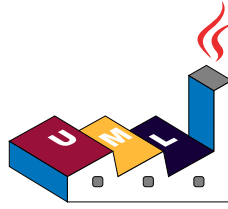


UML-Diagramme mit PlantUML



Sprachreferenz

(Dienstag, 7. März 2017 10:02)

PlantUML ist ein Open Source Projekt, welches das Erstellen von UML-Digrammen ermöglicht. Es werden die folgenden Typen von UML-Diagrammen unterstützt:

- Sequenzdiagramm,
- Anwendungsfalldiagramm,
- Klassendiagramm,
- Aktivitätsdiagramm,
- Komponentendiagramm,
- Zustandsdiagramm,
- Objektdiagramm

Diagramme werden in einer einfachen und intuitiven Sprache in textueller Notation beschrieben.

1 Sequenz-Diagramm

1.1 Grundlagen

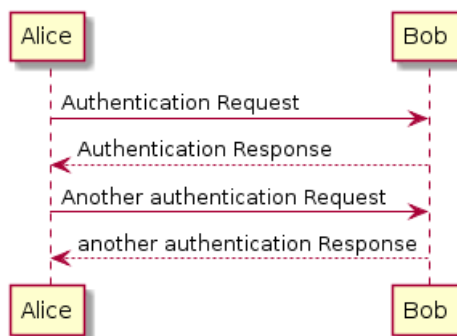
Die Zeichenfolge ">" wird verwendet, um eine Nachricht zwischen zwei Teilnehmern zu zeichnen. Teilnehmer müssen nicht explizit deklariert werden.

Um eine gepunktete Linie zu zeichnen, verwende "-->".

Es ist auch möglich <- und <-- zu verwenden. Dieses ändert nicht die Zeichnung, kann aber die Lesbarkeit erhöhen. Beachte: Das gilt nur für Sequenzdiagramme. In anderen Diagrammen können andere Regeln gelten.

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

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



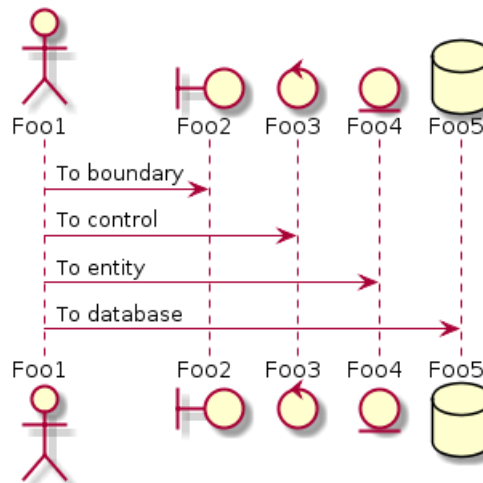
1.2 Deklaration eines Teilnehmers

Mit dem Schlüsselwort `participant` lässt sich die Reihenfolge von Teilnehmern ändern.

Sie können auch folgende andere Schlüsselwörter anstelle von `participant` verwenden:

- `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
```



Teilnehmer können mittels `as` umbenannt werden.

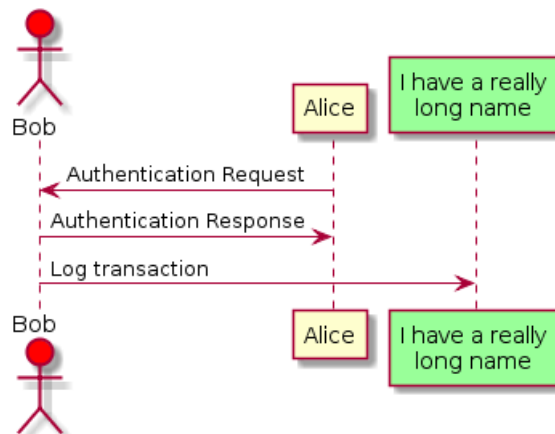
Die Hintergrundfarbe von Teilnehmern oder Akteuren kann mithilfe von HTML Farbcodes oder Farbbezeichnungen gesetzt werden.

```

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

```

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



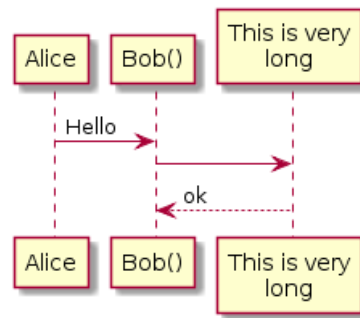
1.3 Verwendung von nicht-alphanumerischen Zeichen

Soll die Bezeichnung eines Teilnehmers nicht-alphanumerische Zeichen enthalten (z.B. Klammern oder Zeilenumbrüche), müssen Anführungszeichen bei der Definition verwendet werden. Das Schlüsselwort `as` kann verwendet werden, um einen Alias für einen Teilnehmer zu definieren.

```

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





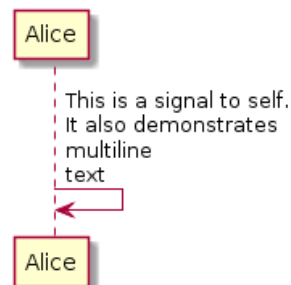
1.4 Nachrichten an sich selbst

Ein Teilnehmer kann auch eine Nachricht an sich selbst schicken.

Die Nachricht kann mehrere Zeilen umfassen. Mit \n können Zeilenumbrüche gemacht werden.

```

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



1.5 Ändern der Pfeilart

Die Art eines Pfeils kann auf verschiedene Weise geändert werden:

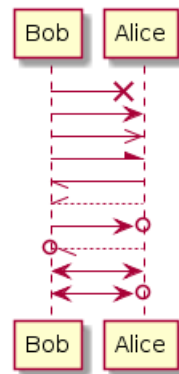
- für eine verloren gegangene Nachricht hängen Sie am Ende des Pfeils ein x an.
- Verwendung von \ oder / anstelle von < oder >, um nur den unteren oder oberen Teil des Pfeils zu zeichnen.
- Verwendung von >> oder //, um eine nicht ausgefüllte Pfeilspitze zu zeichnen.
- Verwendung von -- anstelle von -, um eine gestrichelte Linie zu zeichnen.
- Fügen Sie ein "o" am Ende des Pfeils an
- benutzen Sie zweiseitige Pfeile

```

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

Bob ->o Alice
Bob o\-- Alice

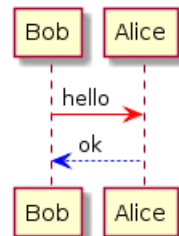
Bob <-> Alice
Bob <->o Alice
@enduml
  
```



1.6 Ändern der Pfeil Farbe

Sie können die Farbe einzelner Pfeile mit folgender Notation ändern:

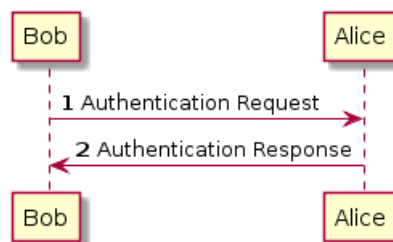
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.7 Nummerierung der Nachrichtenreihenfolge

Das Schlüsselwort `autonumber` kann verwendet werden, um Nachrichten automatisch zu nummerieren.

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



Sie können die Anfangsnummer 'start' mit `autonumber 'start'` festlegen und Sie können diese Nummer mit `autonumber 'start' 'increment'` um 'increment' hochzählen.

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

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

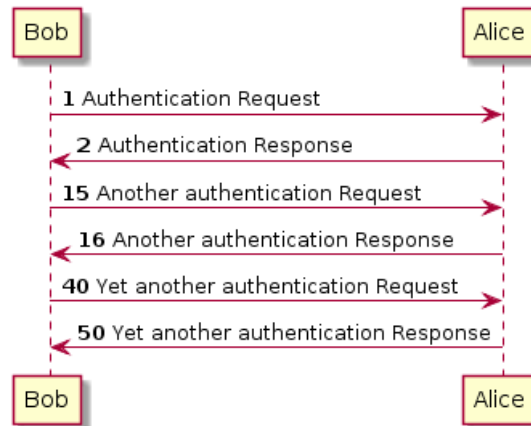
autonumber 40 10
Bob -> Alice : Yet another authentication Request
```



```

Bob <- Alice : Yet another authentication Response
@enduml

```



Man kann das Format der Aufzählung festlegen, indem man ein doppeltes Anführungszeichen verwendet.

Dazu wird die Java Klasse `DecimalFormat` verwendet ('0' bedeutet Ziffer, '#' bedeutet Ziffer und Null wenn die Ziffer fehlt).

Außerdem können HTML Tags für die Formatierung verwendet werden.

```

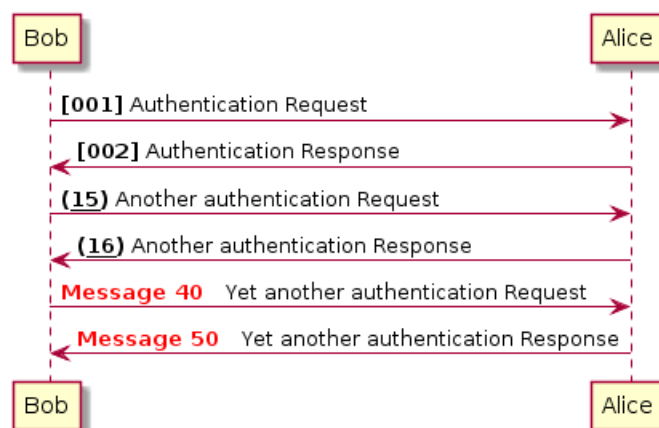
@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

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

```



Mit den Schlüsselwörtern `autonumber stop` bzw. `autonumber resume 'increment' 'format'` wird die Aufzählung pausiert bzw. wieder fortgesetzt.

```

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

autonumber stop

```

```

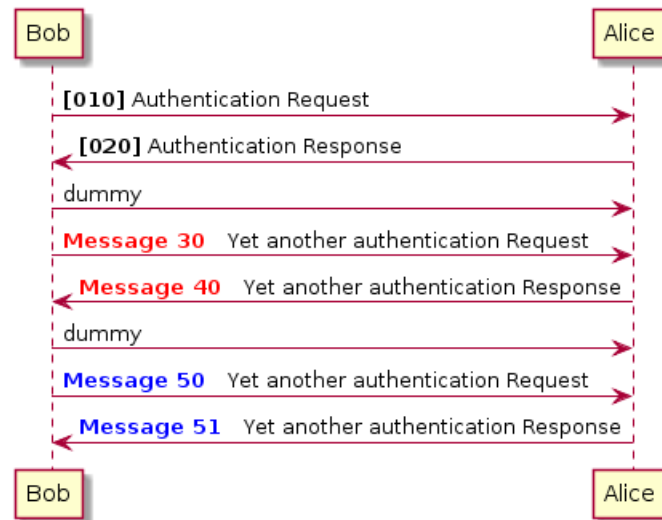
Bob -> Alice : dummy

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 Aufteilung von Diagrammen

Das **newpage** Schlüsselwort wird verwendet, um ein Diagramm in mehrere Bilder aufzuteilen.

Man kann den Titel der neuen Seite direkt hinter dem **newpage** Schlüsselwort angeben.

Das ist sehr praktisch, um große Diagramme auf mehreren Seiten auszudrucken.

```

@startuml

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

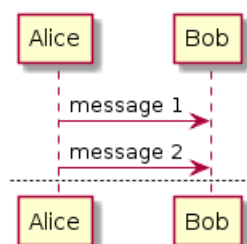
newpage

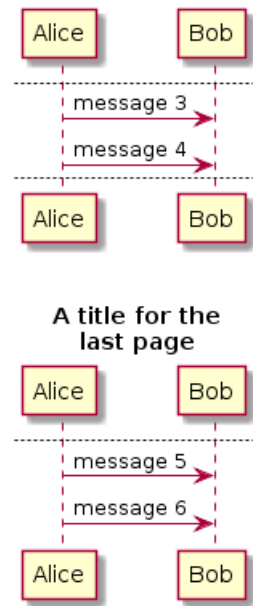
Alice -> Bob : message 3
Alice -> Bob : message 4

newpage A title for the\nlast page

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

```





1.9 Gruppierung von Nachrichten

Nachrichten können mit den folgenden Schlüsselwörtern gruppiert werden:

- alt/else
- opt
- loop
- par
- break
- critical
- group, gefolgt von einem anzuzeigenden Text

Es ist möglich einen Text anzugeben, der im Titel angezeigt werden soll.

Das **end** Schlüsselwort wird verwendet, um die Gruppe zu schließen.

Weiterhin ist es möglich, mehrere Gruppen ineinander zu schachteln.

```

@startuml
Alice -> Bob: Authentication Request

alt successful case

Bob -> Alice: Authentication Accepted

else some kind of failure

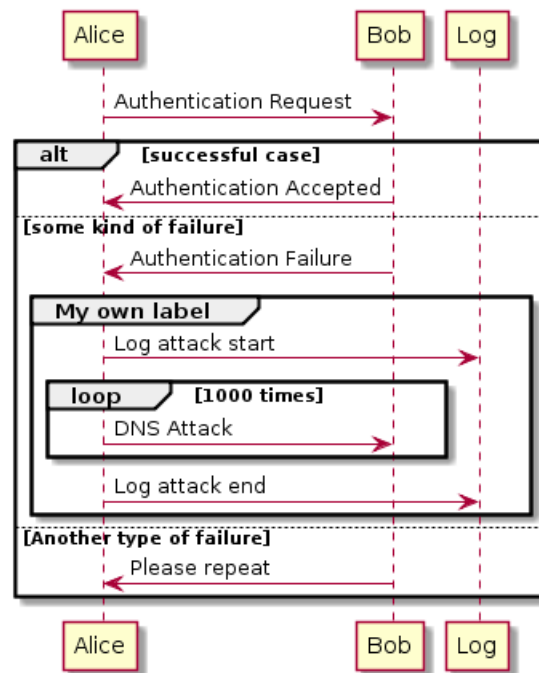
Bob -> Alice: Authentication Failure
group My own label
Alice -> Log : Log attack start
loop 1000 times
Alice -> Bob: DNS Attack
end
Alice -> Log : Log attack end
end

else Another type of failure

Bob -> Alice: Please repeat

end
@enduml
  
```





1.10 Notizen

Notizen zu einer Nachricht werden mit dem Schlüsselwort `note left` (links) oder `note right` (rechts) gleich nach der Nachricht eingeleitet.

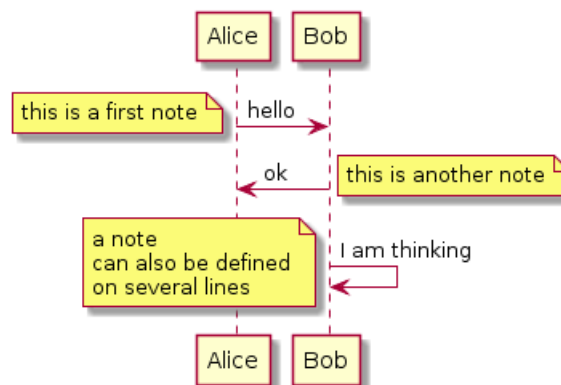
Soll die Notiz mehrere Zeilen umfassen, muss das Schlüsselwort `end note` am Ende der Notiz verwendet werden..

```

@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 Weitere Möglichkeiten für Notizen

Weiterhin ist es Möglich, die Notizen rechts, links, oben oder unten an dem Teilnehmer zu platzieren:

Es ist möglich, die Notizen durch die Änderung der Hintergrundfarbe hervorzuheben.



Außerdem kann man durch die Verwendung des `end note` Schlüsselwortes mehrzeilige Notizen erzeugen.

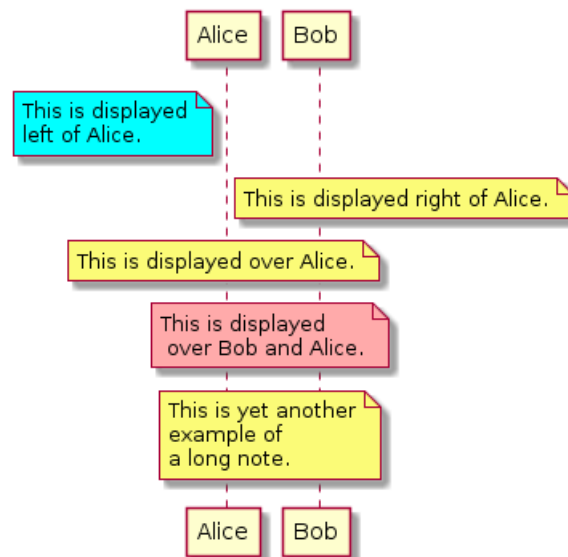
```
@startuml
participant Alice
participant Bob
note left of Alice #aqua
This is displayed
left of Alice.
end note

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.

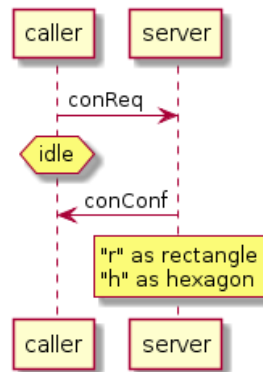
note over Bob, Alice
This is yet another
example of
a long note.
end note
@enduml
```



1.12 Ändern der Form von Notizen

Mit den Schlüsselwörtern `hnote` und `rnote` kann man die Form der Notiz ändern.

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



1.13 Creole und HTML

Es ist auch möglich, den Text mit Creole-Markup zu formatieren.

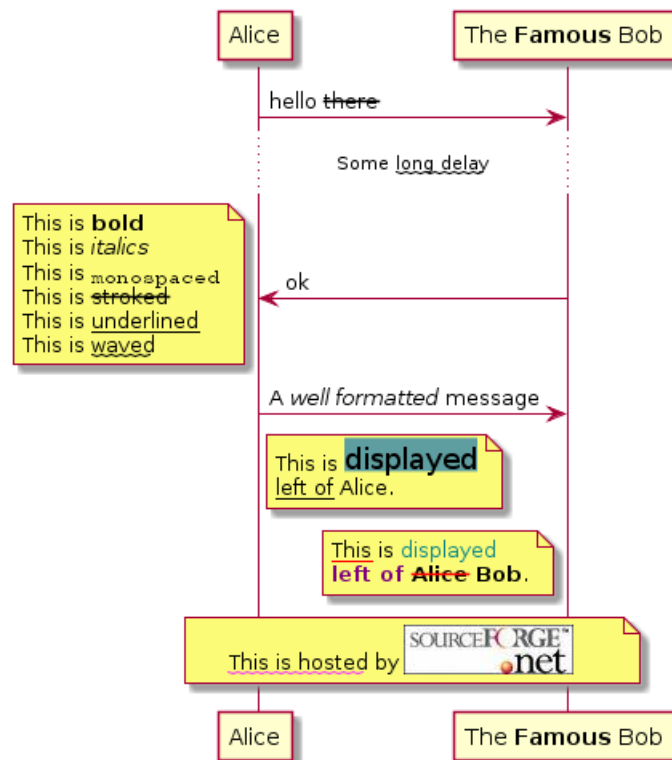
```

@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 Diagramme aufteilen

Bei Bedarf kann ein Diagramm mit dem "==" Separator in logische Schritte unterteilt werden.

```
@startuml
```

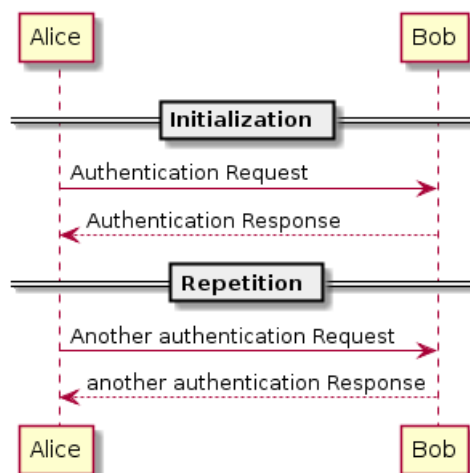
```
== Initialization ==
```

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

```
== Repetition ==
```

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

```
@enduml
```



1.15 Referenz

Die Referenz kann in einem Diagramm mit Hilfe des Schlüsselwortes **ref over** verwendet werden.



```

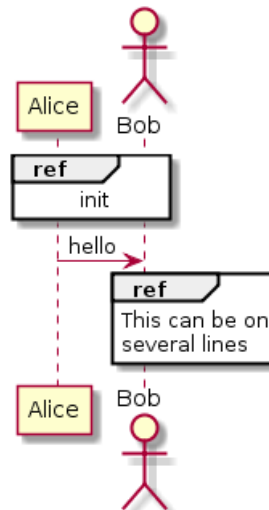
@startuml
participant Alice
actor Bob

ref over Alice, Bob : init

Alice -> Bob : hello

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

```



1.16 Verzögerungen

Mit ... kann man eine Verzögerung in dem Diagramm anzeigen. In dieser Verzögerung kann außerdem eine Nachricht angezeigt werden.

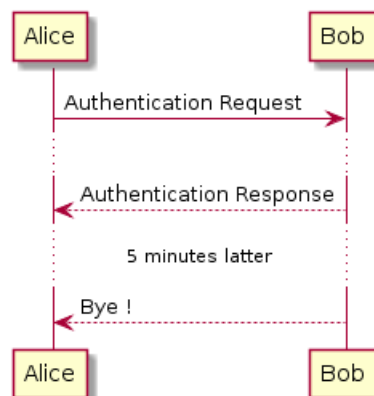
```

@startuml

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

@enduml

```



1.17 Abstände

Mit ||| kann ein Abstand zwischen zwei Nachrichten eingefügt werden.



Außerdem ist es möglich, die Größe des Abstandes in Pixeln festzulegen.

```
@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 Aktivierung und Deaktivierung der Lebenslinie

Mit den Befehlen **activate** und **deactivate** können die Teilnehmer aktiviert und deaktiviert werden.

Wenn ein Teilnehmer aktiviert wurde, dann erscheint seine Lebenslinie.

Die Befehle **activate** und **deactivate** wirken nach der vorhergehenden Nachricht.

Der Befehl **destroy** beendet die Lebenslinie eines Teilnehmers.

```
@startuml
```

```
participant User
```

```
User -> A: DoWork
activate A
```

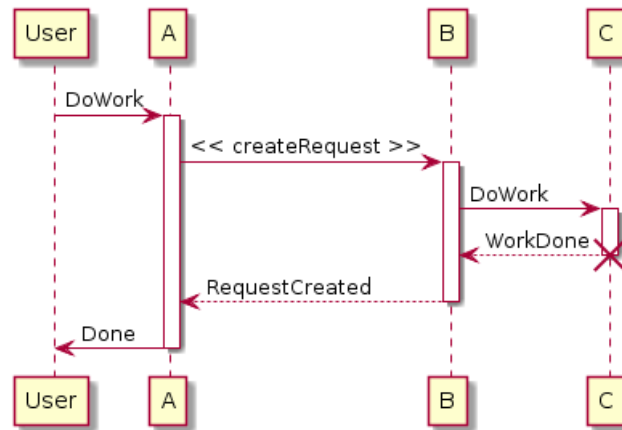
```
A -> B: << createRequest >>
activate B
```

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

```
B --> A: RequestCreated
deactivate B
```

```
A -> User: Done
deactivate A
```

```
@enduml
```



Es ist auch möglich, geschachtelte Lebenslinien zu erzeugen. Außerdem kann man einer Lebenslinie eine Farbe zuweisen.

```

@startuml
participant User

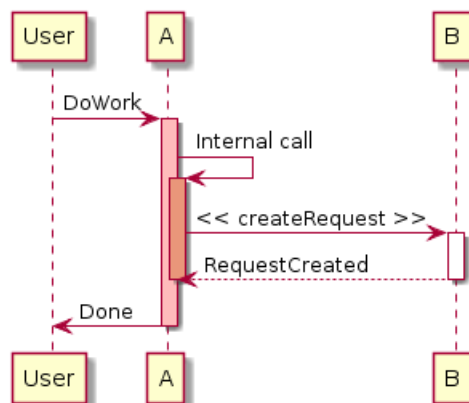
User -> A: DoWork
activate A #FFBBBB

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 Erstellung von Teilnehmern

Das **create** Schlüsselwort kann kurz vor dem ersten Empfang einer Nachricht verwendet werden, um anzuzeigen, dass die Nachricht für die *Erstellung* des neuen Objektes verantwortlich ist.

