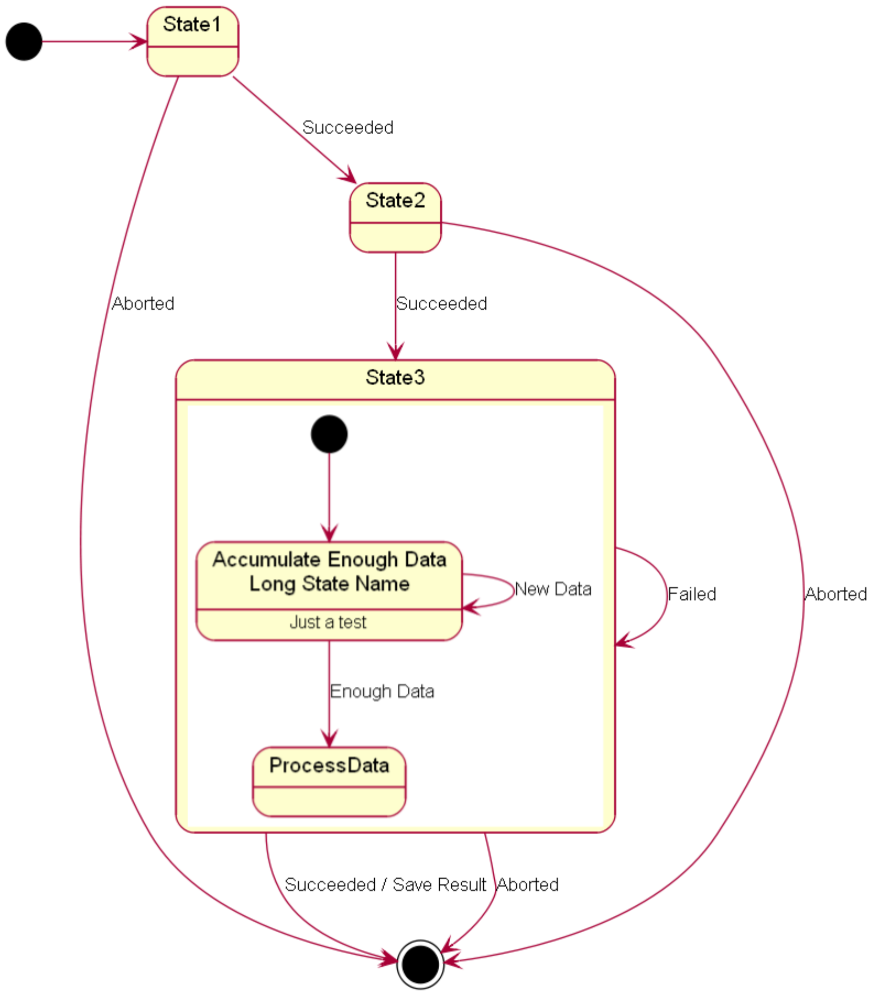
```
@startuml
[*] --> NotShooting

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

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

  state NewValuePreview {
     State1 -> State2
  }
 }
@enduml
```