```

@startuml
Bob -> Alice : hello

create Other
Alice -> Other : new

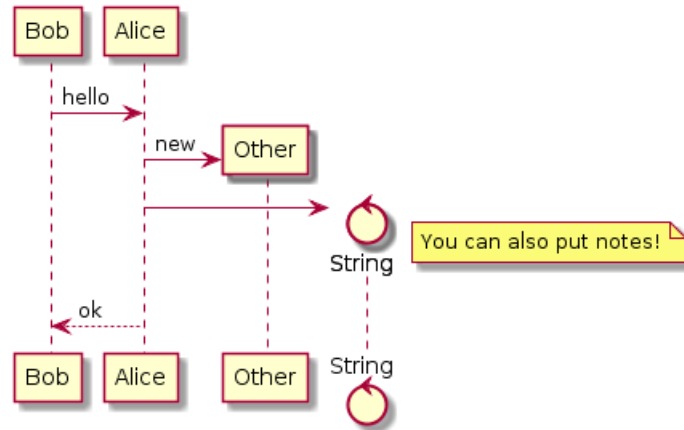
create control String
Alice -> String
  
```



```

note right : You can also put notes!
Alice --> Bob : ok
@enduml

```



1.20 Eingehende und ausgehende Nachrichten

Um sich nur auf ein Teil des Diagramms zu konzentrieren, kann man eingehende und ausgehende Pfeile verwenden.

Mit eckigen Klammern kann man die linke "[" oder die rechte "]" Seite des Pfeils festlegen.

```

@startuml
[-> A: DoWork

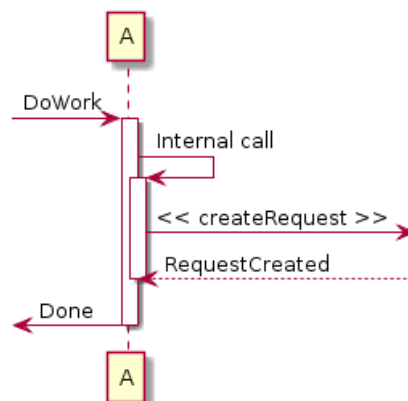
activate A

A -> A: Internal call
activate A

A ->] : << createRequest >>

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

```



Die folgende Syntax ist auch möglich:

```

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

```



```

[x-> Bob

[<- Bob
[x<- Bob

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

Bob <-]
Bob x<-]
@enduml

```



1.21 Stereotypen

Man kann den Objekten Stereotypen zuweisen, indem man den Stereotyp mit zwei spitzen öffnenden "<<" und schließenden Klammern ">>" umschließt.

Innerhalb des Stereotypen ist es möglich einen hervorgehobenen Buchstaben hinzuzufügen, der in einem farbigen Kreis dargestellt wird. Dazu verwendet man die folgende Syntax: "(X,color)".

```

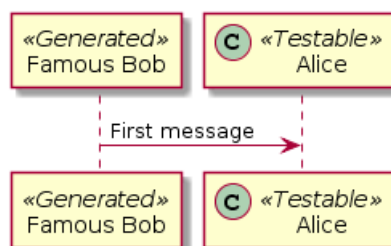
@startuml

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

Bob->>Alice: First message

@enduml

```



Standardgemäß werden *französisches Anführungszeichen* verwendet, um den Stereotyp zu kennzeichnen. Dieses Verhalten kann über den `skinparam guillemet` Befehl beeinflusst werden.

```

@startuml

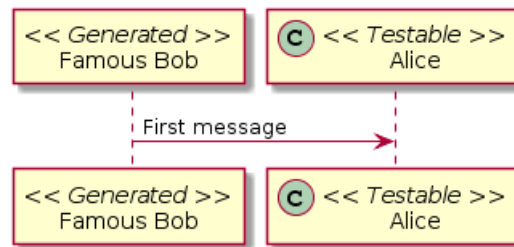
skinparam guillemet false
participant "Famous Bob" as Bob << Generated >>
participant Alice << (C,#ADD1B2) Testable >>

Bob->>Alice: First message

@enduml

```





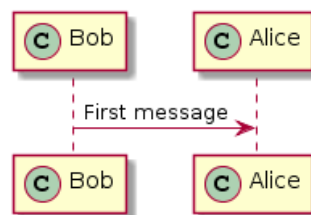
```

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

Bob->>Alice: First message

@enduml

```



1.22 Mehr Information zu Überschriften

Mit Creole-Markup ist es möglich, die Überschrift des Diagramms zu formatieren.

```

@startuml
title __Simple__ **communication** example

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

@enduml

```

Simple communication example



Eine neue Zeile kann mit \n in die Überschrift der Bezeichnung eingetragen werden.

```

@startuml
title __Simple__ communication example\nnon several lines

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

@enduml

```



Simple communication example on several lines



Mehrzeilige Überschriften können mit den `title` und `end title` Schlüsselwörtern erstellt werden.

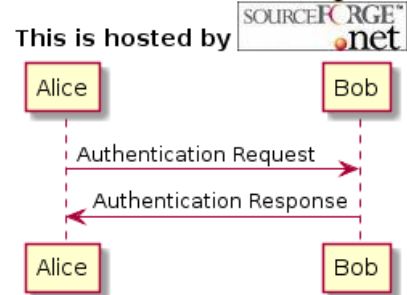
```
@startuml

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 Anpassungen bei den Teilnehmern

Es ist möglich Boxen um Teilnehmer zu zeichnen, indem man die Befehle `box` und `end box` benutzt.

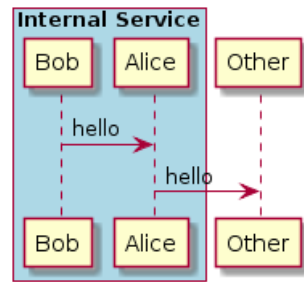
Man kann optional noch einen Titel oder eine Hintergrundfarbe nach dem `box` Schlüsselwort hinzufügen.

```
@startuml

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

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

@enduml
```



1.24 Fußzeile entfernen

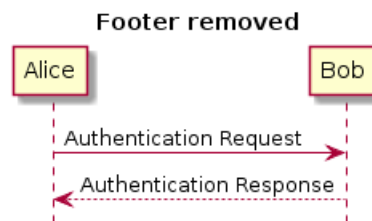
Die Fußzeile eines Diagramms kann mit dem `hide footbox` Schlüsselwort entfernt werden.

```

@startuml
hide footbox
title Footer removed

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

@enduml
  
```



1.25 Der Skinparam Befehl

Mit Hilfe des Skinparam-Befehls können Farben und Zeichenformatierungen angepasst werden.

Der Befehl kann verwendet werden:

- in der Diagramm-Definition (wie alle Befehle),
- durch Definition in einer Include-Datei,
- durch Definition in einer Konfigurationsdatei, die durch die Kommandozeile oder einen ANT-Task übergeben wird.

Es ist auch möglich, weitere Parameter zu editieren. Dies ist in den folgenden Beispielen dargestellt:

```

@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
  
```



```

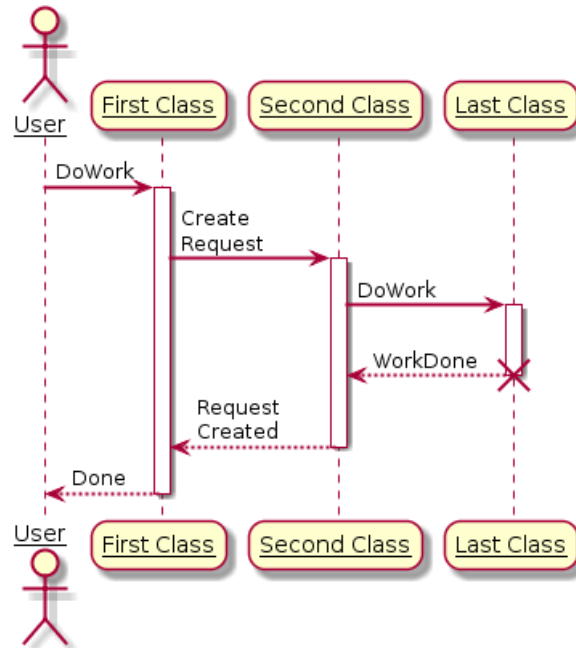
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



```

@startuml
skinparam backgroundColor #EEEEBD
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

C --> B: WorkDone
deactivate C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```

```

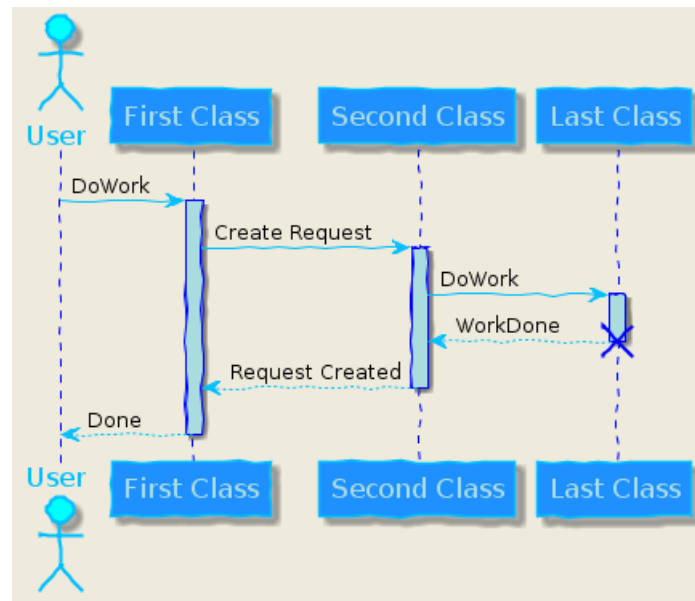
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



1.26 Changing padding

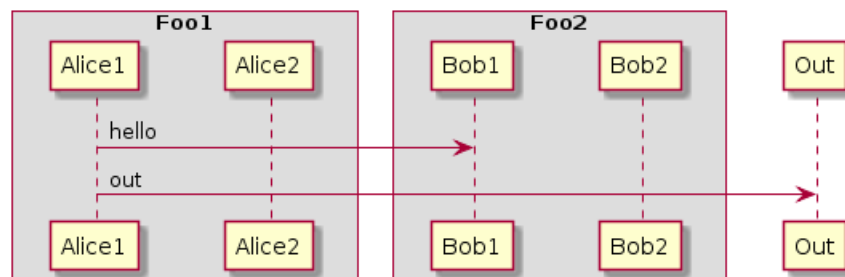
It is possible to tune some padding settings.

```

@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

```



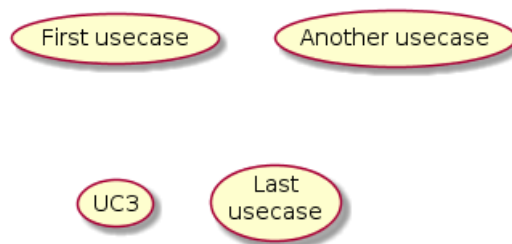
2 Anwendungsfall-Diagramm

2.1 Anwendungsfälle

Anwendungsfälle sind von zwei Klammern eingeschlossen (da zwei Klammern wie ein Oval aussehen).

Alternativ kann man das **usecase** Schlüsselwort verwenden, um einen Anwendungsfall zu definieren. Außerdem ist es möglich einen Alias mit dem **as** Schlüsselwort zu definieren. Dieser Alias wird dann verwendet wenn die Beziehungen festgelegt werden.

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



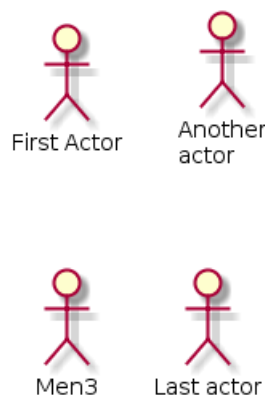
2.2 Akteure

Die Namen von Akteuren werden von zwei Doppelpunkten umschlossen.

Mann kann aber auch das **actor** Schlüsselwort verwenden um einen Akteur zu definieren. Außerdem ist es möglich, mit dem **as** Schlüsselwort einen Alias festzulegen. Dieser Alias wird dann später verwendet, wenn die Beziehungen festgelegt werden.

Wie wir sehen werden, ist die Definition eines Akteur nicht zwingend notwendig.

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



2.3 Beschreibung der Anwendungsfälle

Falls sich eine Beschreibung über mehrere Zeilen erstreckt, kann diese mit Anführungsstrichen eingeschlossen werden.

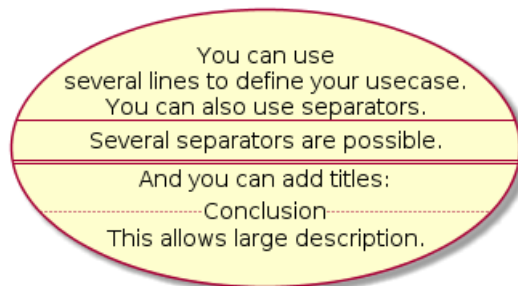


Außerdem kann man die folgenden Separatoren verwenden: -- .. == __. Außerdem kann man Überschriften innerhalb der Separatoren verwenden.

```
@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 Einfaches Beispiel

Um Akteure und Anwendungsfälle miteinander zu verbinden wird der Pfeil "-->" verwendet

Je mehr Bindestriche "-" der Pfeil enthält, desto länger wird der Pfeil. Mit einem Doppelpunkt ":" kann dem Pfeil eine Beschreibung hinzugefügt werden.

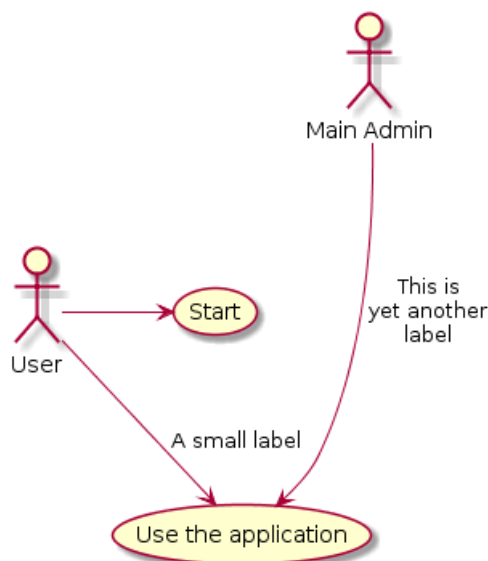
In diesem Beispiel kann man sehen, wie ein vorher nicht deklarierter *User* automatisch als Akteur deklariert wird.

```
@startuml
```

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

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

@enduml
```



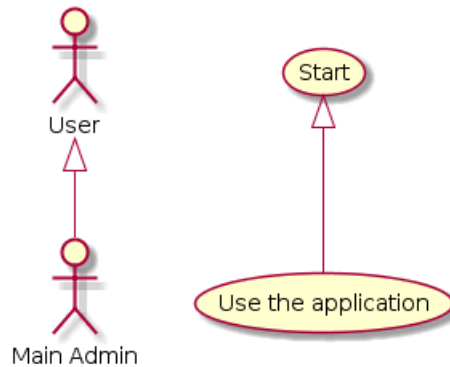
2.5 Erweiterungen / Generalisierungen

Wenn ein Akteur oder Anwendungsfall einen anderen erweitert, dann kann dies mit dem Symbol

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

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

@enduml
```



2.6 Verwenden von Notizen

Mit den `note left of`, `note right of`, `note top of`, `note bottom of` Schlüsselwörtern kann man die Position der Notiz relativ zum Objekt festlegen.

Eine Notiz kann aber auch nur mit dem `note` Schlüsselwort erstellt werden und dann mit dem `..` Symbol den Objekten zugeordnet werden.

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

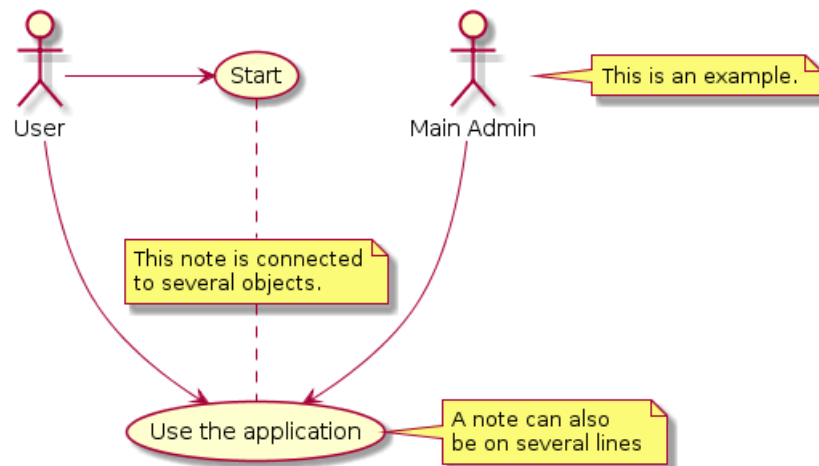
User -> (Start)
User --> (Use)

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 Stereotypen

Stereotypen können während der Erstellung der Akteure und der Anwendungsfälle mit den "<<" und ">>" Symbolen hinzugefügt werden.

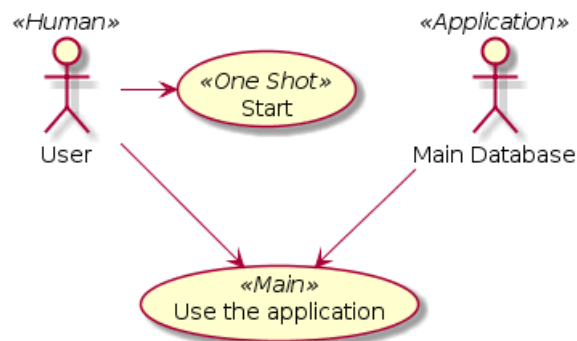
```

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

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

MySQL --> (Use)

@enduml
  
```

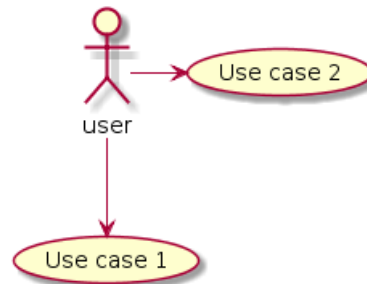


2.8 Ändern der Pfeilrichtungen

Normalerweise haben die Verbindungen zwischen den Klassen zwei Striche -- und werden senkrecht gezeichnet. Es ist aber möglich waagerechte Verbindungen zu erstellen in dem man einen einzelnen Strich (oder Punkt) eingibt:

```

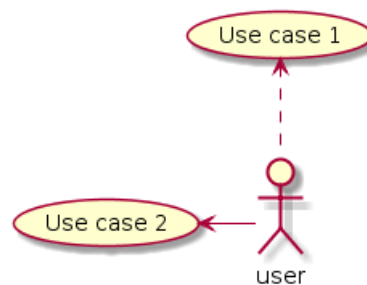
@startuml
:user: --> (Use case 1)
:user: -> (Use case 2)
@enduml
  
```



Sie können auch die Richtung der Verlinkung umkehren:

```

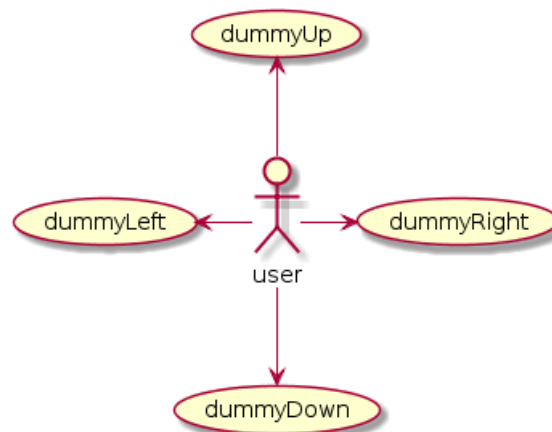
@startuml
(Use case 1) <.. :user:
(Use case 2) <- :user:
@enduml
  
```



Die Richtung der Pfeile kann man durch das hinzufügen der **left**, **right**, **up** oder **down** Schlüsselworte im Pfeil bestimmen:

```

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



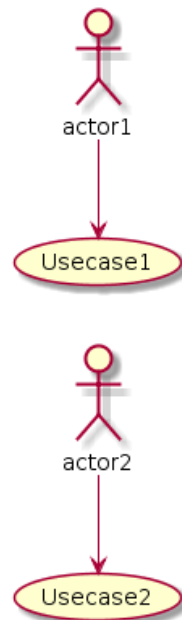
Man kann die Pfeile verkürzen, wenn man nur den ersten Buchstaben für die Richtung verwendet (zum Beispiel, **-d-** anstelle von **-down-**) oder man nimmt die ersten beiden Buchstaben (**-do-**).

Diese Möglichkeit sollte aber nicht missbraucht werden: *Graph Viz* liefert normalerweise recht gute Ergebnisse, ohne das manuell eingegriffen werden muss.

2.9 Aufteilen von Diagrammen auf mehrere Seiten

Mit dem Befehl `newpage` kann das Diagramm auf mehrere Seiten oder Bilder verteilt werden.

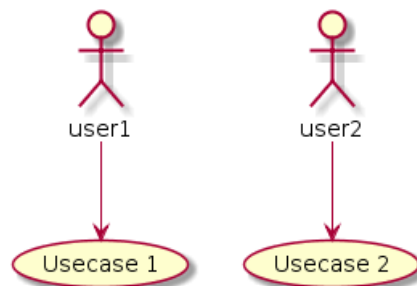
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



2.10 Verändern der Richtung in der die Objekte angeordnet werden

Das voreingestellte Verhalten bei der Erstellung des Diagramms ist von oben nach unten.

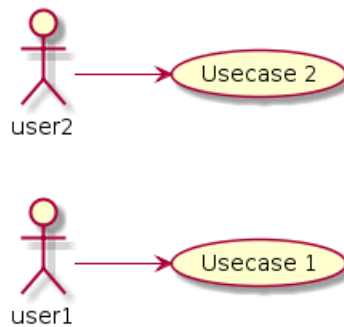
```
@startuml
'default
top to bottom direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)
@enduml
```



Dies lässt sich aber durch die Verwendung des `left to right direction` Befehls verändern. Oft ist das Ergebnis mit dieser Einstellung besser.

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





2.11 Der Skinparam-Befehl

Mit dem `skinparam` Befehl kann die Farbe und die Schriftart der Zeichnung verändert werden.

Sie können den Befehl auf die folgenden Arten verwenden:

- Wie alle andere Befehle In einer Diagrammdefinition,
- in einer Include-Datei,
- In einer Konfigurationsdatei, die durch die Kommandozeile oder den ANT-Task übergeben wird.

Man kann bestimmte Farben und Schriften für Klassen von Akteuren und Anwendungsfälle festlegen.

```

@startuml
skinparam handwritten true

skinparam usecase {
  BackgroundColor DarkSeaGreen
  BorderColor DarkSlateGray
}

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

ArrowColor Olive
ActorBorderColor black
ActorFontName Courier

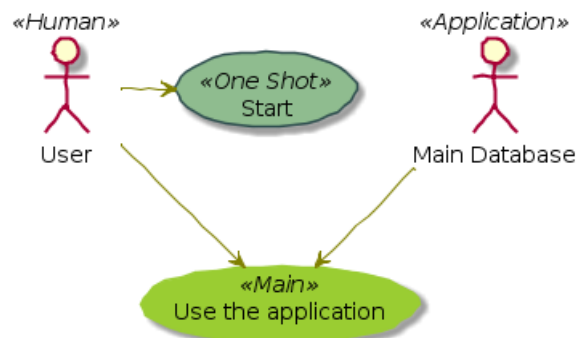
ActorBackgroundColor<< Human >> Gold
}

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

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

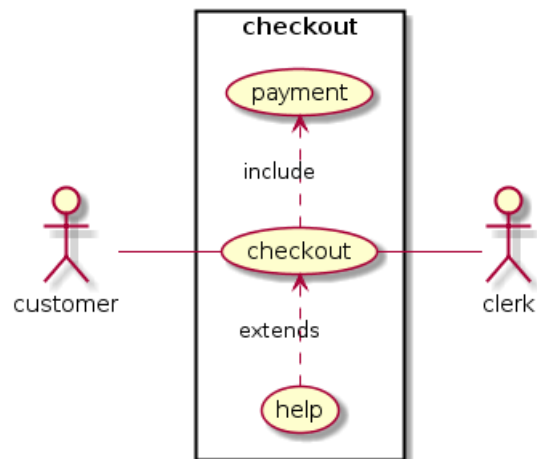
MySql --> (Use)

@enduml
  
```



2.12 Vollständiges Beispiel




```
@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 Klassendiagramm

3.1 Beziehungen zwischen Klassen

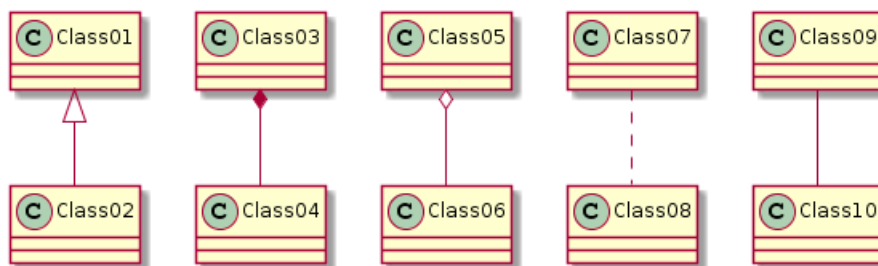
Beziehungen zwischen Klassen werden mit den folgenden Symbolen gekennzeichnet:

Generalisierung	< --	
Komposition	*--	
Aggregation	o--	

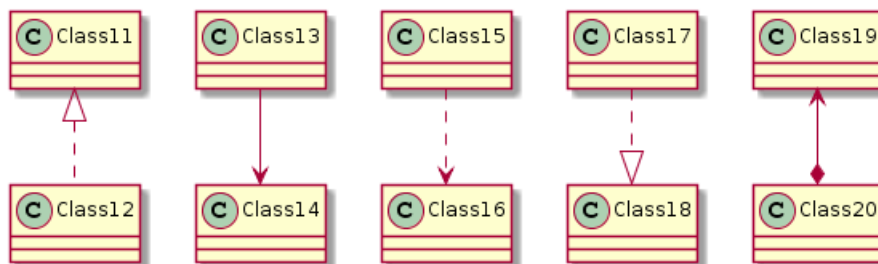
Es ist möglich "--" durch ".." zu ersetzen, um eine gepunktete Linie zu erhalten.

Wenn man diese Regeln kennt, ist es möglich, die folgenden Zeichnungen zu zeichnen:

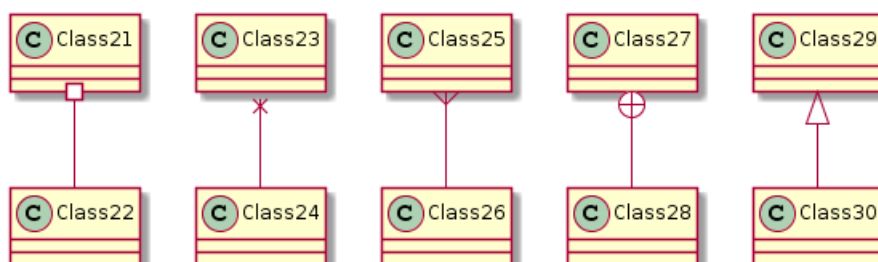
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```



```
@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
```

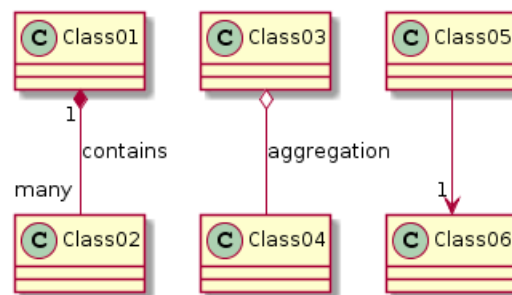


3.2 Beschriften von Beziehungen

Beziehungen können beschriftet werden, durch das Anhängen eines Doppelpunktes ":" gefolgt von dem Beschriftungstext.

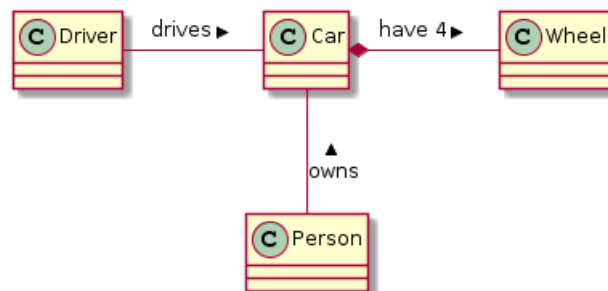
Um Kardinalität anzuzeigen, verwendet man doppelte Anführungszeichen "" auf jeder Seite der Beziehung.

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



Um zu zeigen, in welche Richtung die Beziehung wirkt, können an die Beschriftung zusätzliche Pfeilspitzen angehängt werden, indem man vor die Beschriftung < oder nach der Beschriftung > verwendet.

```
@startuml
class Car
Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns
@enduml
```



3.3 Methoden hinzufügen

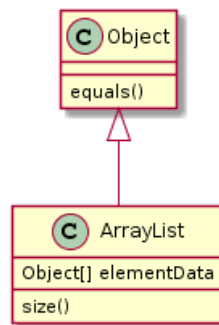
Um Feldern und Methoden zu einer Klasse hinzuzufügen, wird der Doppelpunkt ":" gefolgt von dem Namen des Feldes oder der Methode verwendet.

Das System erkennt anhand der Klammern, ob es sich um eine Methode oder um ein Feld handelt.

```
@startuml
Object <|-- ArrayList

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

@enduml
```



Es ist möglich in Klammern, Feldern und Methoden zu gruppieren

Die Syntax ist sehr flexibel bezüglich der Reihenfolge der Typen und Namen.

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

class Flight {
flightNumber : Integer
departureTime : Date
}

@enduml
```

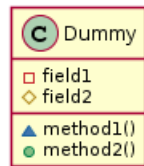


3.4 Sichtbarkeit festlegen

Beim Definieren von Methoden und Feldern kann die Sichtbarkeit mit einem der folgenden Zeichen festgelegt werden:

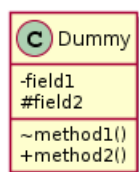
-	□	private
#	◇	protected
~	△	package private
+	○	public

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



Mit dem skinparam classAttributeIconSize 0 Befehl kann dieses Verhalten ausgeschaltet werden :

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

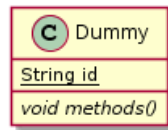


3.5 Abstract und Static

Sie können statische oder abstrakte Methoden und statische Attribute durch benutzen des **static** oder **abstract** Modifikators definieren.

Diese Modifikatoren können am Anfang oder am Ende der Zeile benutzt werden. Es kann auch **classifier** statt **static** benutzt werden.

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



3.6 Der Klassenrumpf für Fortgeschrittene

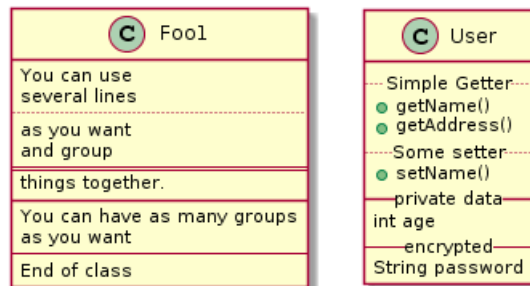
Standardmäßig werden die Methoden und Felder im Klassenrumpf automatisch von PlantUML gruppiert. Mit Hilfe von Trennzeichen können Felder und Methoden aber auch selber geordnet werden. Folgende Trennzeichen sind möglich: `--` (einfache durchgezogene Linie), `..` (einfache unterbrochene Linie), `==` (doppelte durchgezogene Linie), `__` (dicke durchgezogene Linie).

Es können auch Titel innerhalb des Trennzeichen angegeben werden:

```
@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 Notizen und Stereotypen

Stereotypen werden mit dem Schlüsselwort `class` oder mit den Symbolen "`<<`" (doppelte spitze Klammer links) und "`>>`" (doppelte spitze Klammer rechts) definiert. Zwischen den Klammern wird der Name des Stereotyps angegeben.

Mit den `note left of`, `note right of`, `note top of`, `note bottom of` Schlüsselwörtern kann man Notizen und ihre Position festlegen.

Eine Notiz zur zuletzt definierten Klasse wird mit den Schlüsselwörtern `note left`, `note right`, `note top`, `note bottom` hinzugefügt.

Eine Notiz kann aber auch nur mit dem `note` Schlüsselwort erstellt werden und dann mit dem `..` Symbol den Klassen zugeordnet werden.

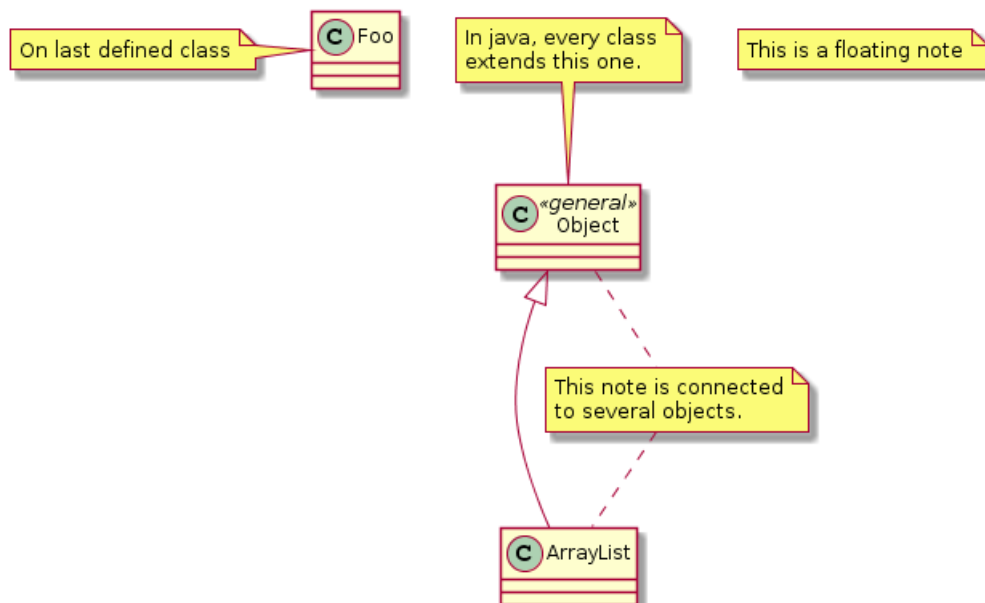
```
@startuml
class Object << general >>
Object <|--- ArrayList

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 Mehr zu Notizen

Es ist auch möglich einige HTML Tags wie:

- ``
- `<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

Es ist auch möglich eine Notiz über mehrere Zeilen zu erstellen.

Eine Notiz bezogen auf die letzte definierte Klasse kann mit `note left`, `note right`, `note top` oder `note bottom` erstellt werden.

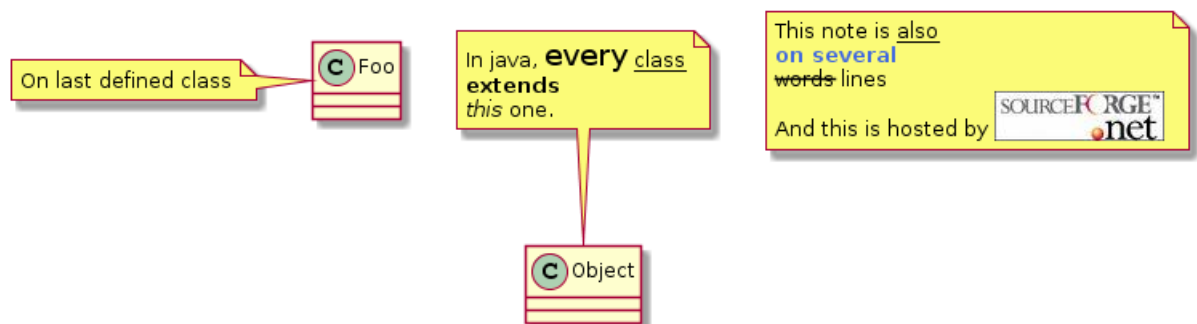
```
@startuml

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
```



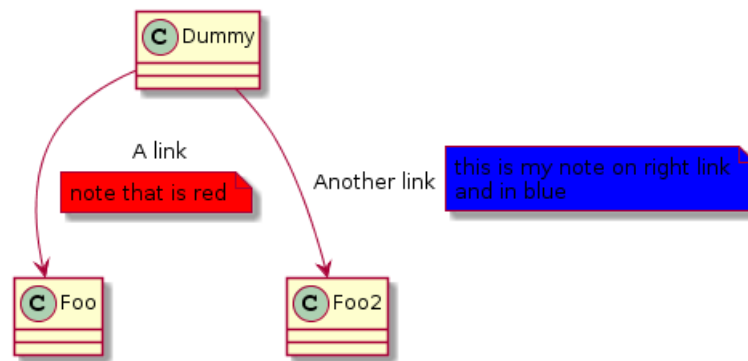
3.9 Notizen zu Beziehungen

Eine Notiz zu einer Beziehung kann direkt nach der Beziehungsdefinition erfolgen: `note on link`.

Zur relativen Positionierung der Notiz können die Schlüsselwörter `note left on link`, `note right on link`, `note top on link`, `note bottom on link` verwendet werden.

```
@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 Abstrakte Klassen und Interfaces

Eine abstrakte Klasse lässt sich über das "abstract" oder das "abstract class" Schlüsselwort definieren. Die Klasse wird dann kursiv gedruckt.

Man kann auch die `interface` und `enum` Schlüsselwörter verwenden.

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

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

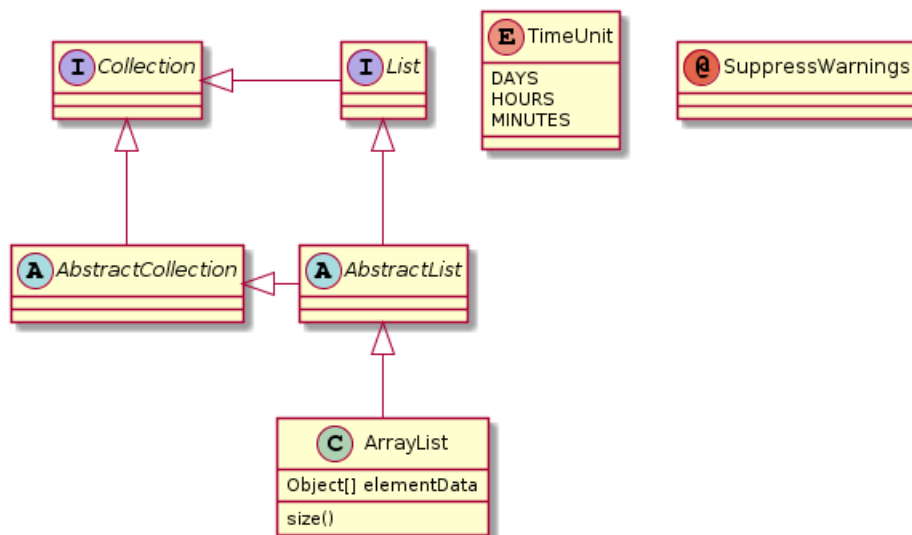
Collection <|-- List
AbstractCollection <|-- AbstractList
AbstractList <|-- ArrayList

class ArrayList {
Object[] elementData
size()
}

enum TimeUnit {
DAYS
HOURS
MINUTES
}

annotation SuppressWarnings

@enduml
```



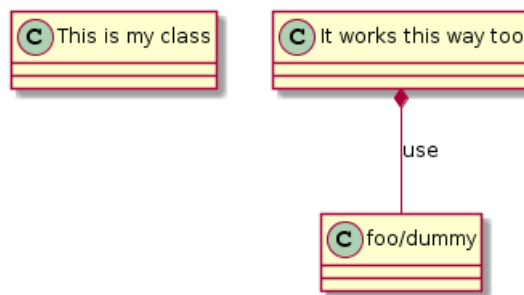
3.11 Verwendung von Sonderzeichen

Wenn sie inn dem Name Ihrer Klasse (oder des Enums, oder der Schnittstelle) Zeichen verwenden wollen, dann gibt es die folgenden Möglichkeiten:

- Verwenden Sie das **as** Schlüsselwort in der Definition
- Schließen Sie den Namen in Hochommas "" ein

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

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



3.12 Verstecken von Attributen, Methoden ...

Die Anzeige einer Klasse kann über das `hide/show` Kommando parametrisiert werden.

Der Basisbefehl ist `hide empty members`. Mit diesem Befehl werden leere Attribute und Methoden ausgeblendet.

Anstelle von `empty members` kann man auch die folgenden Befehle verwenden:

- `empty fields` oder `empty attributes` für leere Felder,
- `empty methods` für leere Methoden,
- `fields` oder `attributes` um Felder auszublenden, auch wenn diese definiert sind,
- `methods` um Methoden auszublenden, auch wenn diese definiert sind,
- `members` um Methoden und Felder auszublenden, auch wenn diese definiert sind,
- `circle` um einen in einen Kreis eingeschlossenen Buchstaben vor dem Klassennamen anzuzeigen,
- `stereotype` um einen Stereotypen anzuzeigen.

Nach dem `hide` oder dem `show` Schlüsselwort kann man auch noch die folgenden Befehle anfügen:

- `class` für alle Klassen,
- `interface` für alle Schnittstellen,
- `enum` für alle Enums,
- `<<foo1>>` für alle Klassen, die mit dem Stereotyp *foo1* ausgezeichnet sind,
- einen namen einer existierenden Klasse.

Es lassen sich mehrere `show/hide` Befehle verketten, um Regeln und ausnahmen festzulegen.

```
@startuml

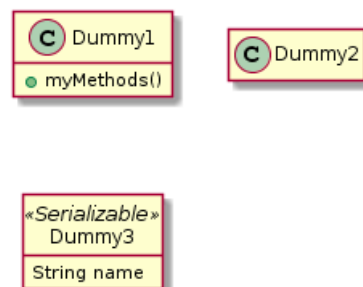
class Dummy1 {
+myMethods()
}

class Dummy2 {
+hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

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

@enduml
```



3.13 Verstecken von Klassen

Mit den `show/hide` Befehlen können Klassen versteckt werden.

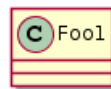
Dies kann hilfreich sein, wenn man eine große !included Datei verwendet und dann einige Klassen nach dem einbinden der Datei verstecken möchte.

```
@startuml
class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```

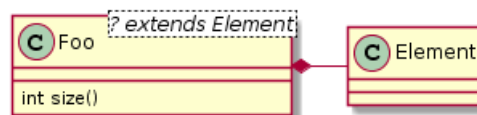


3.14 Verwenden von Generics

Mit spitzen Klammern (< und >) kann die Verwendung von Generics dargestellt werden.

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

@enduml
```



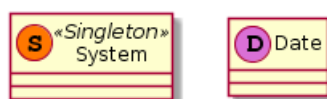
3.15 Besondere Hervorhebungen

Normalerweise werden Klassen, Schnittstellen, Enums und abstrakte Klassen mit einem hervorgehobenen Buchstaben gekennzeichnet (C, I, E or A).

Es ist aber auch möglich eine eigene Hervorhebung zu erstellen wenn man einen Stereotyp definiert. Das wird durch hinzufügen eines einzelnen Buchstabens und einer Farbe so wie im folgenden Beispiel erreicht:

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

@enduml
```



3.16 Pakete

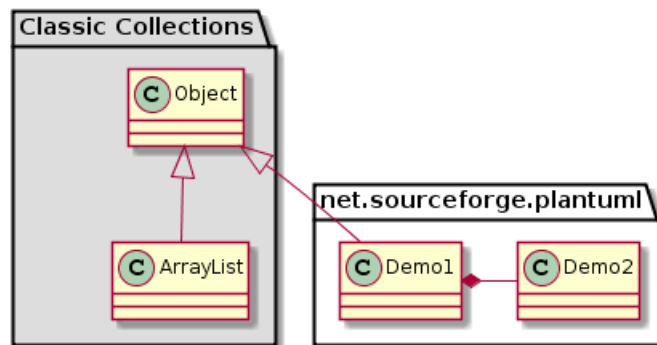
Pakete können über das **package** Schlüsselwort definiert werden. Auf Wunsch kann außerdem die Hintergrundfarbe für das Paket festgelegt werden. Dies kann durch den Farbnamen oder den HTML Code geschehen.

Es ist möglich, Pakete ineinander zu schachteln.

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

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

@enduml
```



3.17 Paketarten

Es stehen verschiedene Arten von Paketen zur Verfügung.

Welches Paket zur Verwendung kommen soll, kann mit dem Befehl `skinparam packageStyle` festgelegt werden. Alternativ kann ein Stereotyp in der Paketdefinition verwendet werden.

```
@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
```





Außerdem ist es möglich, Abhängigkeiten zwischen Paketen zu definieren, wie dies im folgenden Beispiel gezeigt wird:

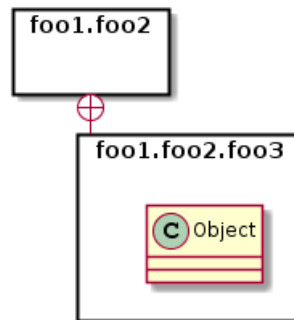
```
@startuml
skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Namensraum

In Paketen ist der Name einer Klasse der eindeutige Bezeichner der Klasse. Das bedeutet, dass man nicht zwei Klassen mit dem gleichen Namen in unterschiedlichen Paketen haben kann.