## Long name

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

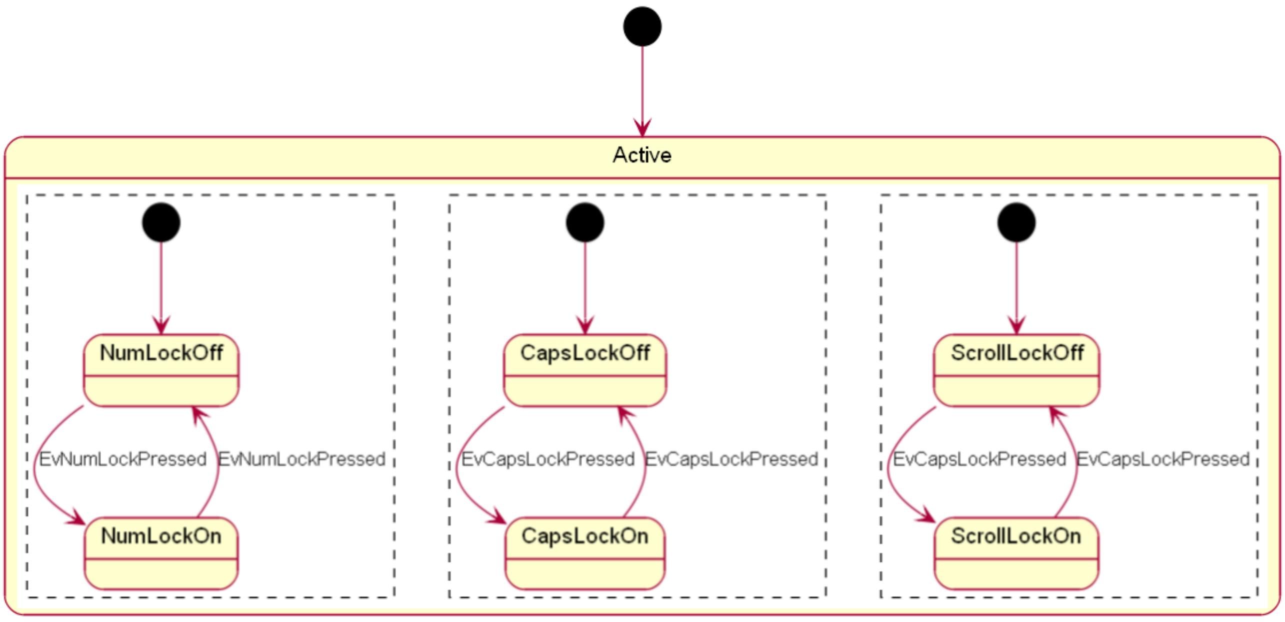
```
@startuml

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

@enduml
```



## Concurrent State

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

```
@startuml

[*] --> Active

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

@enduml
```

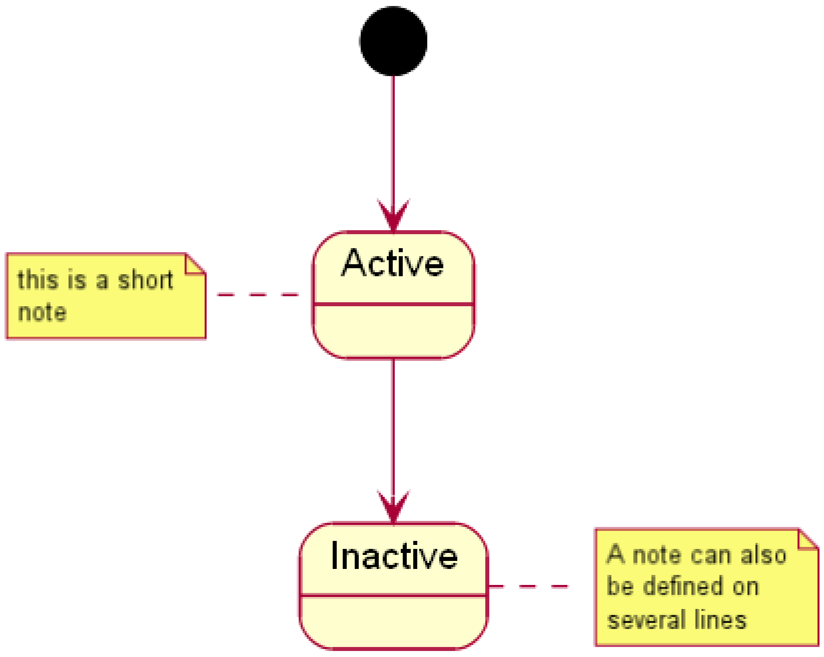


### Note

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

You can also define notes on several lines.