In diesem Fall sollte ein Namensraum anstelle eines Pakets verwendet werden.

Man kann auf eine Klasse aus einem anderen Namensraum verweisen, indem man den vollqualifizierten Namen der Klasse angibt. Klassen aus dem Standardnamensraum werden mit einem beginnenden Punkt gekennzeichnet.

Beachten Sie, dass ein Namensraum nicht explizit festgelegt werden muss: Eine vollqualifizierte Klasse verwendet automatisch den richtigen Namensraum.

```
@startuml
class BaseClass

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

.BaseClass <|-- Meeting
}

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

```

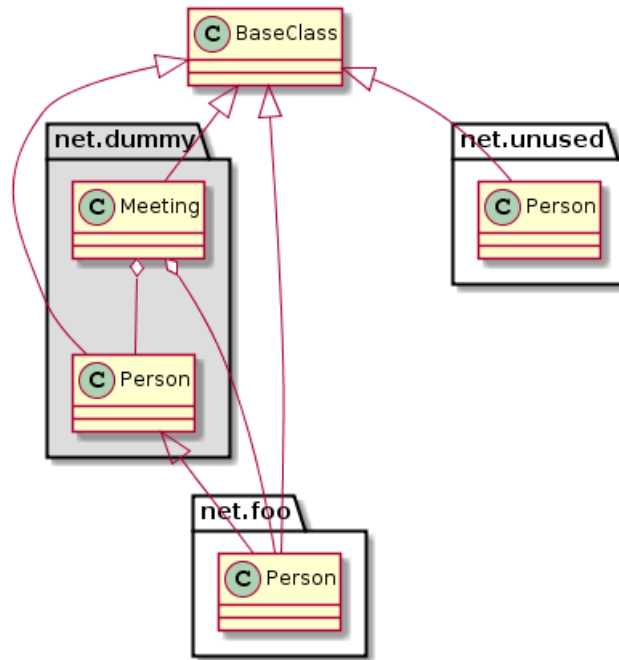
.BaseClass <|-- Person

net.dummy.Meeting o-- Person
}

.BaseClass <|-- net.unused.Person

@enduml

```



3.19 Automatische Erzeugung eines Namensraums

Über folgenden Befehl kann ein anderes Trennzeichen (als der Punkt) definiert werden: `set namespaceSeparator ???`.

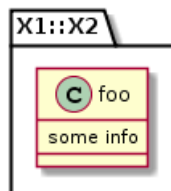
```

@startuml

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

@enduml

```



Die automatische Erzeugung eines Pakets kann mit `set namespaceSeparator none` deaktiviert werden.

```

@startuml

set namespaceSeparator none
class X1.X2.foo {
}

@enduml

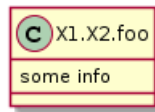
```

```

some info
}

@enduml

```



3.20 Lollipop Schnittstellen

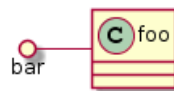
Mit der folgenden Syntax kann man Schnittstellen von Klassen definieren:

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

```

@startuml
class foo
bar ()- foo
@enduml

```



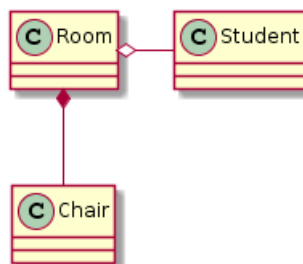
3.21 Ändern der Pfeilrichtung

Normalerweise werden Beziehungen zwischen Klassen mit zwei Strichen `--` definiert und die Klassen werden Vertikal angeordnet. Verwendet man nur einen Strich (oder Punkt), dann werden die Klassen horizontal angeordnet so wie im folgenden Beispiel zu sehen ist:

```

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

```

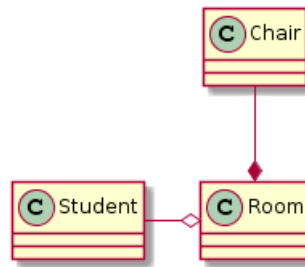


Man kann die Richtung auch durch das Umdrehen der Verbindung ändern:

```

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

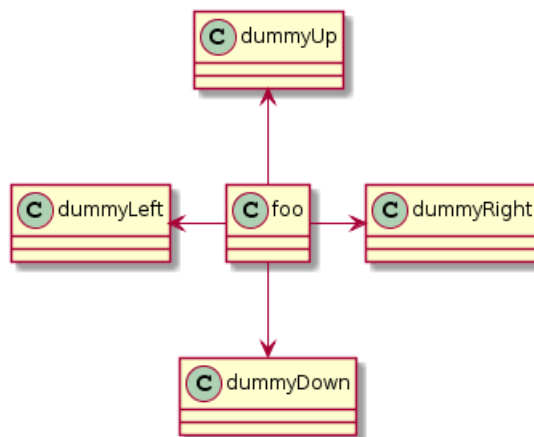
```



Außerdem ist es möglich, die Richtung der Pfeile durch Hinzufügen der `left`, `right`, `up` oder `down` Schlüsselwörter innerhalb der Pfeile zu verändern:

```

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



Die Länge der Pfeile kann verkürzt werden, in dem man nur den ersten Buchstaben für die Richtung verwendet (zum Beispiel, `-d-` anstelle von `-down-`) oder die ersten beiden Buchstaben (`-do-`)

Bitte verwenden Sie diese Möglichkeit nur wenn es unbedingt sein muss: *Graph Viz* liefert normalerweise recht gute Ergebnisse ohne das manuell eingegriffen werden muss.

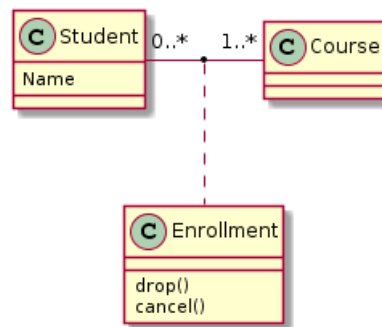
3.22 Assoziationsklassen

Nach dem man eine Beziehung zwischen zwei Klassen definiert hat, kann man eine *association class* definieren. Hierzu ein Beispiel:

```

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

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

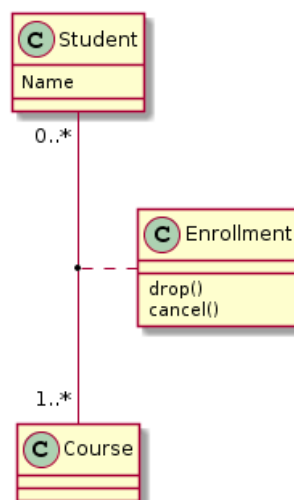



Die Richtung lässt ich aber auch ändern:

```

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

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



3.23 Der Skinparam-Befehl

Mit dem `skinparam` Befehl können die Farben und die Schriften der Zeichnung verändert werden. Der Befehl kann wie folgt verwendet werden:

- In der Definition des Diagramms, so wie alle anderen Befehle auch,
- In einer Include-Datei,
- In einer Konfigurationsdatei, die durch die Kommandozeile oder einen ANT-Task übergeben wird.

```

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

```

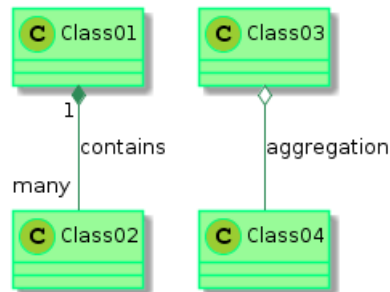
skinparam stereotypeCBackgroundColor YellowGreen

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

Class03 o-- Class04 : aggregation

@enduml

```



3.24 Das Aussehen von Stereotypen verändern

Es ist möglich die Farbe und die Schriftart der Klassen zu verändern, die mit einem Stereotypen ausgezeichnet sind.

```

@startuml

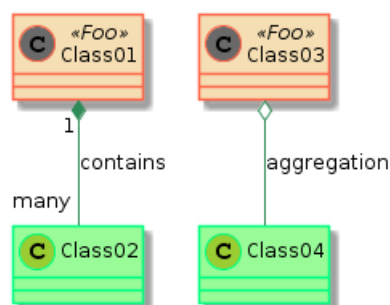
skinparam class {
  BackgroundColor PaleGreen
  ArrowColor SeaGreen
  BorderColor SpringGreen
  BackgroundColor<<Foo>> Wheat
  BorderColor<<Foo>> Tomato
}
skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

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

Class03 o-- Class04 : aggregation

@enduml

```



3.25 Farbverlauf

Mit der Notation können individuelle Farben für Klassen oder Notizen definiert werden.

Es kann entweder der Standardname der Farbe oder der RGB Code verwendet werden.

Für den Hintergrund kann ebenfalls ein Farbverlauf verwendet werden: Zwei Farbnamen getrennt durch:

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

abhängig von der Richtung des Verlaufs.

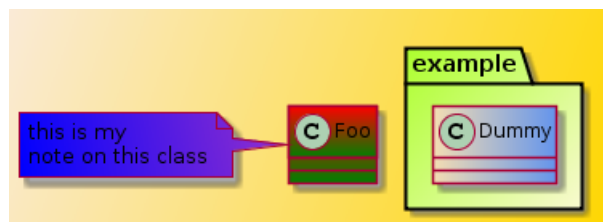
So könnte dies zum Beispiel aussehen:

```
@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 Hilfe beim Layout

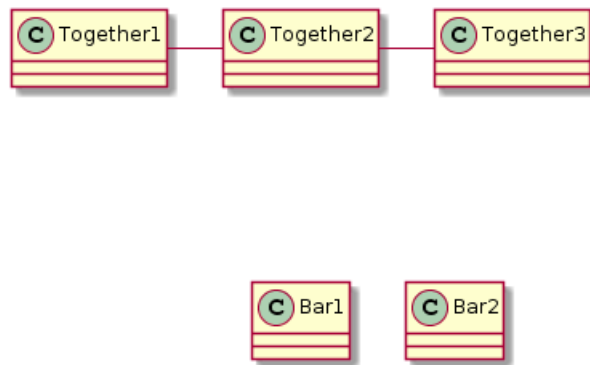
Sometimes, the default layout is not perfect...

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

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

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

@enduml
```



3.27 Große Dateien aufteilen

Manchmal erhält man sehr große Bilddateien. Mit dem "page (hpages)x(vpages)" Befehl kann das erzeugte Bild auf mehrere Dateien verteilt werden:

Mit dem "page (hpages)x(vpages)" Befehl kann das erzeugte Bild auf mehrere Dateien aufgeteilt werden:

hpages gibt die Anzahl von horizontalen Seiten an, und vpages gibt die Anzahl von vertikalen Seiten an.

You can also use some specific skinparam settings to put borders on splitted pages (see example).

```

@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

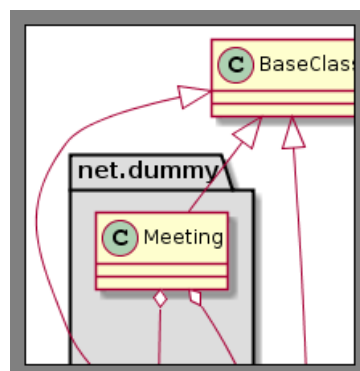
  .BaseClass <|-- Meeting
}

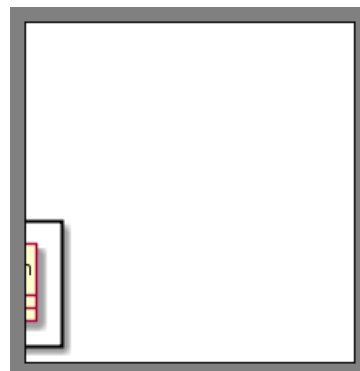
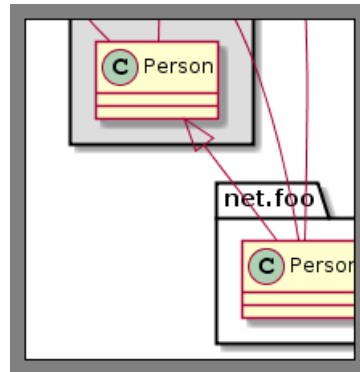
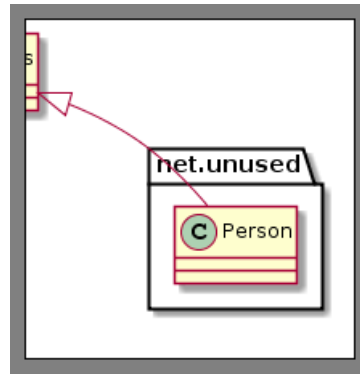
namespace net.foo {
  net.dummy.Person <|-- Person
  .BaseClass <|-- Person

  net.dummy.Meeting o-- Person
}

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

```





4 Aktivitätsdiagramm

4.1 Einfache Aktivität

Mit (*) kann der Startknoten und der Endknoten des Aktivitätsdiagramms festgelegt werden.

In einigen Fällen kann man (*top) verwendet um den Startpunkt an den Anfang des Diagramms zu verlegen.

mit --> können Pfeile definiert werden.

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

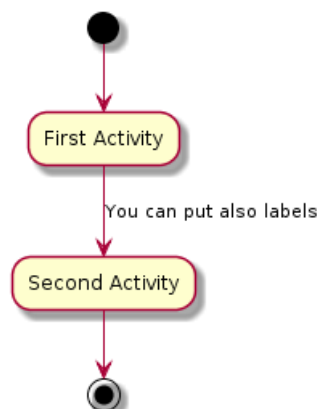


4.2 Beschriftungen an Pfeilen

Ein Pfeil beginnt automatisch an der zuletzt verwendeten Aktivität.

Pfeile lassen sich beschriften in dem man den Text für die Beschriftung in eckige Klammern ([und]) direkt hinter die Definition des Pfeils schreibt.

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



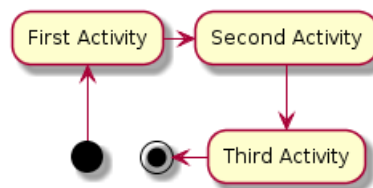
4.3 Pfeilrichtung ändern

Mit dem Symbol -> kann ein waagerechter Pfeil erstellt werden. Mann kann die Richtung der Pfeile auch mit der folgenden Syntax beeinflussen:



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

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

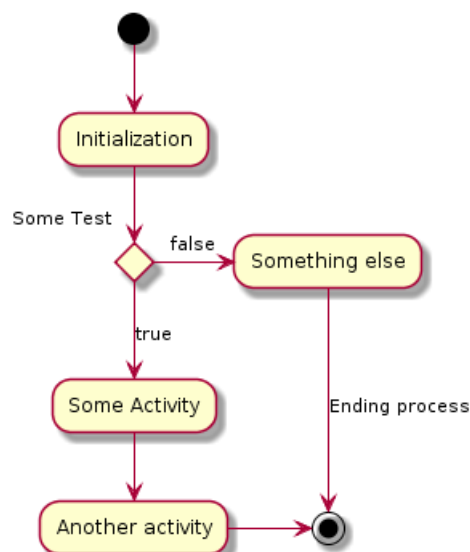


4.4 Verzweigungen

Mit den if/then/else Schlüsselworten können Verzweigungen definiert werden.

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

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

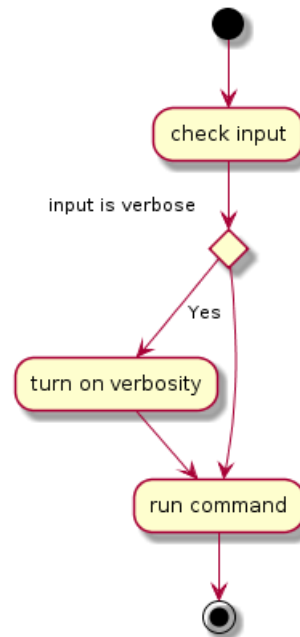


Unglücklicherweise muss man manchmal die gleiche Aktivität im Diagrammtext wiederholen.

```

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

```



4.5 Mehr über Verzweigungen

Normalerweise ist ein eine Verzweigung mit der zuletzt definierten Aktivität verbunden. Mit dem **if** Schlüsselwort ist es aber möglich, diese Voreinstellung zu überschreiben.

Außerdem kann man Verzweigungen auch schachteln.

```

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

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

else
->[false] "activity 2"

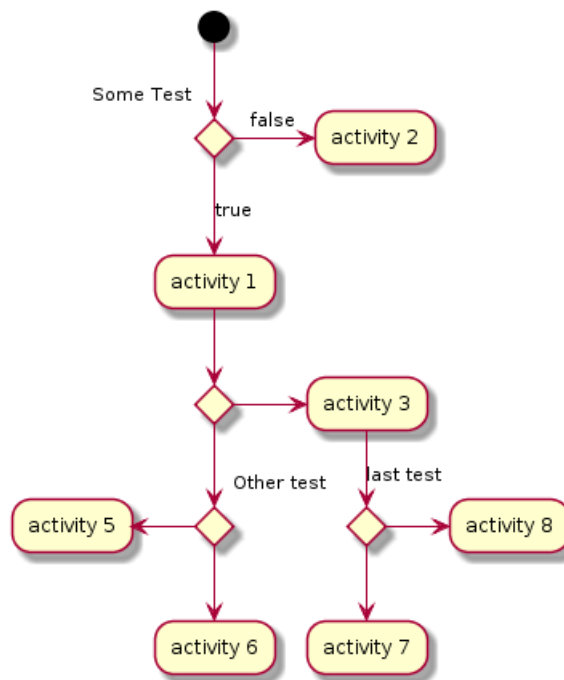
endif

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

```



```
endif
@enduml
```



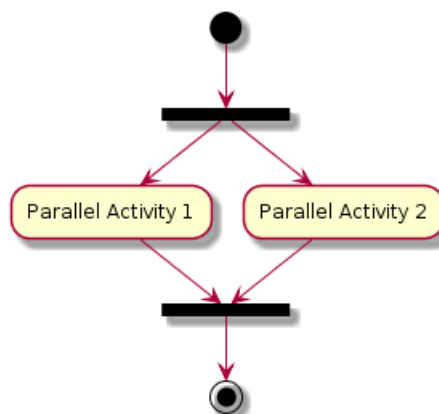
4.6 Synchronisation

Mit "=== code ===" können Synchronisationsbalken erzeugt werden.

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

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

--> (*)
@enduml
```



4.7 Lange Beschreibungen für Aktivitäten

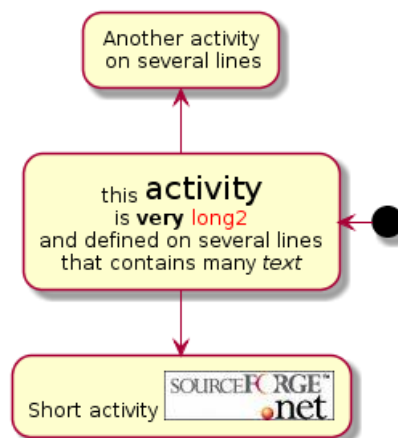
Die Beschreibung einer Aktivität kann sich auch über mehrere Zeilen erstrecken. Mit dem `\n` Symbol kann ein Zeilenvorschub in die Beschreibung eingefügt werden. Außerdem kann man HTML Tags verwenden. Hier ein Beispiel:

Mit dem Schlüsselwort `as` kann man auch eine kurze Kodierung zur Aktivität hinzufügen. Diese Kodierung kann später in der Diagrammbeschreibung verwendet werden.

```
@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 Notizen

Mit den folgenden Befehlen können einer Aktivität Notizen zugeordnet werden: `note left`, `note right`, `note top` or `note bottom`, Gleich nach der Beschreibung der Aktivität die man festhalten will.

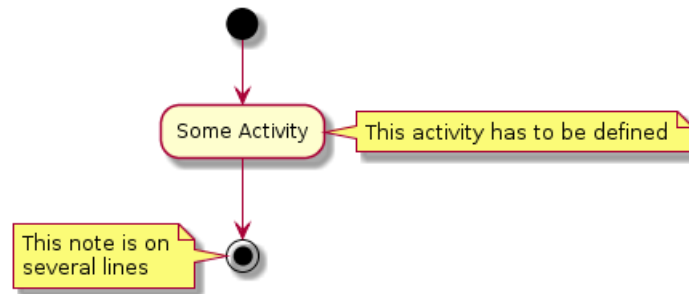
Wenn Sie eine Notiz für den Startpunkt erstellen wollen müssen Sie diese Notiz ganz am Anfang des Diagramms definieren.

Es ist auch möglich, eine Notiz mit mehreren Zeilen zu erstellen. Dazu werden die `end note` Schlüsselworte verwendet.

```
@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 Partitionen

Partitionen können mit dem `partition` Schlüsselwort erzeugt werden. Dabei kann auch eine Hintergrundfarbe festgelegt werden. (Durch einen HTML Farbcode oder Namen).

Neue Aktivitäten werden automatisch in die zuletzt verwendete Partition eingefügt.

Eine Partition lässt sich über das `end partition` Schlüsselwort schließen.

```

@startuml

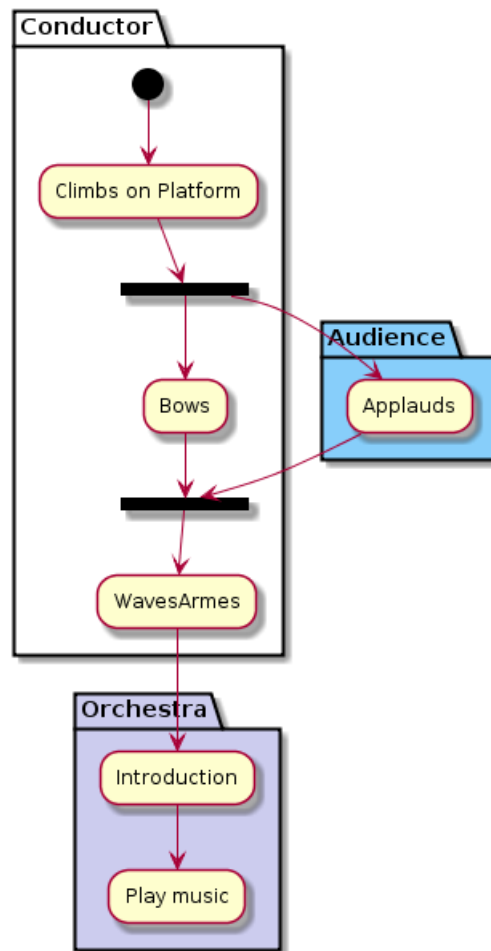
partition Conductor {
(*) --> "Climbs on Platform"
--> === S1 ===
--> Bows
}

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

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

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

@enduml
  
```



4.10 Der Skinparam Befehl

Mit dem `skinparam` Befehl kann man die Farbe und die Schriftart der Zeichnung verändern. Man kann diesen Befehl wie folgt verwenden:

- In der definition des Diagramms, so wie jeden anderen Befehl auch,
- In einer eingebundenen Datei,
- In einer Konfigurationsdatei, die per Kommandozeile oder ANT-Task übergeben wird.