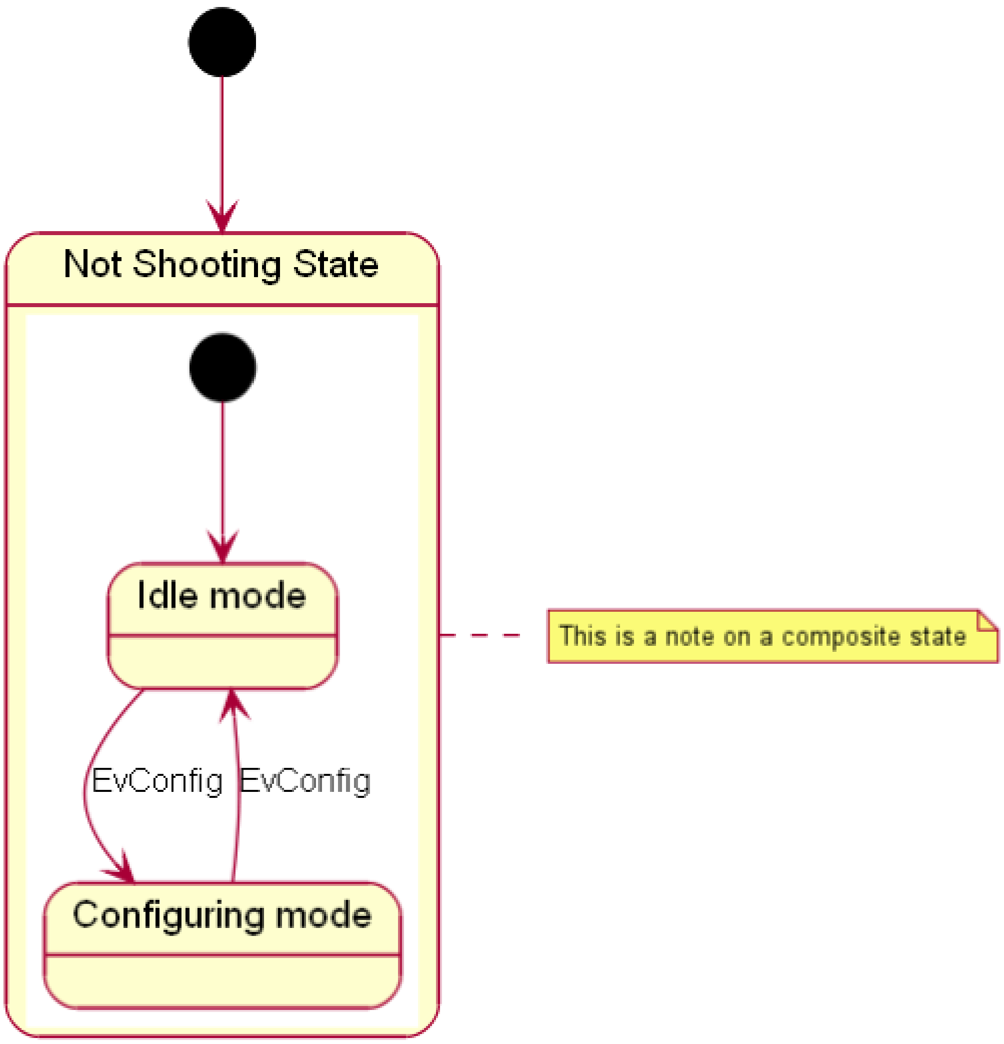
```
@startuml

[*] --> 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
```



## More in notes

You can put notes on composite states.

```
@startuml

[*] --> NotShooting

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

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

@enduml
```



# 7- Common commands

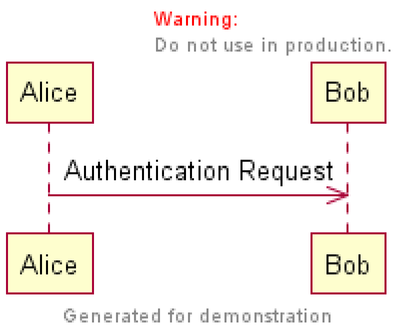
## Footer and header

You can use the commands `header` or `footer` to add a footer or a header on any generated diagram.
You can optionally specify if you want a `center`, `left` or `right` footer/header, by adding a keywork.

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

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

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

header
<font color=red>Warning:</font> This is a demonstration diagram.
Do not use in production.
endheader

center footer Generated for demonstration

@enduml
```

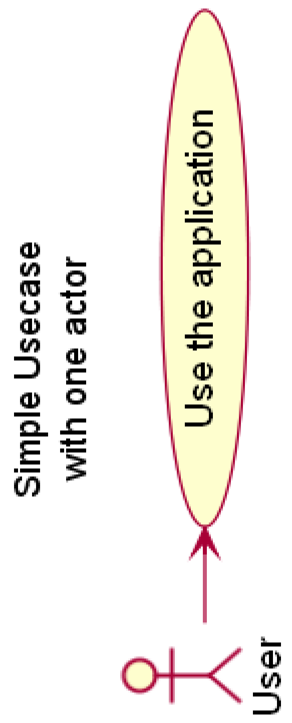


## Rotation

Sometimes, and especially for printing, you may want to rotate the generated image, so that it fits better in the page.

You can use the `rotate` command for this.

```
@startuml img/commons_002.png
rotate

title Simple Usecase\nwith one actor

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

@enduml
```



# 8- Changing fonts and colors

## Usage

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

Example:

```
skinparam backgroundColor yellow
```

You can use this command :

- In the diagram definition, like any other commands,
- In an included file (see ***9- Preprocessing***),
- In a configuration file, provided in the command line or the ANT task.

## Colors

You can use either standard color name or RGB code.

| **Parameter name** | **Default value** | **Color** | **Comment** |
|---|---|---|---|
| backgroundColor | white | ⬜ | Background of the page |
| activityArrowColor | #A80036 | 🟥 | Color of arrows in activity diagrams |
| activityBackgroundColor | #FEFECE | 🟨 | Background of activities |
| activityBorderColor | #A80036 | 🟥 | Color of activity borders |
| activityStartColor | black | ⬛ | Starting circle in activity diagrams |
| activityEndColor | black | ⬛ | Ending circle in activity diagrams |
| activityBarColor | black | ⬛ | Synchronization bar in activity diagrams |
| usecaseArrowColor | #A80036 | 🟥 | Color of arrows in usecase diagrams |
| actorBackgroundColor | #FEFECE | 🟨 | Head's color of actor in usecase diagrams |
| actorBorderColor | #A80036 | 🟥 | Color of actor borders in usecase diagrams |
| usecaseBackgroundColor | #FEFECE | 🟨 | Background of usecases |
| usecaseBorderColor | #A80036 | 🟥 | Color of usecase borders in usecase diagrams |
| classArrowColor | #A80036 | 🟥 | Color of arrows in class diagrams |
| classBackgroundColor | #FEFECE | 🟨 | Background of classes/interface/enum in class diagrams |
| classBorderColor | #A80036 | 🟥 | Borders of classes/interface/enum in class diagrams |
| packageBackgroundColor | #FEFECE | 🟨 | Background of packages in class diagrams |
| packageBorderColor | #A80036 | 🟥 | Borders of packages in class diagrams |
| stereotypeCBackgroundColor | #ADD1B2 | 🟩 | Background of class spots in class diagrams |
| stereotypeABackgroundColor | #A9DCDF | 🟦 | Background of abstract class spots in class diagrams |
| stereotypeIBackgroundColor | #B4A7E5 | 🟪 | Background of interface spots in class diagrams |

| | | | |
|---|---|---|---|
| stereotypeEBackgroundColor | #EB937F | 🟧 | Background of enum spots in class diagrams |
| componentArrowColor | #A80036 | 🟥 | Color of arrows in component diagrams |
| componentBackgroundColor | #FEFECE | 🟨 | Background of components |
| componentBorderColor | #A80036 | 🟥 | Borders of components |
| interfaceBackgroundColor | #FEFECE | 🟨 | Background of interface in component diagrams |
| interfaceBorderColor | #A80036 | 🟥 | Border of interface in component diagrams |
| noteBackgroundColor | #FBFB77 | 🟨 | Background of notes |
| noteBorderColor | #A80036 | 🟥 | Border of notes |
| stateBackgroundColor | #FEFECE | 🟨 | Background of states in state diagrams |
| stateBorderColor | #A80036 | 🟥 | Border of states in state diagrams |
| stateArrowColor | #A80036 | 🟥 | Colors of arrows in state diagrams |
| sequenceArrowColor | #A80036 | 🟥 | Color of arrows in sequence diagrams |
| sequenceActorBackgroundColor | #FEFECE | 🟨 | Head's color of actor in sequence diagrams |
| sequenceActorBorderColor | #A80036 | 🟥 | Border of actor in sequence diagrams |
| sequenceGroupBackgroundColor | #EEEEEE | ⬜ | Header color of alt/opt/loop in sequence diagrams |
| sequenceLifeLineBackgroundColor | white | ⬜ | Background of life line in sequence diagrams |
| sequenceLifeLineBorderColor | #A80036 | 🟥 | Border of life line in sequence diagrams |
| sequenceParticipantBackgroundColor | #FEFECE | 🟨 | Background of participant in sequence diagrams |
| sequenceParticipantBorderColor | #A80036 | 🟥 | Border of participant in sequence diagrams |