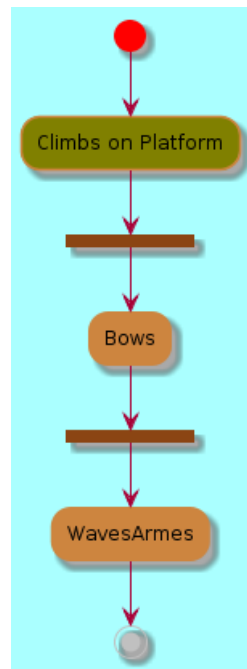
Man kann spezifische Farben und Schriften für immer wiederkehrende Aktivitäten festlegen.

```
@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 Oktagon

Man kann die Form zu einem Oktagon mit dem Befehl `skinparam activityShape octagon` ändern.

```

@startuml
'Default is skinparam activityShape roundBox
skinparam activityShape octagon

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

@enduml

```



4.12 Komplettes Beispiel

```

@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

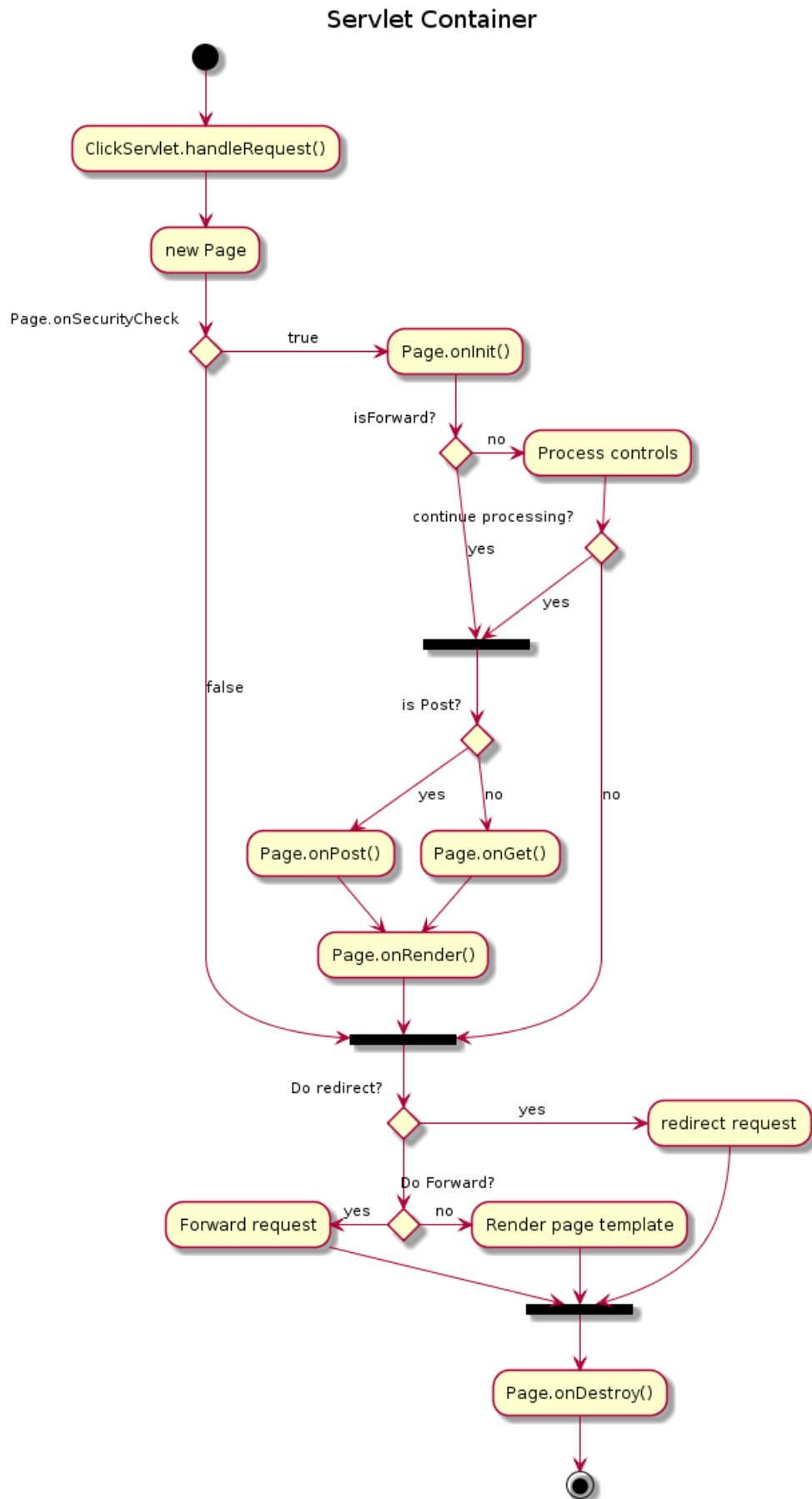
if "is Post?" then
-->[yes] "Page.onPost()"
--> "Page.onRender()" as render
--> ===REDIRECT_CHECK===
else
-->[no] "Page.onGet()"
--> render
endif

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 Aktivitätsdiagramm (Beta)

Die momentane Syntax für das Aktivitätsdiagramm hat einige Einschränkungen und Nachteile (zum Beispiel ist es schwierig zu pflegen).

Mit **beta version** wird eine komplett neue Syntax und Umsetzung den Benutzern (angefangen bei V7947) vorgeschlagen, sodaß wir besseres Format und Syntax definieren können.

Another advantage of this new implementation is that it's done without the need of having Graphviz installed (as for sequence diagrams).

Die neue Syntax wird die alte ersetzen. Allerdings wird aus Gründen der Kompatibilität die alte Syntax noch weiter erkannt werden um *ascending compatibility* sicherzustellen.

Benutzer werden schlicht aufgefordert, auf die neue Syntax zu migrieren.

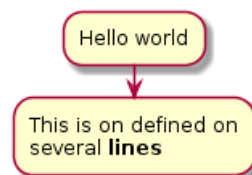
5.1 Einfache Aktivität

Aktivitäts Label beginnen mit **:** und enden mit **;**.

Textformatierungen können mit Creole Wiki Syntax erfolgen.

Sie sind in ihrer Festlegungsreihenfolge indirekt verbunden.

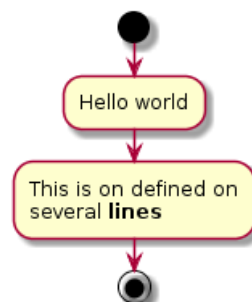
```
@startuml
:Hello world;
:This is on defined on
several lines;
@enduml
```



5.2 Start Stop

Man kann die **start** und **stop** Schlüsselwörter verwenden um Beginn und Ende des Diagramms zu kennzeichnen.

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

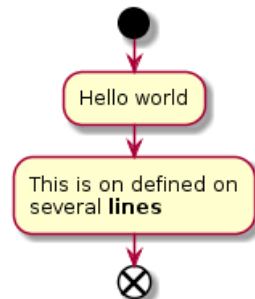


Das Schlüsselwort **end** beendet ebenfalls das Diagramm, zeigt aber als Symbol den durchkreuzten Kreis.


```

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

```



5.3 Bedingung

Man kann `if`, `dann` und `else` Schlüsselwörter verwenden, um Tests ins Diagramm einzufügen. Etiketten können mit Klammern versehen werden.

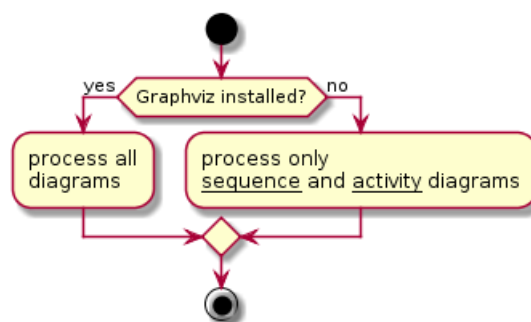
```

@startuml
start

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

stop
@enduml

```



Man kann das `elseif` Schlüsselwort für mehrere Tests verwenden:

```

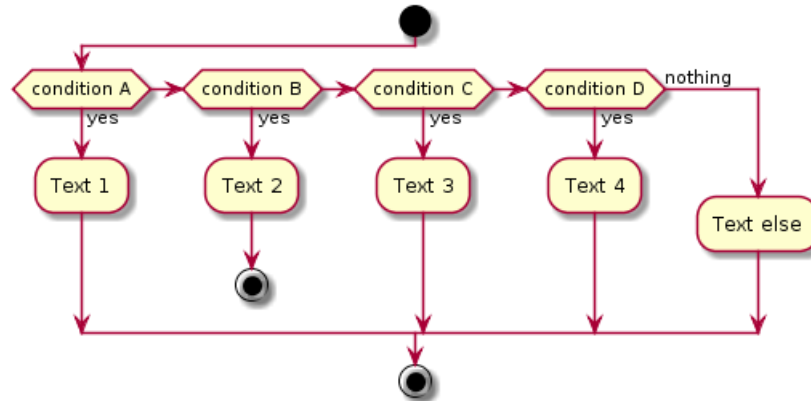
@startuml
start
if (condition A) then (yes)
:Text 1;
elseif (condition B) then (yes)
:Text 2;
stop
elseif (condition C) then (yes)
:Text 3;
elseif (condition D) then (yes)
:Text 4;

```

```

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

```



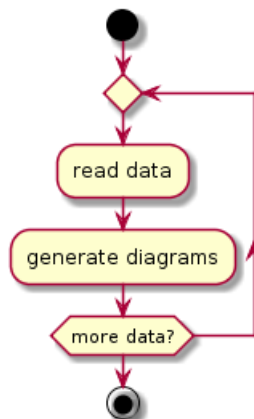
5.4 Repeat-Schleife

Mit den `repeat` und `repeatwhile` Schlüsselwörtern können Repeat-Schleifen dargestellt werden.

```

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

```



5.5 While-Schleife

Mit den `while` und `end while` Schlüsselwörtern können While-Schleifen dargestellt werden.

```

@startuml
start
while (data available?)

```

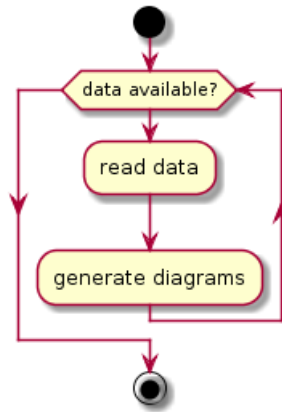
```

:read data;
:generate diagrams;
endwhile

stop

@enduml

```

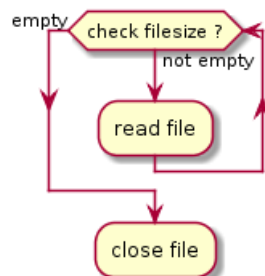


Es ist möglich eine Beschriftung hinter dem `endwhile` Schlüsselwort anzugeben. Eine Beschriftung kann aber auch mit dem `is` Schlüsselwort hinzugefügt werden..

```

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

```



5.6 Parallele Verarbeitung

Mit dem `fork`, `fork again` und `end fork` Schlüsselworten kann eine parallele Verarbeitung angezeigt werden.

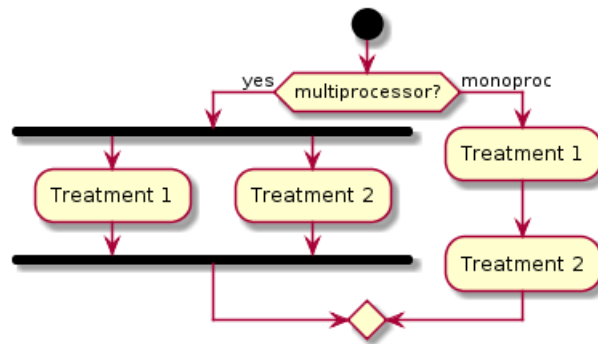
```

@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 Notizen

Textformatierung kann mit Creole Wiki Syntax gemacht werden.

Eine Anmerkung kann auch schweben, indem das Schlüsselwort **floating** benutzt wird.

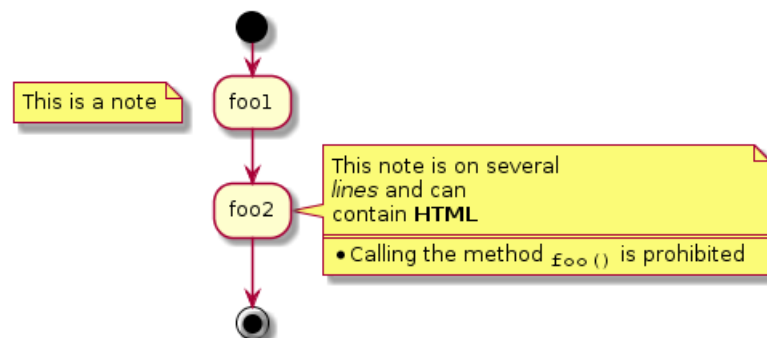
```

@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 Farben

Man kann spezielle Farben für gewisse Aktivitäten verwenden

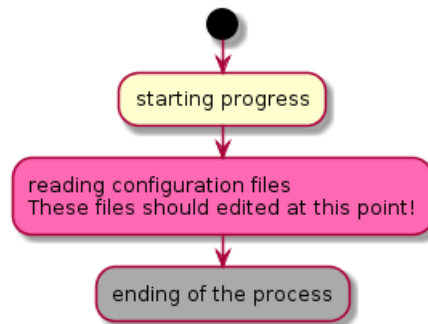
```

@startuml

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

@enduml

```



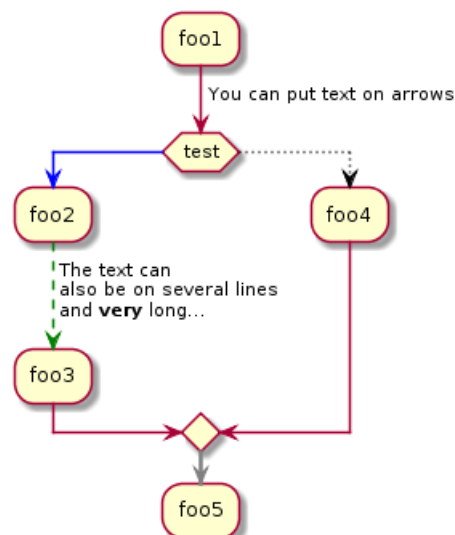
5.9 Pfeile

Über die -> Notation, können Texte an den Pfeilen angezeigt werden und die Farbe der Pfeile geändert werden.

Es sind auch gepunktete, gestrichelte, dicke oder unsichtbare Pfeile möglich.

```

@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 Gruppierung

Aktivitäten können durch Partitionen gruppiert werden:

```

@startuml
start
partition Initialization {
  
```



```

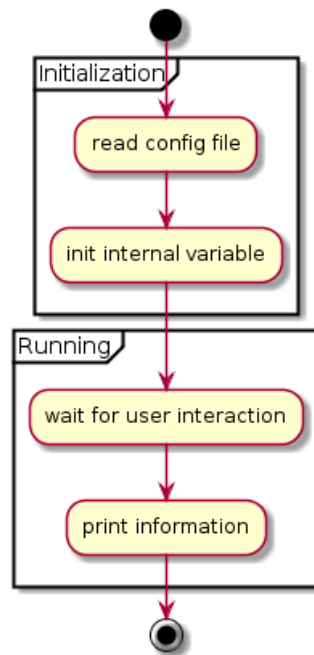
:read config file;
:init internal variable;
}
partition Running {
:wait for user interaction;
:print information;
}

```

```

stop
@enduml

```



5.11 Schwimmbahnen

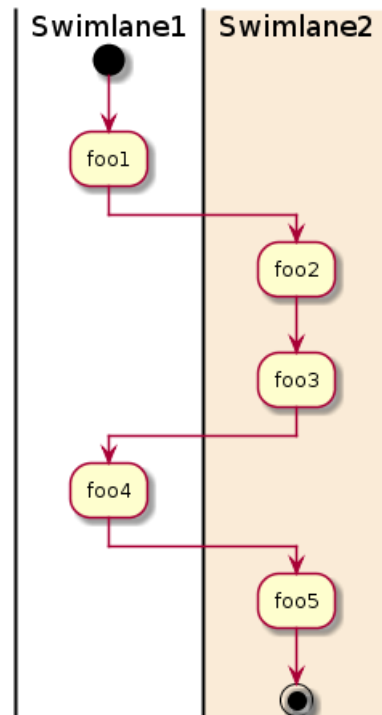
Mit dem Pipe Zeichen | kann man Schwimmbahnen definieren.

Es ist auch möglich die Schwimmbahnfarbe zu ändern.

```

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

```

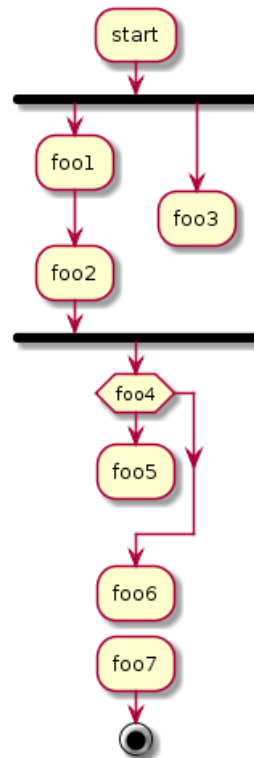


5.12 Abtrennen

Es ist möglich mit dem **detach** Schlüsselwort einen Pfeil zu entfernen.

```

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



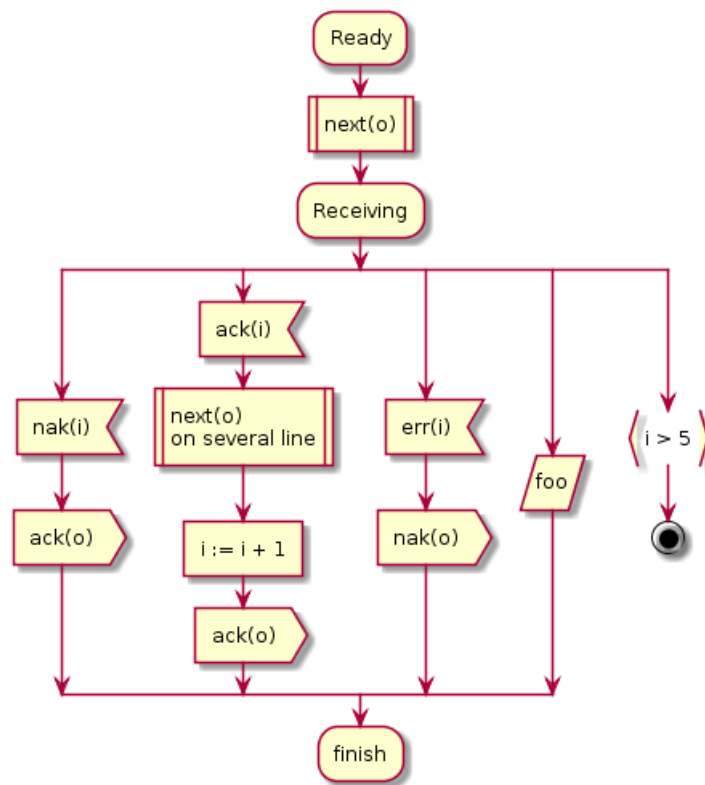
5.13 SDL-Diagramme

Durch Ändern des letzten Separators ; können Sie unterschiedliche Wiedergabe für die Aktivität einstellen:

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

```

@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 Komplettes Beispiel

```

@startuml

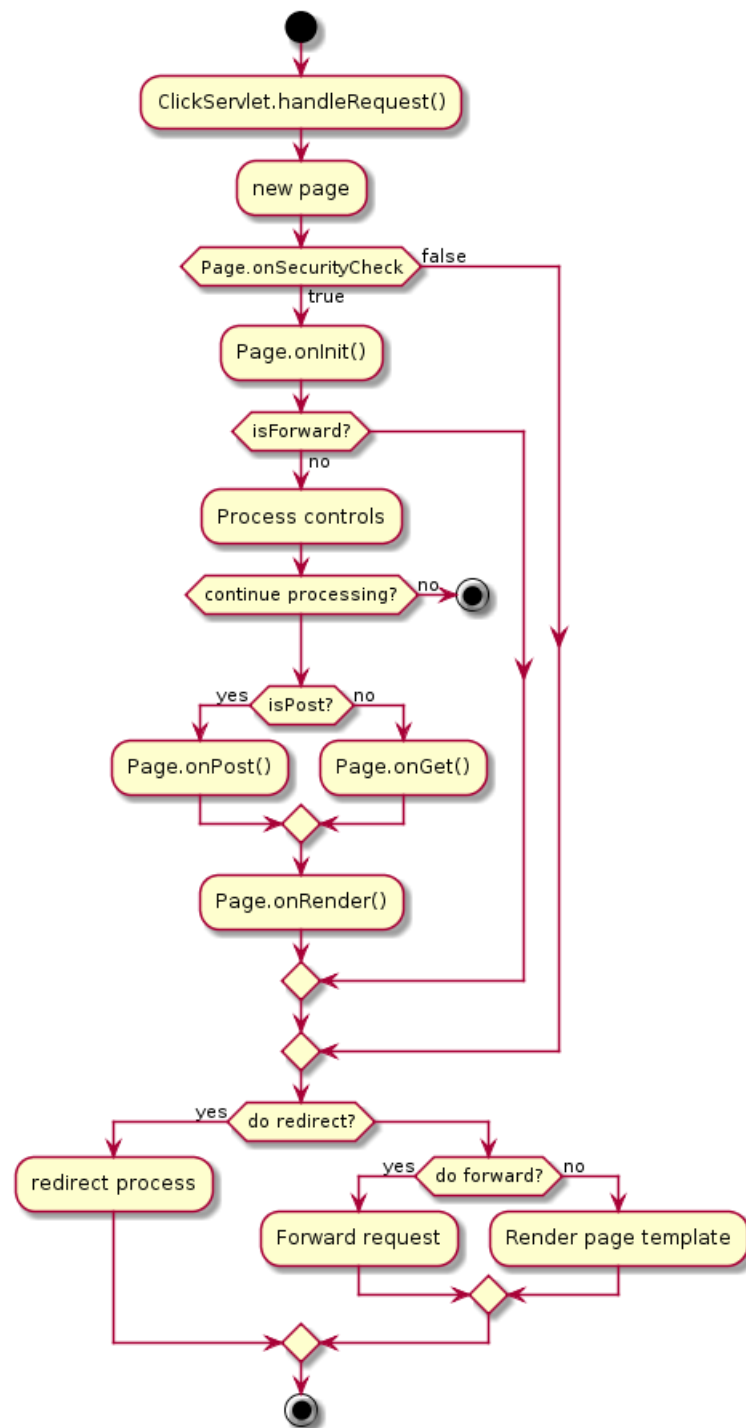
start
:ClickServlet.handleRequest();
:new page;
if (Page.onSecurityCheck) then (true)
:Page.onInit();
if (isForward?) then (no)
:Process controls;
if (continue processing?) then (no)
stop
endif
endif

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
  
```



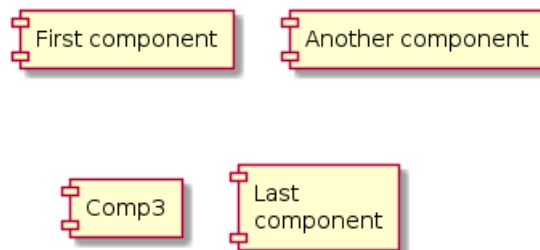
6 Komponentendiagramm

6.1 Komponenten

Komponenten werden mittels eckiger Klammern definiert.

Alternativ kann das Schlüsselwort **component** verwendet werden, um eine Komponente zu definieren. Mittels Schlüsselwort **as** lassen sich Aliase definieren. Aliase können verwendet werden, wenn Beziehungen definiert werden.

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



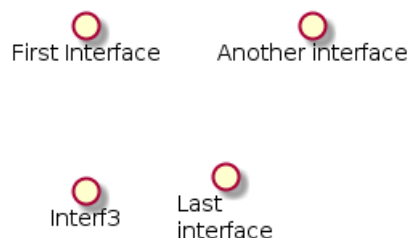
6.2 Schnittstellen

Schnittstellen werden mit zwei Runden Klammern "()" definiert.

Alternativ kann das Schlüsselwort **interface** verwendet werden, um Schnittstellen zu definieren. Mittels Schlüsselwort **as** lassen sich Aliase definieren. Aliase können verwendet werden, wenn Beziehungen definiert werden.

Die Deklaration von Schnittstellen ist optional.

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



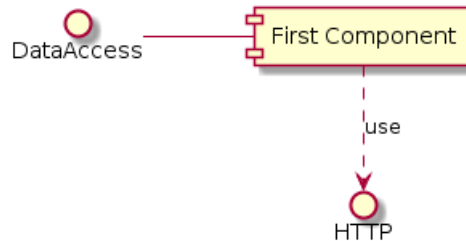
6.3 Beispiel

Verbindungen zwischen Elementen können mit folgenden Symbolen erstellt werden: **..** (gestrichelte Linie), **--** (ausgezogene Linie), and **-->** (Pfeile).