## Font color, name and size

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

Example:

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

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

| **Param name** | **Default value** | **Comment** |
|---|---|---|
| activityFontColor<br>activityFontSize<br>activityFontName | black<br>14 | Used for activity box |
| activityArrowFontColor<br>activityArrowFontSize<br>activityArrowFontName | black<br>13 | Used for text on arrows in activity diagrams |
| classArrowFontColor<br>classArrowFontSize<br>classArrowFontName | black<br>10 | Used for text on arrows in class diagrams |
| classAttributeFontColor<br>classAttributeFontSize<br>classAttributeFontName | black<br>10 | Class attributes and methods |
| classFontColor<br>classFontSize<br>classFontName | black<br>12 | Used for classes name |
| componentFontColor<br>componentFontSize<br>componentFontName | black<br>14 | Used for components name |
| componentArrowFontColor<br>componentArrowFontSize<br>componentArrowFontName | black<br>13 | Used for text on arrows in component diagrams |
| noteFontColor<br>noteFontSize<br>noteFontName | black<br>13 | Used for notes in all diagrams but sequence diagrams |
| packageFontColor<br>packageFontSize<br>packageFontName | black<br>14 | Used for package and partition names |

| | | |
|---|---|---|
| sequenceActorFontColor<br>sequenceActorFontSize<br>sequenceActorFontName | black<br>13 | Used for actor in sequence diagrams |
| sequenceArrowFontColor<br>sequenceArrowFontSize<br>sequenceArrowFontName | black<br>13 | Used for text on arrows in sequence diagrams |
| sequenceGroupingFontColor<br>sequenceGroupingFontSize<br>sequenceGroupingFontName | black<br>11 | Used for text for "else" in sequence diagrams |
| sequenceGroupingHeaderFontColor<br>sequenceGroupingHeaderFontSize<br>sequenceGroupingHeaderFontName | black<br>13 | Used for text for "alt/opt/loop" headers in sequence diagrams |
| sequenceParticipantFontColor<br>sequenceParticipantFontSize<br>sequenceParticipantFontName | black<br>13 | Used for text on participant in sequence diagrams |
| sequenceTitleFontColor<br>sequenceTitleFontSize<br>sequenceTitleFontName | black<br>13 | Used for titles in sequence diagrams |
| titleFontColor<br>titleFontSize<br>titleFontName | black<br>18 | Used for titles in all diagrams but sequence diagrams |
| stateFontColor<br>stateFontSize<br>stateFontName | black<br>14 | Used for states in state diagrams |
| stateArrowFontColor<br>stateArrowFontSize<br>stateArrowFontName | black<br>13 | Used for text on arrows in state diagrams |
| usecaseFontColor<br>usecaseFontSize<br>usecaseFontName | black<br>14 | Used for usecase labels in usecase diagrams |
| usecaseActorFontColor<br>usecaseActorFontSize<br>usecaseActorFontName | black<br>14 | Used for actor labels in usecase diagrams |
| usecaseArrowFontColor<br>usecaseArrowFontSize<br>usecaseArrowFontName | black<br>13 | Used for text on arrows in usecase diagrams |

# 9- Preprocessing

Some minor preprocessing capabilities are included in PlantUML, and available for all diagrams.

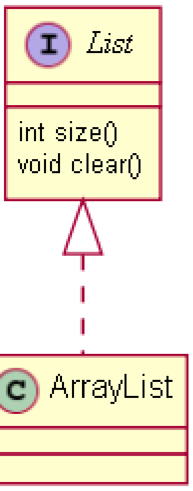
Thoses functionnalities are very similar to the C language preprocessor, except that the special character (#) has been changed to the exclamation mark (!).

## Including files

Use the `!include` directive to include file in your diagram.

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.

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



**File List.iuml:**

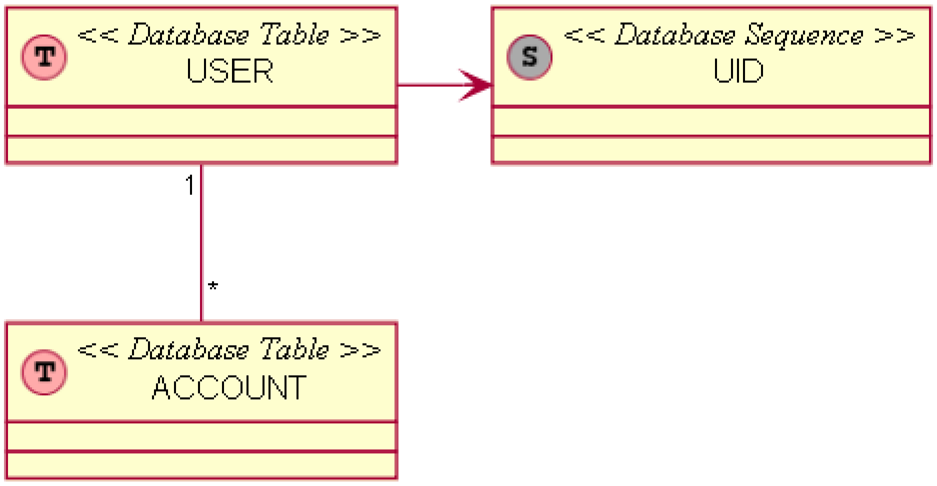
```
interface List
List : int size()
List : void clear()
```

The file List.iuml can be included in many diagrams, and any modification in this file will change all diagrams that include it.

## Constant definition

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