```

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

```



6.4 Notizen

Schlüsselwörter: `note left of`, `note right of`, `note top of`, `note bottom of` Diese Schlüsselwörter können eingesetzt werden, um Notizen für ein einzelnes Objekt zu erstellen.

Eine Notiz kann mit `note` definiert werden. Danach kann sie mittels `..` mit anderen Objekten verbunden werden.

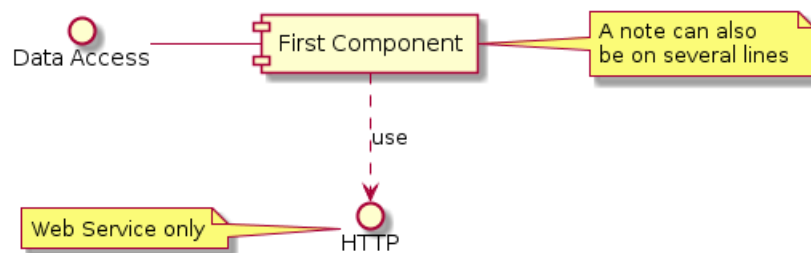
```

@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 Gruppierende Komponenten

Mit package lassen sich Komponenten und Schnittstellen gruppieren.

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

```

@startuml

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

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

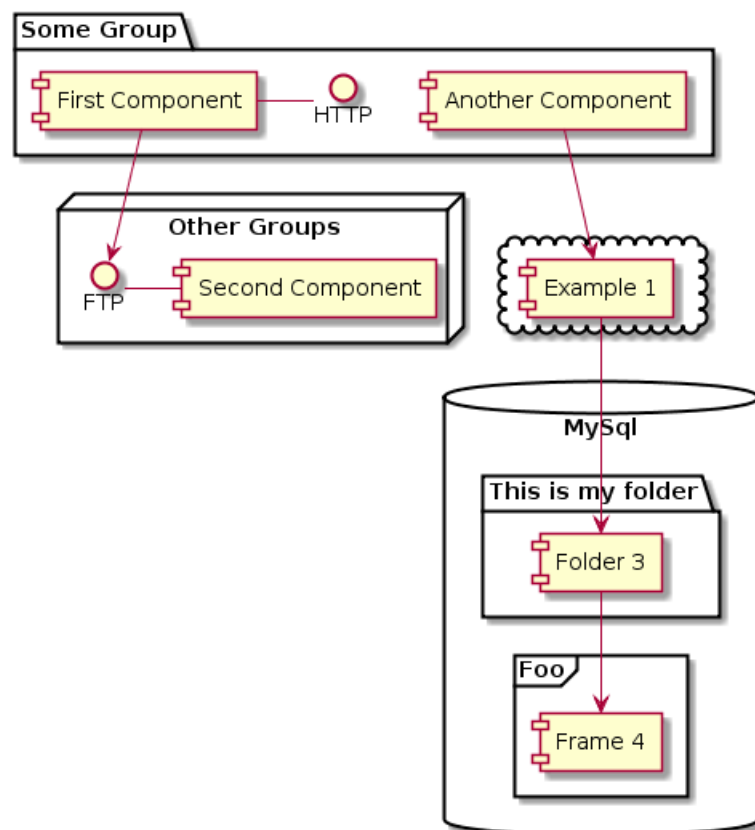
cloud {
  [Example 1]
}

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

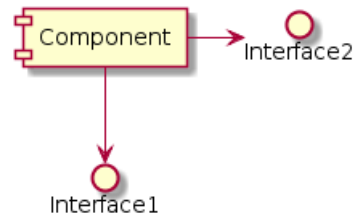
```



6.6 Ändern der Pfeilrichtung

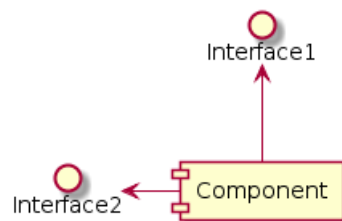
Verbindungen werden mit zwei Minus-Zeichen -- definiert und sind vertikal orientiert. Um eine horizontale Orientierung zu erhalten, kann die Verbindung mit nur einem Minus-Zeichen (oder Punkt) definiert werden:

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



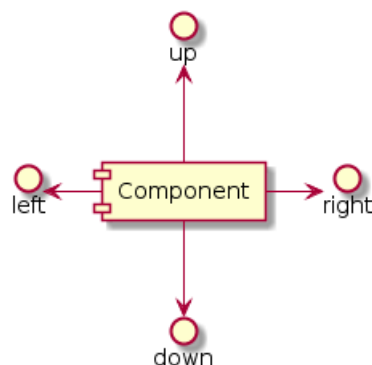
Die Pfeilsymbole können umgedreht werden, um die Pfeilrichtung zu ändern:

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



Die Pfeilrichtung lässt sich auch mit den Schlüsselwörtern **left**, **right**, **up** und **down** ändern. Diese Schlüsselwörter werden innerhalb des Pfeil-Symbols eingesetzt:

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



Die Pfeillänge kann verkürzt werden, wenn bei der Deklaration der Pfeilrichtung nur der Anfangsbuchstabe (oder ersten zwei Anfangsbuchstaben) verwendet werden: Beispielsweise **-d-** oder **-do-** statt **-down-**.

Diese Funktionalität ist jedoch mit Bedacht einzusetzen, da *GraphViz* normalerweise gute Resultate ohne manuelle Eingriffe erzielt.

6.7 UML2-Notation verwenden

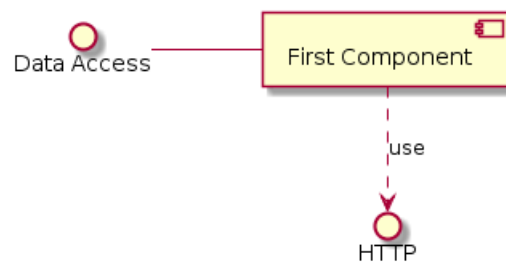
Der skinparam componentStyle uml2 Befehl wird verwendet, um in die UML2 Notation umzuschalten.

```
@startuml
skinparam componentStyle uml2

interface "Data Access" as DA

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

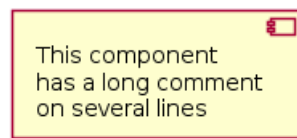
@enduml
```



6.8 Mehrzeilige Beschreibung

Es ist möglich mehrzeilige Beschreibungen zu erstellen mithilfe von eckigen Klammern

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



6.9 Individuelle Farben

Eine Farbe kann nach der Komponenten Definition angegeben werden.

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



6.10 Verwendung von Sprites in Stereotypen

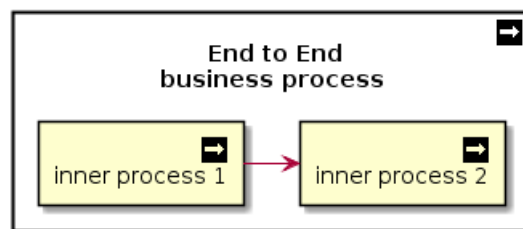
Sie können Sprites innerhalb von stereotypen Komponenten verwenden.

```

@startuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFF0FFFFF
FFFFFFFFF0FFFFF
FF000000000000FF
FF000000000000FF
FF000000000000FF
FFFFFFFFF0FFFFF
FFFFFFFFF0FFFFF
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 Der Skinparam Befehl

Mit `skinparam` lassen sich Farben und Schriftarten des Diagramms ändern.

Der `skinparam`-Befehl lässt sich in folgenden Bereichen einsetzen:

- In Diagramm-Definitionen, wie alle Schlüsselwörter.
- In zu inkludierenden Dateien.
- In Konfigurationsdateien, welche via Kommandozeile oder ANT-Task zur Verfügung gestellt werden.

Es können unterschiedliche Farben und Schriftarten für Komponenten und Schnittstellen verwendet werden.

```

@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
}

```



```

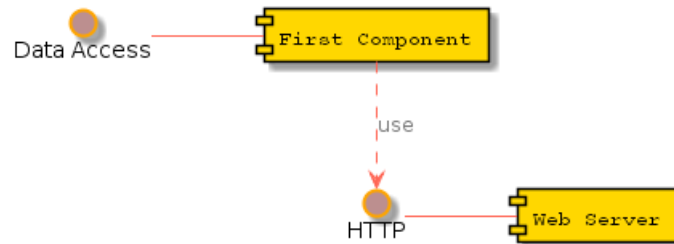
ArrowColor #FF6655
ArrowFontColor #777777
}

() "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>>

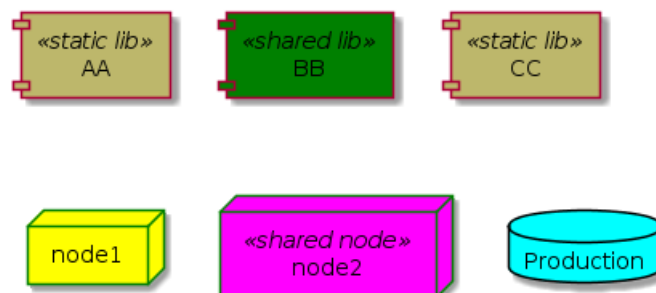
node node1
node node2 <<shared node>>
database Production

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 Zustandsdiagramme

7.1 Einfache Zustandsdiagramme

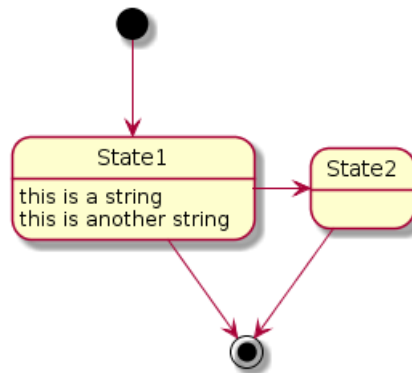
Für den Startpunkt und den Endpunkt im Zustandsdiagramms können Sie das Symbol [*] verwenden.

Verwenden Sie --> um Pfeile zu definieren.

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

State1 -> State2
State2 --> [*]

@enduml
```



7.2 Verschachtelter Zustand

Ein Zustand kann auch verschachtelt werden. Dies funktioniert mit dem **state** Schlüsselwort und den geschweiften Klammern.

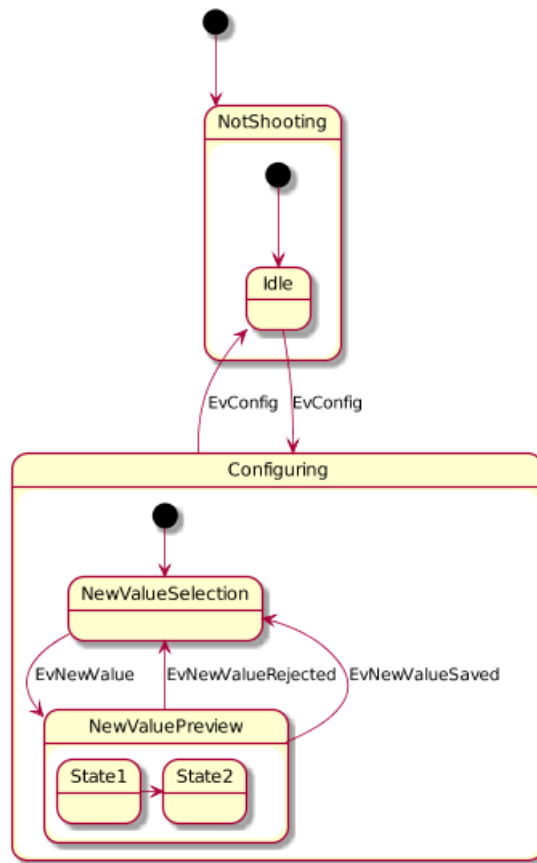
```
@startuml
scale 350 width
[*] --> NotShooting

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 Lange Bezeichnungen für einen Zustand

Mit dem `state` Schlüsselwort können auch längere Bezeichnungen eines Status definiert werden.

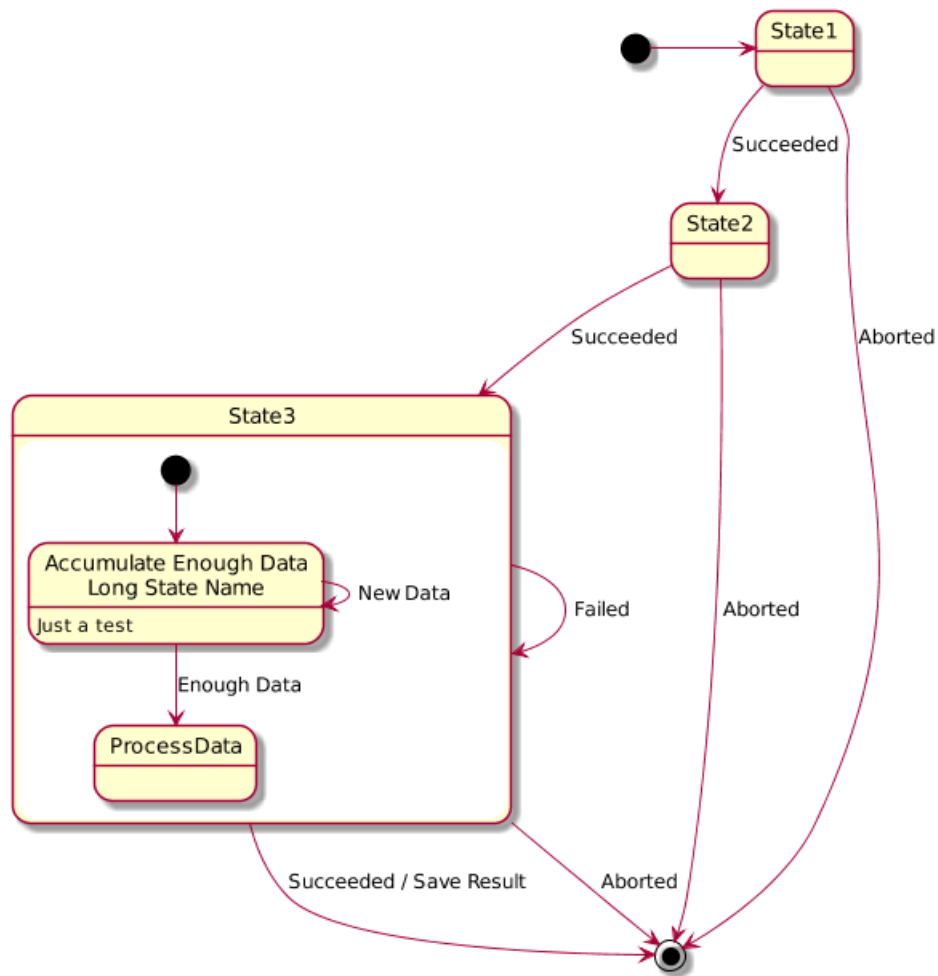
```

@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 Konkurrierende Zustände

Miteinander konkurrierende Zustände können mit dem "--" Symbol in einem zusammengesetzten Zustand zusammengefasst werden..

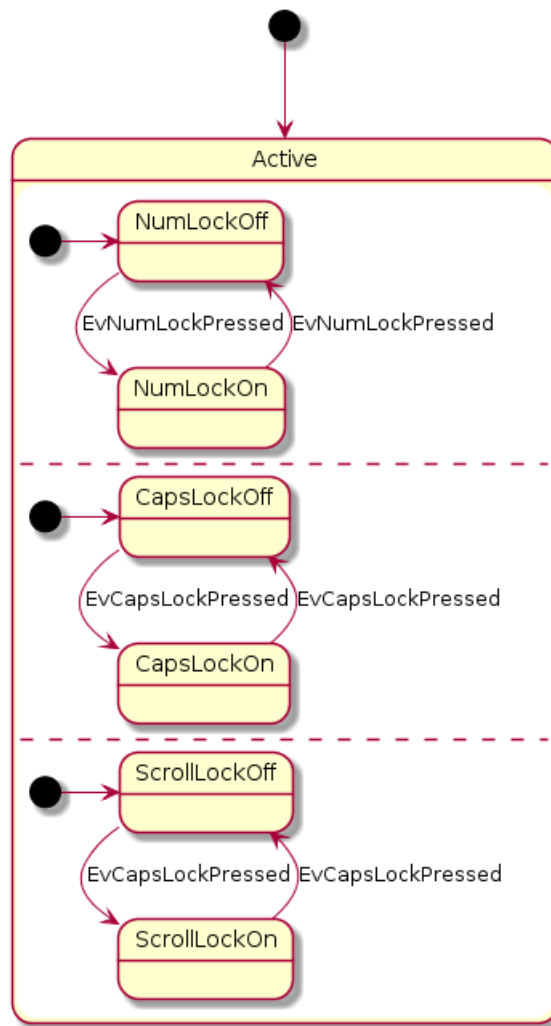
```

@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

```



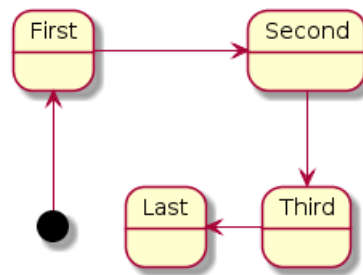
7.5 Pfeilrichtung

Mit dem `->` Symbol können waagerechte Pfeile erzeugt werden. Man kann die Richtung der Pfeile außerdem mit der folgenden Syntax festlegen:

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

```

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



Man kann die Länge eines Pfeils verkürzen, indem man nur den ersten Buchstaben der Richtung verwendet (zum Beispiel, `-d-` anstelle von `-down-`) oder die ersten beiden Buchstaben (`-do-`).

Beachten Sie, dass Sie mit dieser Möglichkeit sorgfältig umgehen: *GraphViz* liefert normalerweise recht gute Ergebnisse, ohne dass manuell eingegriffen werden muss.

7.6 Notizen

Notizen können mit den `note left of`, `note right of`, `note top of`, `note bottom of` Schlüsselworten

an die Zustände gebunden werden. Die Notizen können sich auch über mehrere Zeilen erstrecken.

```

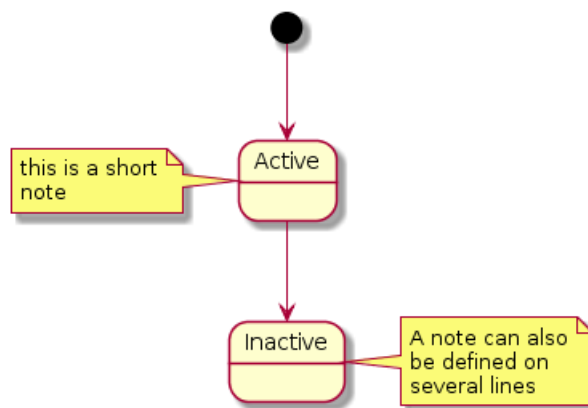
@startuml

[*] --> 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
  
```

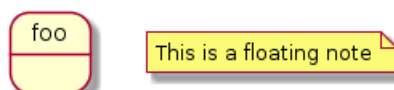


Es ist auch möglich, freistehende Notizen hinzuzufügen.

```

@startuml

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



7.7 Mehr über Notizen

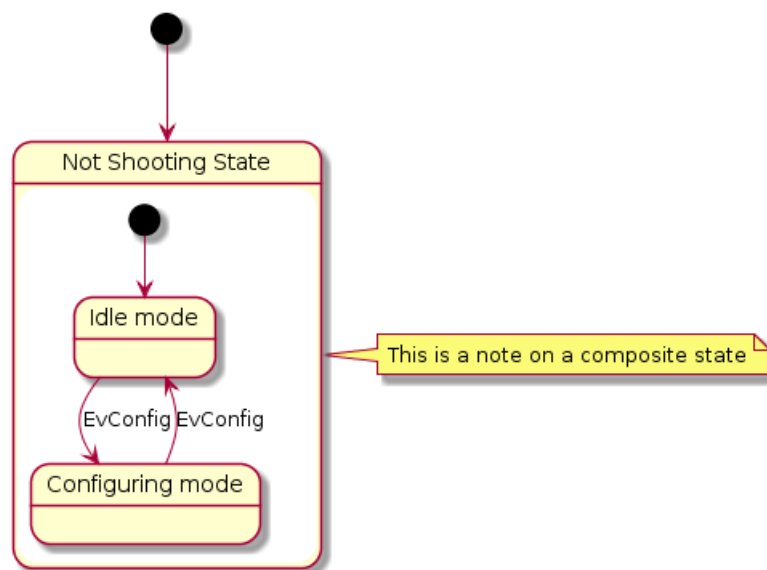
Es ist auch möglich, Notizen für einen verbundenen Zustand zu erstellen.

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

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

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

@enduml
```



7.8 Skinparam

Mit `skinparam` kann man die Farben und Schriftarten der Zeichnung bearbeiten

Man kann diesen Befehl verwenden:

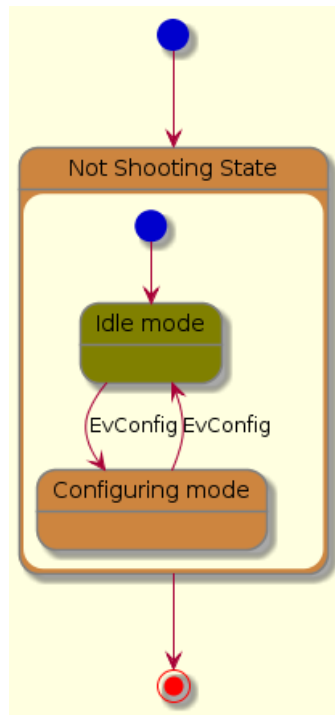
- In der Definition des Diagramms, so wie alle anderen Befehle auch,
- In einer Include-Datei,
- In einer Konfigurationsdatei, die durch die Kommandozeile oder einen ANT-Task übergeben wird.

Es können spezielle Farben und Schriftarten für stereotypische Zustände definiert werden.

```
@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
```

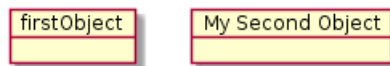


8 Objektdiagramm

8.1 Definition von Objekten

Eine Instanz eines Objekts wird mit dem Schlüsselwort `object` definiert.

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



8.2 Beziehungen zwischen Objekten

Beziehungen zwischen Objekten werden mit den folgenden Symbolen definiert:

Vererbung	< --	
Komposition	*--	
Aggregation	o--	

Um eine gestrichelte Linie zu zeichnen, kann "--" durch ".." ersetzt werden.

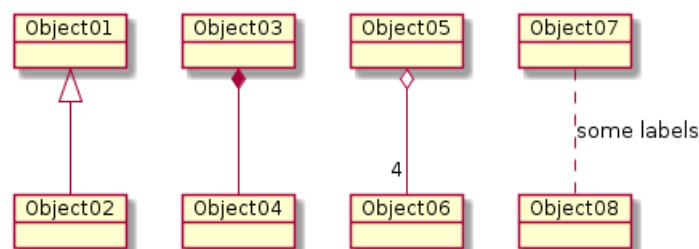
Auf diese Weise können die folgenden Diagramme gezeichnet werden:

Mit `:` gefolgt von dem Beschriftungstext kann an die Beziehung eine Beschriftung hinzugefügt werden.

Um die Kardinalität anzugeben, können doppelte Anführungszeichen auf jeder Seite der Beziehung verwendet werden.

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

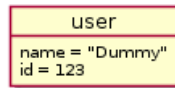
Object01 <|-- Object02
Object03 *-- Object04
Object05 o-- "4" Object06
Object07 .. Object08 : some labels
@enduml
```



8.3 Hinzufügen von Attributen

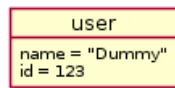
Um Attribute zu deklarieren, wird das Symbol `:"`, gefolgt vom Feldnamen, verwendet:

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



Es ist auch möglich, alle Attribute eines Objekts zwischen geschweiften Klammern {} aufzuführen:

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



8.4 Gemeinsam mit klassendiagrammen verwendete Funktionen

- Sichtbarkeit
- Hinzufügen von Notizen
- Verwendung von Paketen
- Formatieren der Ausgabe

9 Allgemeine Befehle

9.1 Kommentare

Kommentare werden mit einem einfachen Anführungszeichen ' eingeleitet.

Sie können Kommentare über mehrere Zeilen schreiben, in dem Sie '/' für den Start und '/' für das Ende verwenden.

9.2 Kopf- und Fusszeile

Mit `header` oder `footer` können Kopf- und Fußzeilen für jedes generierte Diagramm gesetzt werden.

Kopf- und Fußzeilen können mittels `center`, `left` oder `right` positioniert werden.

Wie auch bei Titelzeilen, können Kopf- und Fußzeilen mehrere Zeilen umfassen.

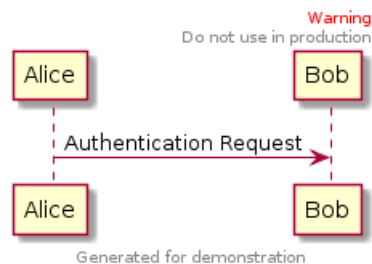
In Kopf- und Fußzeilen können HTML-Elemente verwendet werden.

```
@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 Skalierung

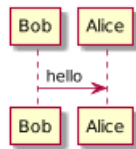
Mit dem Befehl `scale` können generierte Diagramme vergrößert oder verkleinert werden.

Der Skalierungsfaktor kann als Ganzzahl oder als Bruchzahl geschrieben werden. Es kann die Breite oder die Höhe in Pixeln festgelegt werden. Das Diagramm wird dadurch proportional skaliert. Werden sowohl Breite und Höhe angegeben, wird das Diagramm auf die vorgegebene Größe skaliert.

- `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 Titel

Mit `title` kann ein Titel für das Diagramm gesetzt werden. Eine neue Zeile kann mit `\n` in die Überschrift der Bezeichnung eingetragen werden.

Es stehen einige `skinparam` Einstellungen zur Gestaltung des Titels zur Verfügung.

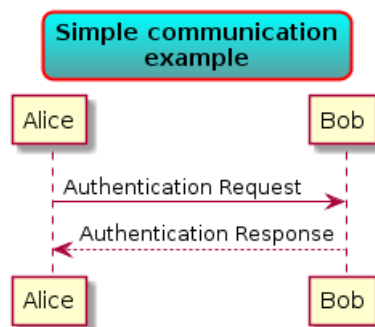
```

@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
  
```



Mit Creole-Markup ist es möglich, die Überschrift des Diagramms zu formatieren.

Mehrzeilige Überschriften können mit den `title` und `end title` Schlüsselwörtern erstellt werden.

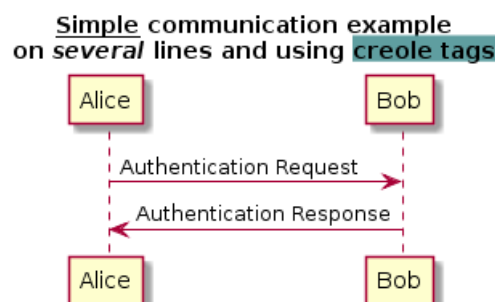
```

@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

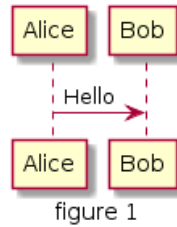
@enduml
  
```



9.5 Beschriftung

Mittels `caption` kann eine Beschriftung zum Diagramm hinzugefügt werden.

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

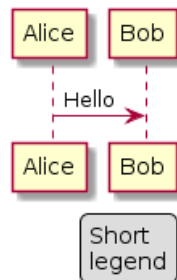


9.6 Diagramm Legende

Mit den Schlüsselwörtern `legend` und `end legend` wird eine Legende angelegt.

Optional können Sie mit `left`, `right` oder `center` die Ausrichtung links, rechts oder mittig festlegen.

```
@startuml
Alice -> Bob : Hello
legend right
Short
legend
endlegend
@enduml
```



10 Salt

Salt ist ein Unterprojekt, das in PlantUML enthalten ist und das beim Entwickeln von graphischen Oberflächen nützlich ist.

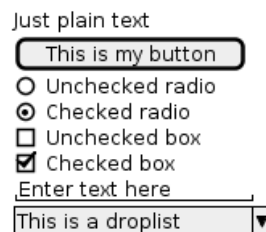
Man kann entweder das Schlüsselwort `@startsalt` oder `@startuml` gefolgt von einer Zeile mit dem Schlüsselwort `salt` verwenden.

10.1 Grundlegendes

Ein Fenster muss mit einer geschweiften Klammer beginnen und enden. Darin kann folgendes definiert werden:

- ein Button mit `[` und `]`.
- Radio-Button mit `(` und `)`.
- eine Checkbox mit `[` und `]`.
- Freitextfeld mit `"`.

```
@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
```



Das Ziel dieses Werkzeugs ist das Darstellung von einfachen und Beispiel-Fenstern.

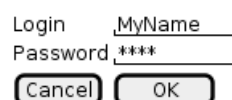
10.2 Nutzung von Gittern

Eine Tabelle wird automatisch erstellt, wenn ein öffnende Klammer `{` benutzt wird.

Zum trennen von Spalten wird `|` verwendet.

Ein Beispiel:

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



Gleich nach der öffnenden Klammer kann durch das erste Zeichen angegeben werden, ob die Linien des Gitters gezeichnet werden sollen:

Um alle senkrechten und waagerechten Linien anzuzeigen:

! Alle senkrechten Linien

- Alle waagerechten Linien

+ Alle äusseren Linien

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

10.3 Verwendung von Trennern

Sie können mehrere horizontale Linien als Trenner verwenden.

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

10.4 Baum Widget (Tree Widget)

Um einen Baum zu erhalten, beginnen Sie mit {T und verwenden + um die Hierarchie Tiefe zu kennzeichnen.

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



```

++++ Berlin
++ Africa
}
}
@endsalt

```



10.5 Enclosing brackets

You can define subelements by opening a new opening bracket.

```

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

```

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

10.6 Adding tabs

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

```

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

```

General
☐ Fullscreen
☐ Behavior
☐ Saving

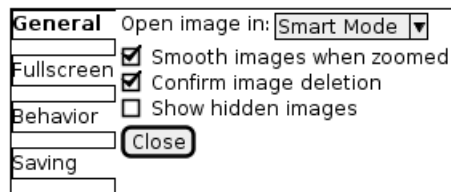
Open image in: Smart Mode ▼
☒ Smooth images when zoomed
☒ Confirm image deletion
☐ Show hidden images
Close

Tab could also be vertically oriented:


```

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

```



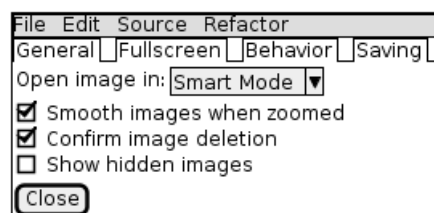
10.7 Using menu

You can add a menu by using {* notation.

```

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

```

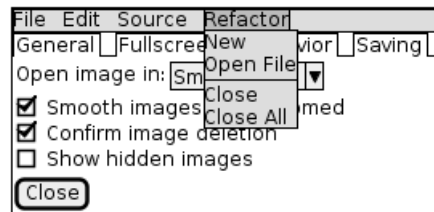


It is also possible to open a menu:

```

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

```



10.8 Advanced table

You can use two special notations for table :

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

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

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

11 Creole

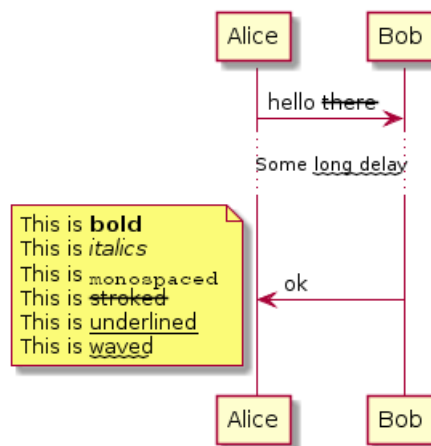
A light Creole engine have been integrated into PlantUML to have a standardized way of defining text style.

All diagrams are now supporting this syntax.

Note that ascending compatibility with HTML syntax is preserved.

11.1 Emphasized text

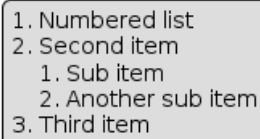
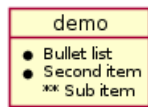
```
@startuml
Alice -> Bob : hello --there--
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
This is bold
This is italics
This is "monospaced"
This is --stroked--
This is __underlined__
This is ~~waved~~
end note
@enduml
```



11.2 List

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

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



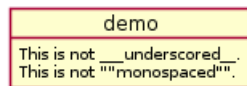
11.3 Escape character

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

```

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

```

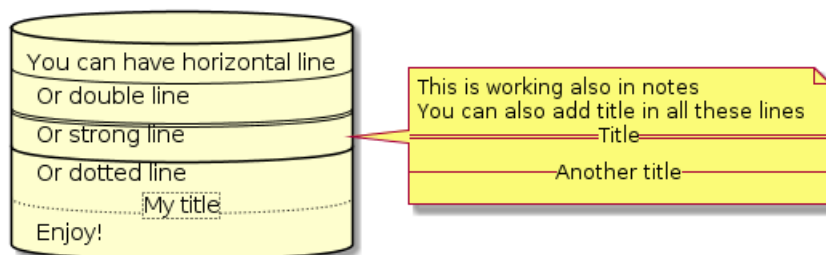


11.4 Horizontal lines

```

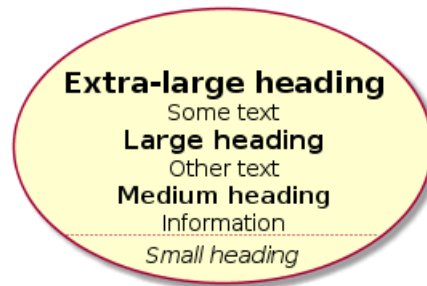
@startuml
database DB1 as "
You can have horizontal line
----
Or double line
====
Or strong line
-----
Or dotted line
..My title..
Enjoy!
"
note right
This is working also in notes
You can also add title in all these lines
==Title==
--Another title--
end note
@enduml

```



11.5 Headings

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

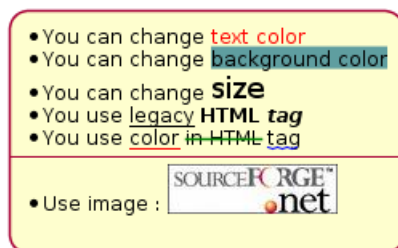


11.6 Legacy HTML

Some HTML tags are also working:

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

```
@startuml
:* You can change <color:red>text color</color>
* You can change <back:cadetblue>background color</back>
* You can change <size:18>size</size>
* You use <u>legacy</u> <b>HTML <i>tag</i></b>
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
----
* Use image : <img:sourceforge.jpg>
;
@enduml
```



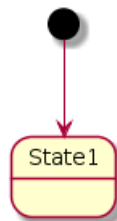
11.7 Tabelle

Es ist möglich Tabellen zu erstellen.

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

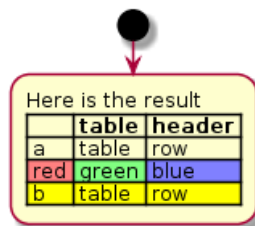
Example of simple table

	table	header
a	table	row
b	table	row



You can specify background colors for cells and lines.

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

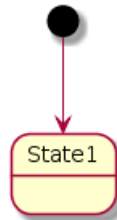
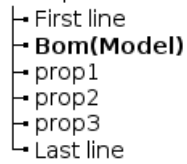


11.8 Tree

You can use |_ characters to build a tree.

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

Example of Tree



11.9 Special characters

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

```

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



11.10 OpenIconic

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

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

```

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

♥Use of OpenIconic♥



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

```

@startuml
listopeniconic
@enduml
  
```

List Open Iconic

Credit to

<https://useiconic.com/open>

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--



11.11 Defining and using sprites

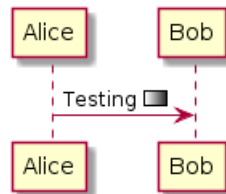
A *Sprite* is a small graphic element that can be used in diagrams.

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

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

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

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



11.12 Encoding Sprite

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

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

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

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

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

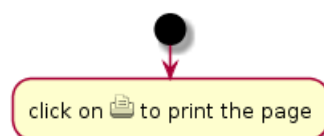
11.13 Importing Sprite

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

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

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

11.14 Examples

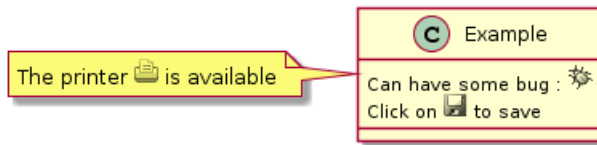


```

@startuml
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPEo
start
:click on <$printer> to print the page;
@enduml

```

Use of sprites (🖨️, ⚙️...)



```

@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p_FEjQEqB1z27aeqCqixa8S40T7C53cKpsHpaYPDJY_12MHM-BLRyywPhrrlw3qu
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPEo
sprite $disk {
444445566677881
436000000009991
43600000000ACA1
53700000001A7A1
537000000012B8A1
538000000123B8A1
63800001233C9A1
634999AABBC99B1
744566778899AB1
7456AAAAA99AAB1
8566AFC228AABB1
8567AC8118BBBB1
867BD4433BBBBB1
39AAAAABBBBBBC1
}

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

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

note left : The printer <$printer> is available

@enduml

```

12 Anpassen von Farben und Schriftarten

12.1 Verwendung

Sie können die Farbe und die Schriftart der Zeichnung verändern in dem Sie den `skinparam` Befehl verwenden. Hier ein Beispiel:

```
skinparam backgroundColor yellow
```

Dieser Befehl kann wie folgt verwendet werden :

- In der Definition des Programms wie jeder andere Befehl auch,
- in einer Include-Datei (siehe *Preprocessing*),
- In einer Konfigurationsdatei, die durch die Kommandozeile oder den ANT-Task übergeben wird.

12.2 Verschachtelung

Um Wiederholungen zu vermeiden ist es möglich, Definitionen zu verschachteln. Siehe die folgende Definition :

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

entspricht exakt der dieser kürzeren Fassung:

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

12.3 Farben

Zur Festlegung der Farbe kann man den Namen der Farbe verwenden oder den RGB Code.

Parameter Name	Standard Wert	Farbe	Beschreibung
backgroundColor	white		Hintergrund der Seite
activityArrowColor	#A80036		Farben von Pfeilen in Aktivitätsdiagrammen
activityBackgroundColor	#FEFECE		Hintergrundfarbe für Aktivitäten
activityBorderColor	#A80036		Randfarbe für die Aktivitäten
activityStartColor	black		Startpunkt in Aktivitätsdiagrammen
activityEndColor	black		Endpunkt ind Aktivitätsdiagrammen
activityBarColor	black		Synchronisierungsblock in Aktivitätsdiagrammen
usecaseArrowColor	#A80036		Farbe der Pfeile in Anwendungsfalldiagrammen
usecaseActorBackgroundColor	#FEFECE		Titelfarbe vom "actor" im usecase Diagramm
usecaseActorBorderColor	#A80036		Rahmenfarbe vom "actor" im usecase Diagramm
usecaseBackgroundColor	#FEFECE		Hintergrund für Anwendungsfälle
usecaseBorderColor	#A80036		Farbe der Rahmen von Anwendungsfällen in Anwendungsfalldiagrammen
classArrowColor	#A80036		Farbe von Pfeilen in Klassendiagrammen
classBackgroundColor	#FEFECE		Hintergrund für Klassen/Schnittstellen/Enums in Klassendiagrammen
classBorderColor	#A80036		Randfarbe von Borders Klassen/Schnittstellen/Enums in Klassendiagrammen
packageBackgroundColor	#FEFECE		Hintergrundfarbe von Paketen in Klassendiagrammen
packageBorderColor	#A80036		Rahmen von Paketen in Klassendiagrammen
stereotypeCBackgroundColor	#ADD1B2		Hintergrundfarbe der Klassenhervorhebung in Klassendiagrammen
stereotypeABackgroundColor	#A9DCDF		Hintergrundfarbe von Hervorhebungen für abstrakte Klassen in Klassendiagrammen
stereotypeIBackgroundColor	#B4A7E5		Hintergrundfarbe für die Schnittstellenmarkierung in Klassendiagrammen
stereotypeEBackgroundColor	#EB937F		Hintergrundfarbe von Enum-Symbolen in Klassendiagrammen
componentArrowColor	#A80036		Farben und Pfeile in Komponentendiagrammen
componentBackgroundColor	#FEFECE		Hintergrundfarbe der Komponenten
componentBorderColor	#A80036		Rahmen von Komponenten
componentInterfaceBackgroundColor	#FEFECE		Hintergrundfarbe von Schnittstellen in Komponentendiagrammen
componentInterfaceBorderColor	#A80036		Randfarbe von Schnittstellen in Komponentendiagrammen
noteBackgroundColor	#FBFB77		Hintergrundfarbe von Notizen
noteBorderColor	#A80036		Rahmenfarbe von Notizen
stateBackgroundColor	#FEFECE		Hintergrundfarbe eines Status im Statusdiagramm
stateBorderColor	#A80036		Randfarbe von Zuständen in Zustandsdiagrammen
stateArrowColor	#A80036		Farbe von Pfeilen in Zustandsdiagrammen
stateStartColor	black		Startpunkt in Zustandsdiagrammen
stateEndColor	black		Kreis als Endzustand in einem state diagrams
sequenceArrowColor	#A80036		Pfeilfarbe in einem sequence diagrams
sequenceActorBackgroundColor	#FEFECE		Farbe des Kopfes eines Akteurs im Sequenzdiagramm
sequenceActorBorderColor	#A80036		Randfarbe eines Akteurs im Sequenzdiagramm
sequenceGroupBackgroundColor	#EEEEEE		Farbe der Überschrift von alt/opt/loop in Sequenzdiagrammen
sequenceLifeLineBackgroundColor	white		Hintergrund der Lebenslinie in Sequenzdiagrammen
sequenceLifeLineBorderColor	#A80036		Rahmen der Lebenslinie in Sequenzdiagrammen
sequenceParticipantBackgroundColor	#FEFECE		Hintergrundfarbe eines Objekts im Sequenzdiagramm
sequenceParticipantBorderColor	#A80036		Rahmenfarbe eines Objektes in einem Sequenzdiagramm



12.4 Schriftfarbe, Name und Größe

Die Schriftart kann mit den `xxxFontColor`, `xxxFontSize` und `xxxFontName` Parametern verändert werden.

Beispiel:

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

Man kann weiterhin die Standardschrift ändern indem man `skinparam defaultFontName` benutzt.

Beispiel:

```
skinparam defaultFontName Aapex
```

Beachten Sie, dass die Namen der Schriftarten sehr stark von dem System abhängig sind, das Sie verwenden. Wenn Sie die Diagramme auf unterschiedlichen Systemen erzeugen lassen wollen, dann sollten Sie die Namen der Schriften mit Bedacht einsetzen und sich vergewissern, dass die Schriften auf allen Systemen zur Verfügung stehen.