```
@startuml

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

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

@enduml
```



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

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

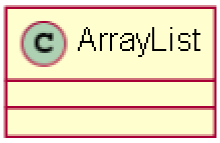
## Conditions

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

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

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

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

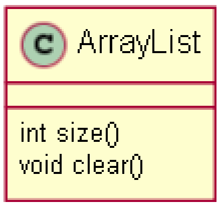


**File ArrayList.iuml:**

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

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

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

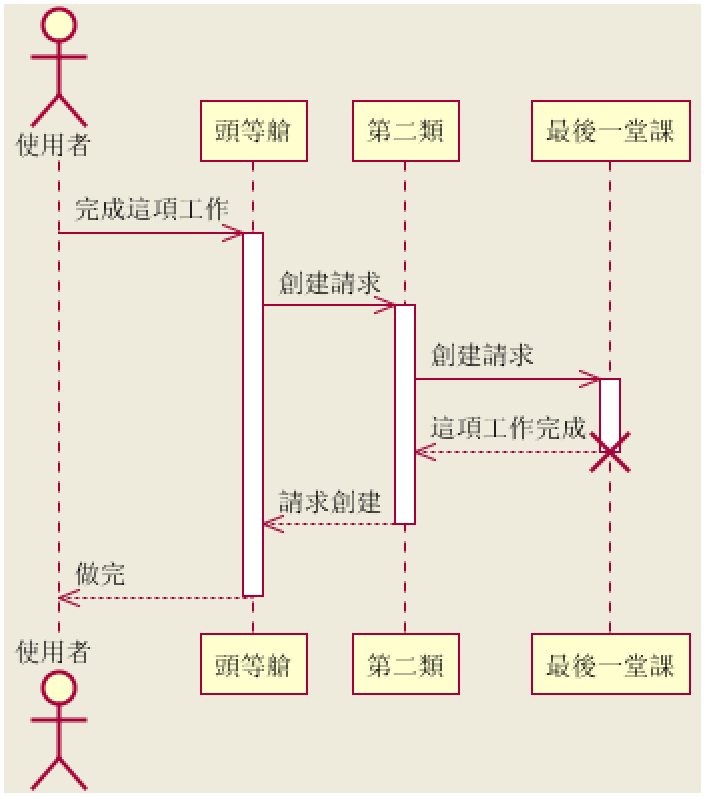


You can also use the `!ifndef` directive that includes lines if the provided constant has NOT been defined.

# 10- Internationalization

The PlantUML language use *letters* to define actor, usecase and so one.
But *letters* are not only A-Z latin characters, it could be *any kind of letter from any language*.