Parameter Name	Default Value	Comment
activityFontColor activityFontSize activityFontStyle activityFontName	black 14 plain	Farbe für den Kasten einer Aktivität
activityArrowFontColor activityArrowFontSize activityArrowFontStyle activityArrowFontName	black 13 plain	Wird für den Text an den Pfeilen in den Aktivitätsdiagrammen verwendet
circledCharacterFontColor circledCharacterFontSize circledCharacterFontStyle circledCharacterFontName circledCharacterRadius	black 17 bold Courier 11	Wird in dem Kreis für die Klasse, die Enums und andere verwendet
classArrowFontColor classArrowFontSize classArrowFontStyle classArrowFontName	black 10 plain	Wird für die Pfeile in Klassendiagrammen verwendet
classAttributeFontColor classAttributeFontSize classAttributeIconSize classAttributeFontStyle classAttributeFontName	black 10 10 plain	Klassen Attribute und Methoden
classFontColor classFontSize classFontStyle classFontName	black 12 plain	Wird für den Klassennamen verwendet
classStereotypeFontColor classStereotypeFontSize classStereotypeFontStyle classStereotypeFontName	black 12 italic	Wird für den Stereotyp in Klassen verwendet
componentFontColor componentFontSize componentFontStyle componentFontName	black 14 plain	Wird für den Namen der Komponente verwendet
componentStereotypeFontColor componentStereotypeFontSize componentStereotypeFontStyle componentStereotypeFontName	black 14 italic	Wird für den Stereotyp in Komponentendiagrammen verwendet



componentArrowFontColor componentArrowFontSize componentArrowFontStyle componentArrowFontName	black 13 plain	Textfarbe bei Pfeilen in Komponentendiagrammen
noteFontColor noteFontSize noteFontStyle noteFontName	black 13 plain	Wird für Notizen in allen Diagrammen mit Ausnahme des Sequenzdiagramms verwendet
packageFontColor packageFontSize packageFontStyle packageFontName	black 14 plain	Wird für Paket- und Partitionsnamen verwendet
sequenceActorFontColor sequenceActorFontSize sequenceActorFontStyle sequenceActorFontName	black 13 plain	Wird für den Akteur in den Sequenzdiagrammen verwendet
sequenceDividerFontColor sequenceDividerFontSize sequenceDividerFontStyle sequenceDividerFontName	black 13 bold	Wird für den Text in den Teilern in Sequenzdiagrammen verwendet
sequenceArrowFontColor sequenceArrowFontSize sequenceArrowFontStyle sequenceArrowFontName	black 13 plain	Wird für die Pfeile in Sequenzdiagrammen verwendet
sequenceGroupingFontColor sequenceGroupingFontSize sequenceGroupingFontStyle sequenceGroupingFontName	black 11 plain	Wird als Text für "else" in Sequenzdiagrammen verwendet
sequenceGroupingHeaderFontColor sequenceGroupingHeaderFontSize sequenceGroupingHeaderFontStyle sequenceGroupingHeaderFontName	black 13 plain	Wird für die "alt/opt/loop" Kopfzeilen in Sequenzdiagrammen verwendet
sequenceParticipantFontColor sequenceParticipantFontSize sequenceParticipantFontStyle sequenceParticipantFontName	black 13 plain	Wird für den Text an den Objekten im Sequenzdiagramm verwendet
sequenceTitleFontColor sequenceTitleFontSize sequenceTitleFontStyle sequenceTitleFontName	black 13 plain	Wird für die Überschriften in den Sequenzdiagrammen verwendet
titleFontColor titleFontSize titleFontStyle titleFontName	black 18 plain	Farbe für Überschriften in allen Diagrammen mit Ausnahme des Sequenzdiagramms
stateFontColor stateFontSize stateFontStyle stateFontName	black 14 plain	Schriftfarbe für die Zustände in Zustandsdiagrammen
stateArrowFontColor stateArrowFontSize stateArrowFontStyle stateArrowFontName	black 13 plain	Wird für den Text und die Pfeile im Statusdiagramm verwendet
stateAttributeFontColor stateAttributeFontSize stateAttributeFontStyle stateAttributeFontName	black 12 plain	Schriftfarbe für die Beschreibung der Zustände in Zustandsdiagrammen

usecaseFontColor usecaseFontSize usecaseFontStyle usecaseFontName	black 14 plain	Wird für die Anwendungsfallbeschriftung in den Anwendungsfalldiagrammen verwendet
usecaseStereotypeFontColor usecaseStereotypeFontSize usecaseStereotypeFontStyle usecaseStereotypeFontName	black 14 italic	Wird für die Stereotypen in Anwendungsfällen verwendet
usecaseActorFontColor usecaseActorFontSize usecaseActorFontStyle usecaseActorFontName	black 14 plain	Wird für die Beschriftung der Akteure in den Anwendungsfalldiagrammen verwendet
usecaseActorStereotypeFontColor usecaseActorStereotypeFontSize usecaseActorStereotypeFontStyle usecaseActorStereotypeFontName	black 14 italic	Wird für den Stereotyp beim Akteur verwendet
usecaseArrowFontColor usecaseArrowFontSize usecaseArrowFontStyle usecaseArrowFontName	black 13 plain	Wird für die Textfarbe bei den Pfeilen im Anwendungsfalldiagramm verwendet
footerFontColor footerFontSize footerFontStyle footerFontName	black 10 plain	Farbe der Fußzeile
headerFontColor headerFontSize headerFontStyle headerFontName	black 10 plain	Farbe für die Überschriften

12.5 Schwartzdiagramme

Um schwarzweiße Diagramme zu erzeugen, kann der `skinparam monochrome true` Befehl verwendet werden.

```
@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

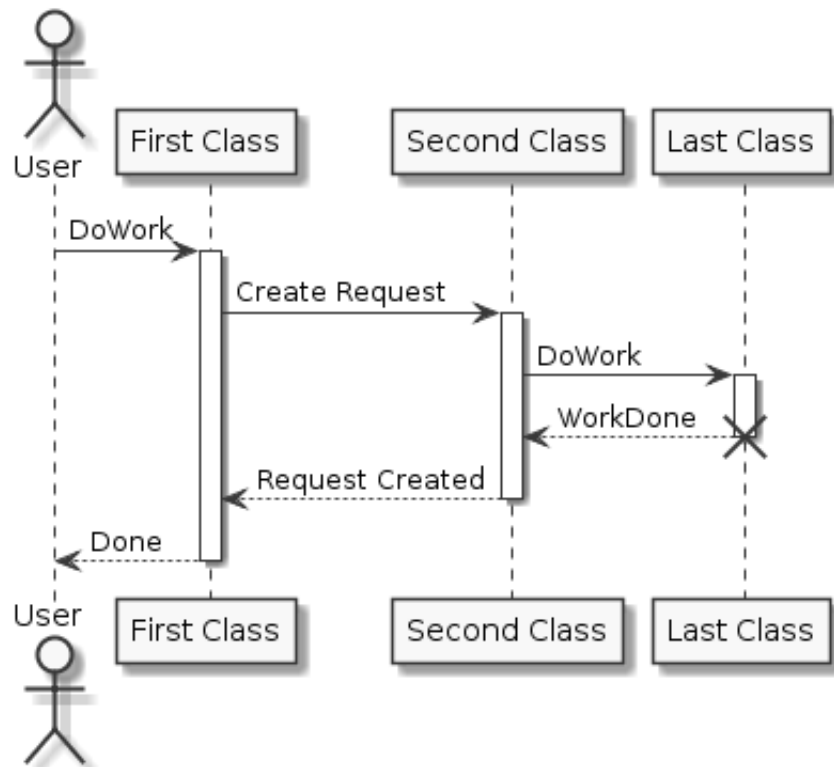
A -> B: Create Request
activate B

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

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



13 Preprocessing

PlantUML bietet auch ein rudimentäres Preprocessing an. Dies kann in allen Diagramme verwendet werden.

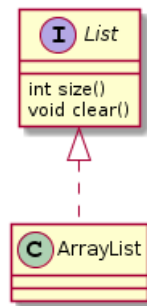
Diese Möglichkeiten ähneln sehr stark dem **C** Präprozessor, mit der Ausnahme, dass anstelle des Schlüsselzeichen **"#"** das Ausrufezeichen **"!"** verwendet wird.

13.1 Dateien einbinden

Verwenden Sie den **!include** Befehl um eine Datei in Ihr Diagramm einzubinden.

Stellen Sie sich vor, Sie verwenden die gleiche Klasse in vielen unterschiedlichen Diagrammen. Anstelle die Beschreibung der Klasse zu duplizieren, können Sie eine Datei erstellen, die diese Beschreibung enthält.

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



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

Die Datei **List.iuml** kann in viele Diagramme eingebunden werden. Jede Veränderung in dieser Datei wirkt sich auf alle Diagramme aus, die diese Datei einbinden.

A file can be only be included once. If you want to include several times the very same file, you have to use the directive **!include_many** instead of **!include**.

Außerdem kann man verschiedene **@startuml/@enduml** in einer eingebundenen Datei definieren und festlegen, welcher dieser Blöcke verwendet werden soll. Dazu hängt man **!0** an den Befehl an wobei **0** die Blocknummer ist.

Wenn man die Datei zum Beispiel mit **!include foo.txt!1** einbindet, dann wird der zweite **@startuml/@enduml** Block aus der Datei **foo.txt** verwendet.

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 URLs einbinden

Mit dem **!includeurl** Befehl kann man eine Datei aus dem Internet oder dem Intranet in sein Diagramm einbinden.

Man kann auch **!includeurl http://someurl.com/mypath!0** verwenden, um festzulegen, welcher **@startuml/@enduml** Block eingebunden werden soll. Die **!0** Notation zeigt auf das erste Diagramm in der Datei.

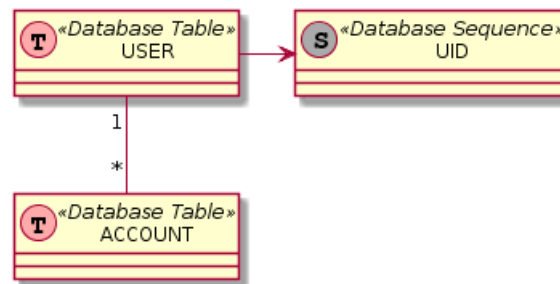


13.3 Konstanten definieren

Konstanten können über die `!define` Direktive festgelegt werden. Wie in der `C` Programmiersprache, kann ein Name für eine Konstante nur alphanumerische Zeichen und den Unterstrich enthalten. Ein Konstantenname darf nicht mit einer Ziffer beginnen.

```
@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
```



Natürlich können Sie die `!include` Direktive auch verwenden, um alle Ihre Konstanten in einer einzigen Datei zu definieren und diese in Ihr Diagramm einbinden.

Konstanten können mit der `!undef XXX` Direktive definiert werden.

Konstanten können gelöscht werden mit `!undef XXX`.

Sie können Konstanten auch innerhalb einer Kommandozeile mit dem `-D` Flag definieren.

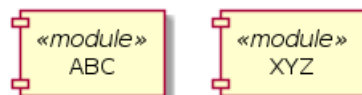
```
java -jar plantuml.jar -DTITLE="My title" atest1.txt
```

Beachten Sie, dass das `-D` Flag nach dem `"-jar plantuml.jar"` Abschnitt gesetzt werden muss.

13.4 Macro Definition

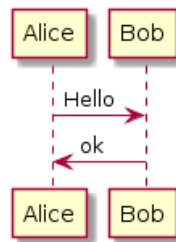
Sie können Macros mit Argumenten definieren.

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



Ein Macro kann mehrere Argumente haben,

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



13.5 Adding date and time

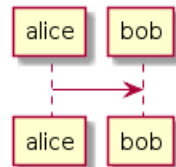
You can also expand current date and time using the special variable `%date%`.

Date format can be specified using format specified in SimpleDataFormat documentation.

```

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

Generated Sun Feb 26 12:15:13 UTC 2017 or 2017.02.26 at 12:15



13.6 Macro mit mehreren Zeilen

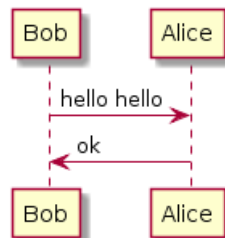
Sie können Macros über mehrere Zeilen definieren. Benutzen Sie dazu `!definelong` am Anfang und `!enddefinelong` am Ende.

```

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

```

AUTHEN(Bob,Alice)
@enduml
  
```



13.7 Default values for macro parameters

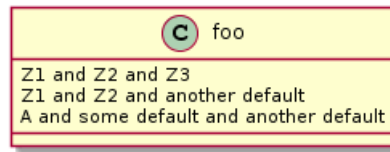
It is possible to assign default values to macro parameters.

```

@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
```



13.8 Bedingungen

Über die Direktiven `!ifdef XXX` und `!endif` können Bedingungen in die Zeichnungen eingefügt werden.

Die Zeilen zwischen den beiden Direktiven werden nur verwendet, wenn die Konstante hinter der `!ifdef` Direktive vorher definiert wurde

Sie können über einen `!else` Teil Zeilen festlegen, die verwendet werden, wenn die Konstante nicht definiert wurde.

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

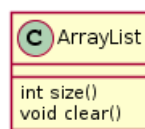


File `ArrayList.iuml`:

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

Mit der `!define` Direktive können optionale Teile des Diagramms eingeschaltet werden.

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



Außerdem kann mit der `!ifndef` Direktive ein Bereich aktiviert werden, wenn die Konstante *NICHT* definiert wurde.

You can use boolean expression with parenthesis, operators `and` `||` in the test.

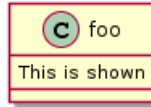
```
@startuml
!define SHOW_FIELDS
!undef SHOW_METHODS
class foo {
!ifdef SHOW_FIELDS || SHOW_METHODS
This is shown
!endif
```



```

#ifdef SHOW_FIELDS && SHOW_METHODS
This is NOT shown
#endif
}
@enduml

```



13.9 Suchpfad

Definieren Sie die Java Eigenschaft "plantuml.include.path" in der Kommandozeile

Zum Beispiel:

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

Achten Sie darauf, dass die -D Option vor der -jar Option steht. -D Optionen, die nach der -jar Option gesetzt werden, werden benutzt um Konstanten für den plantuml Präprozessors zu definieren.

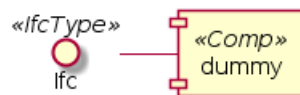
13.10 Fortgeschrittene Merkmale

Sie können Text an ein Macro Argument mit den ## Zeichen anhängen.

```

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

```

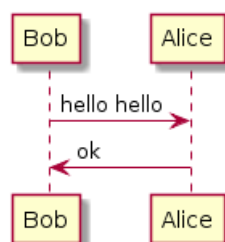


A kann durch ein weiteres Makro definiert werden.

```

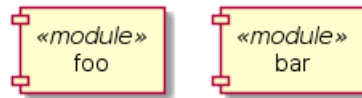
@startuml
!define DOUBLE(x) x x
!definelong AUTHEN(x,y)
x -> y : DOUBLE(hello)
y -> x : ok
!enddefinelong
AUTHEN(Bob,Alice)
@enduml

```



A Macro mit selben Namen kann mehrfach definiert werden, wenn Sie eine unterschiedliche Anzahl an Argumenten definieren.

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



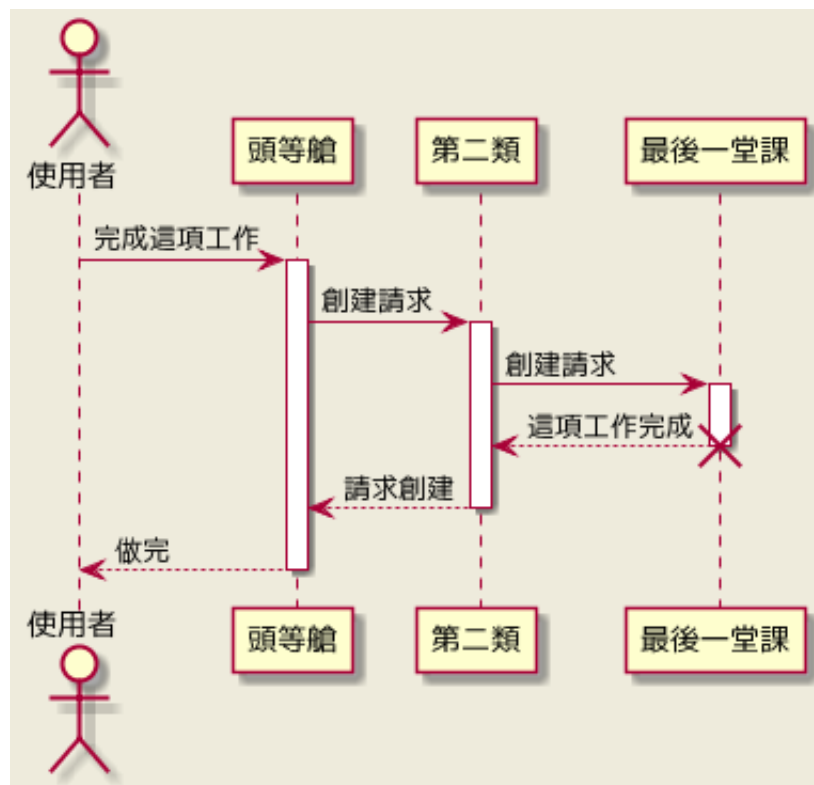
Sie können für ein include auch System Umgebungsvariablen oder Konstanten verwenden:

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

14 Übersetzung

Die PlantUML Sprache verwendet *Buchstaben* um Akteure, Anwendungsfälle und so weiter zu definieren. Aber mit *Buchstaben* sind nicht nur die lateinischen Buchstaben A-Z gemeint, es kann *jede Art von Zeichen in jedweder Sprache verwendet werden*.

```
@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 Zeichensatz

Der voreingestellte Zeichensatz beim Lesen der Textdateien mit den UML Textbeschreibungen ist abhängig vom System. Normalerweise sollte es hier keine Probleme geben. Aber es kann Situationen geben, in denen Sie einen anderen Zeichensatz verwenden wollen. Dies lässt sich zum Beispiel über die Kommandozeile erreichen:



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

Oder mit dem Ant-Task:

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

Abhängig von Ihrer Java-Installation sind die folgenden Zeichensätze verfügbar: ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.

15 Farbnamen

Hier ist eine Liste PlantUML bekannter Farben. Beachten Sie, dass bei Farbnamen die Groß-/Kleinschreibung relevant ist.

	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	Sequenz-Diagramm	1
1.1	Grundlagen	1
1.2	Deklaration eines Teilnehmers	1
1.3	Verwendung von nicht-alphanumerischen Zeichen	2
1.4	Nachrichten an sich selbst	3
1.5	Ändern der Pfeilart	3
1.6	Ändern der Pfeil Farbe	4
1.7	Nummerierung der Nachrichtenreihenfolge	4
1.8	Aufteilung von Diagrammen	6
1.9	Gruppierung von Nachrichten	7
1.10	Notizen	8
1.11	Weitere Möglichkeiten für Notizen	8
1.12	Ändern der Form von Notizen	9
1.13	Creole und HTML	10
1.14	Diagramme aufteilen	11
1.15	Referenz	11
1.16	Verzögerungen	12
1.17	Abstände	12
1.18	Aktivierung und Deaktivierung der Lebenslinie	13
1.19	Erstellung von Teilnehmern	14
1.20	Eingehende und ausgehende Nachrichten	15
1.21	Stereotypen	16
1.22	Mehr Information zu Überschriften	17
1.23	Anpassungen bei den Teilnehmern	18
1.24	Fußzeile entfernen	19
1.25	Der Skinparam Befehl	19
1.26	Changing padding	21
2	Anwendungsfall-Diagramm	22
2.1	Anwendungsfälle	22
2.2	Akteure	22
2.3	Beschreibung der Anwendungsfälle	22
2.4	Einfaches Beispiel	23
2.5	Erweiterungen / Generalisierungen	24
2.6	Verwenden von Notizen	24
2.7	Stereotypen	25
2.8	Ändern der Pfeilrichtungen	25
2.9	Aufteilen von Diagrammen auf mehrere Seiten	27
2.10	Verändern der Richtung in der die Objekte angeordnet werden	27
2.11	Der Skinparam-Befehl	28
2.12	Vollständiges Beispiel	29

3	Klassendiagramm	30
3.1	Beziehungen zwischen Klassen	30
3.2	Beschriften von Beziehungen	31
3.3	Methoden hinzufügen	32
3.4	Sichtbarkeit festlegen	33
3.5	Abstract und Static	34
3.6	Der Klassenrumpf für Fortgeschrittene	35
3.7	Notizen und Stereotypen	36
3.8	Mehr zu Notizen	37
3.9	Notizen zu Beziehungen	38
3.10	Abstrakte Klassen und Interfaces	39
3.11	Verwendung von Sonderzeichen	40
3.12	Verstecken von Attributen, Methoden	41
3.13	Verstecken von Klassen	42
3.14	Verwenden von Generics	42
3.15	Besondere Hervorhebungen	42
3.16	Pakete	43
3.17	Paketarten	43
3.18	Namensraum	44
3.19	Automatische Erzeugung eines Namensraums	45
3.20	Lollipop Schnittstellen	46
3.21	Ändern der Pfeilrichtung	46
3.22	Assoziationsklassen	47
3.23	Der Skinparam-Befehl	48
3.24	Das Aussehen von Stereotypen verändern	49
3.25	Farbverlauf	49
3.26	Hilfe beim Layout	50
3.27	Große Dateien aufteilen	51
4	Aktivitätsdiagramm	53
4.1	Einfache Aktivität	53
4.2	Beschriftungen an Pfeilen	53
4.3	Pfeilrichtung ändern	53
4.4	Verzweigungen	54
4.5	Mehr über Verzweigungen	55
4.6	Synchronisation	56
4.7	Lange Beschreibungen für Aktivitäten	57
4.8	Notizen	57
4.9	Partitionen	58
4.10	Der Skinparam Befehl	59
4.11	Oktagon	60
4.12	Komplettes Beispiel	60

5	Aktivitätsdiagramm (Beta)	63
5.1	Einfache Aktivität	63
5.2	Start Stop	63
5.3	Bedingung	64
5.4	Repeat-Schleife	65
5.5	While-Schleife	65
5.6	Parallele Verarbeitung	66
5.7	Notizen	67
5.8	Farben	67
5.9	Pfeile	68
5.10	Gruppierung	68
5.11	Schwimmbahnen	69
5.12	Abtrennen	70
5.13	SDL-Diagramme	71
5.14	Komplettes Beispiel	72
6	Komponentendiagramm	74
6.1	Komponenten	74
6.2	Schnittstellen	74
6.3	Beispiel	74
6.4	Notizen	75
6.5	Gruppierende Komponenten	75
6.6	Ändern der Pfeilrichtung	77
6.7	UML2-Notation verwenden	78
6.8	Mehrzeilige Beschreibung	78
6.9	Individuelle Farben	78
6.10	Verwendung von Sprites in Stereotypen	78
6.11	Der Skinparam Befehl	79
7	Zustandsdiagramme	81
7.1	Einfache Zustandsdiagramme	81
7.2	Verschachtelter Zustand	81
7.3	Lange Bezeichnungen für einen Zustand	82
7.4	Konkurrierende Zustände	83
7.5	Pfeilrichtung	84
7.6	Notizen	85
7.7	Mehr über Notizen	86
7.8	Skinparam	86
8	Objektdiagramm	88
8.1	Definition von Objekten	88
8.2	Beziehungen zwischen Objekten	88
8.3	Hinzufügen von Attributen	88
8.4	Gemeinsam mit klassendiagrammen verwendete Funktionen	89

9 Allgemeine Befehle	90
9.1 Kommentare	90
9.2 Kopf- und Fusszeile	90
9.3 Skalierung	90
9.4 Titel	91
9.5 Beschriftung	92
9.6 Diagramm Legende	92
10 Salt	93
10.1 Grundlegendes	93
10.2 Nutzung von Gittern	93
10.3 Verwendung von Trennern	94
10.4 Baum Widget (Tree Widget)	94
10.5 Enclosing brackets	95
10.6 Adding tabs	95
10.7 Using menu	96
10.8 Advanced table	97
11 Creole	98
11.1 Emphasized text	98
11.2 List	98
11.3 Escape character	99
11.4 Horizontal lines	99
11.5 Headings	100
11.6 Legacy HTML	100
11.7 Tabelle	101
11.8 Tree	101
11.9 Special characters	102
11.10 OpenIconic	102
11.11 Defining and using sprites	104
11.12 Encoding Sprite	104
11.13 Importing Sprite	104
11.14 Examples	104
12 Anpassen von Farben und Schriftarten	106
12.1 Verwendung	106
12.2 Verschachtelung	106
12.3 Farben	107
12.4 Schriftfarbe, Name und Größe	108
12.5 Schwarzzeildiagramme	111

13 Preprocessing	112
13.1 Dateien einbinden	112
13.2 URLs einbinden	112
13.3 Konstanten definieren	113
13.4 Kacro Definition	113
13.5 Adding date and time	114
13.6 Macro mit mehreren Zeilen	114
13.7 Default values for macro parameters	114
13.8 Bedingungen	115
13.9 Suchpfad	116
13.10 Fortgeschrittene Merkmale	116
14 Übersetzung	118
14.1 Zeichensatz	118
15 Farbnamen	120