```
@startuml

skinparam backgroundColor #EEEBDC

actor 使用者
participant "頭等艙" as A
participant "第二類" as B
participant "最後一堂課" as 別的東西

使用者 -> A: 完成這項工作
activate A

A -> B: 創建請求
activate B

B -> 別的東西: 創建請求
activate 別的東西
別的東西 --> B: 這項工作完成
destroy 別的東西

B --> A: 請求創建
deactivate B

A --> 使用者: 做完
deactivate A
@enduml
```



## Charset

The default charset used when *reading* the text files containing the UML text description is system dependant.
Normally, it should just be fine, but in some case, you may want to the use another charset. For example, with the command line:

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

Or, with the ant task:

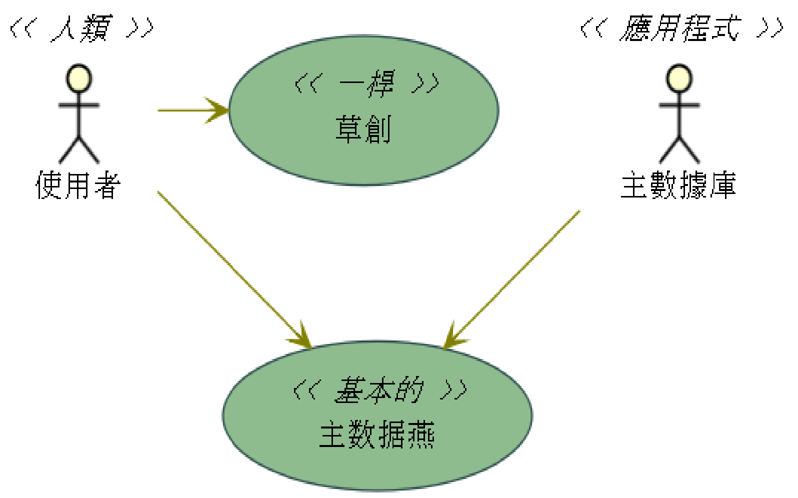
```
<!-- Put images in c:/images directory -->
<target name="main">
  <plantuml dir="./src" charset="UTF-8" />
</target>
```

Depending of your Java installation, the following charset should be available: `ISO-8859-1`, `UTF-8`, `UTF-16BE`, `UTF-16LE`, `UTF-16`.

## Font Issues

When using East Asian Fonts, you may have some issues, because Graphviz default font may *not* contains some characters.
So you may have to force the usage of a system font that contains thoses characters, by adding the following lines in your diagram descriptions.

```
skinparam usecaseActorFontName MS Mincho
skinparam usecaseFontName MS Mincho
```



## Content

1- Sequence Diagram....2
Basic examples....2
Declaring participant....3
Message to Self....4
Message sequence numbering....5
Title....8
Splitting diagrams....9
Grouping message....10
Notes on messages....11
Some other notes....12
Formatting using HTML....13
Lifeline Activation and Destruction....14
Stereotypes and Spots....16
More information on titles....17
Removing Footer....19
Skin....20
2- Use Case Diagram....21
Usecases....21
Actors....21
Basic example....22
Extension....23
Using notes....24
Stereotypes....25
Title the diagram....26
Left to right direction....27
A complete example....28
3- Class Diagram....29
Relations between classes....29
Label on relations....30
Adding methods....31
Notes and stereotypes....32
More on notes....33
Chained relations....34
Abstract class and interface....35
Specific Spot....36
Using packages....37
Title the diagram....39
Association classes....40
Splitting large files....42
4- Activity Diagram....43
Simple Activity....43
Arrows....44
Branches....45
Synchronization....46

Long activity description....47
Notes....48
Partition....49
Title the diagram....50
5- Component Diagram....51
Components....51
Interfaces....51
Basic example....52
Using notes....53
Title the diagram....54
6- State Diagram....55
Simple State....55
Composite State....56
Long name....57
Concurrent State....58
Note....59
More in notes....60
7- Common commands....61
Footer and header....61
Rotation....62
8- Changing fonts and colors....63
Usage....63
Colors....64
Font color, name and size....66
9- Preprocessing....68
Including files....68
Constant definition....69
Conditions....70
10- Internationalization....71
Charset....72
Font Issues....72