




3 Klassendiagramm

3.1 Beziehungen zwischen Klassen

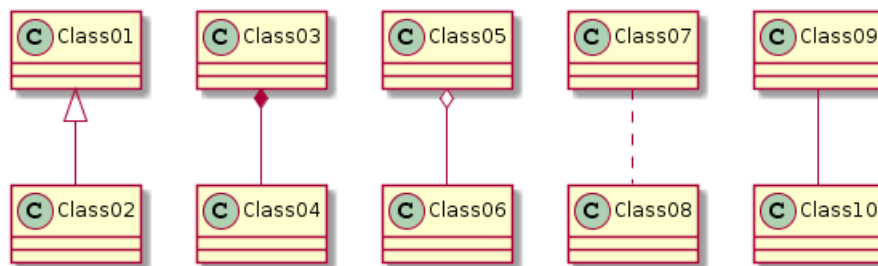
Beziehungen zwischen Klassen werden mit den folgenden Symbolen gekennzeichnet:

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

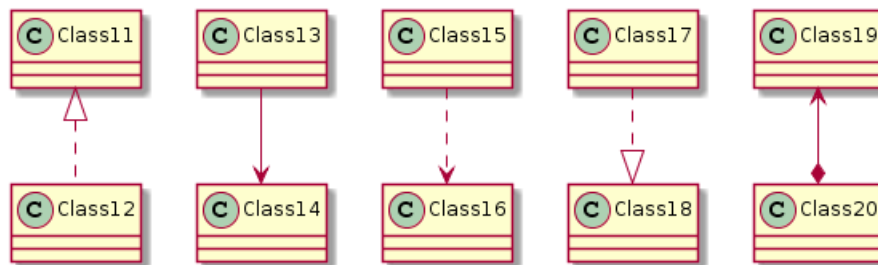
Es ist möglich "--" durch ".." zu ersetzen, um eine gepunktete Linie zu erhalten.

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

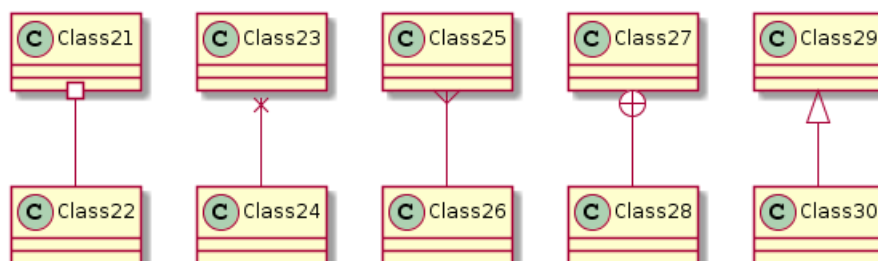
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```



```
@startuml
Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20 not implemented (only basic ones)
@enduml
```



```
@startuml
Class21 #-- Class22
Class23 x-- Class24
Class25 }-- Class26
Class27 +-- Class28
Class29 ^-- Class30 same semantics as generalization??
@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.

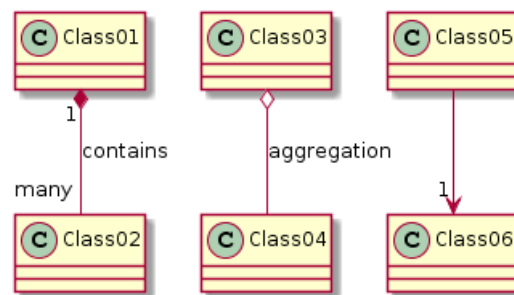
```
@startuml
```

```
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
```

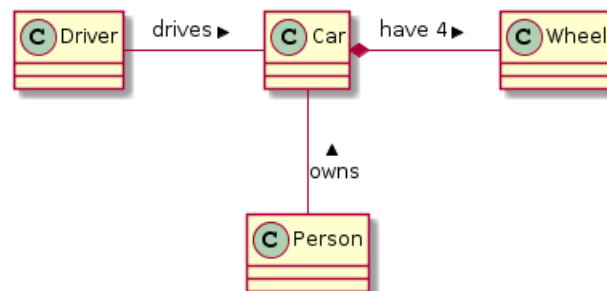
```
class Car
```

```
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.

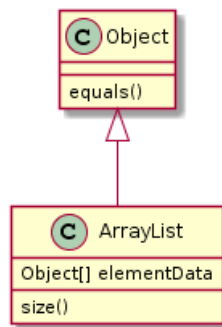
Das System erkennt anhand der Klammern, ob es sich um eine Methode oder um ein Feld handelt.

```
@startuml
Object <|-- ArrayList

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

@enduml
```

uml displays types after names ...
implemented: elementData Object[]



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()
}
```

not supported, syntax: [name] : [type]

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

@enduml
```

gilt das auch für ohne {} obj syntax ??
impl: ja



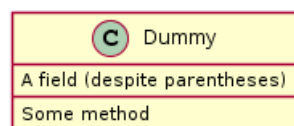
Sie können die Modifier {field} und {method} verwenden, um das Standardverhalten des Parsers bei Feldern und Methoden zu übersteuern.

```
@startuml
class Dummy {
{field} A field (despite parentheses)
{method} Some method
}

@enduml
```

NOT SUPPORTED
one can use {field} and {method}
but this does not change anything!

else we would lose types...



3.4 Sichtbarkeit festlegen

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

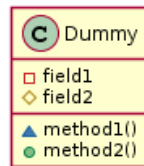
-	□	private
#	◇	protected
~	△	package private
+	○	public

```
@startuml
```

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

```
@enduml
```

where do one places {field}, {method} ??
impl: after modifiers



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

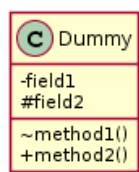
```
@startuml
```

```
skinparam classAttributeIconSize 0
```

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

```
@enduml
```

not implemented, use css



3.5 Abstract und Static

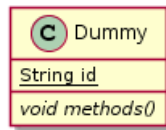
Sie können statische oder abstrakte Methoden und statische Attribute durch benutzen des **static** oder **abstract** Modifikators definieren.

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
```

should this work with {field}{method}?
or only one modifier accepted??
impl: {field}{method} AND modifiers static, abstract

impl allowed after
visibility modifiers
e.g.
+{static}{method} foo



3.6 Der Klassenrumpf für Fortgeschrittene

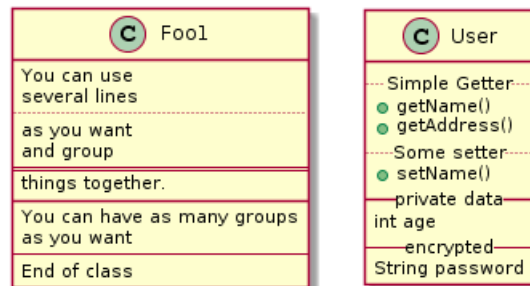
Standardmäßig werden die Methoden und Felder im Klassenrumpf automatisch von PlantUML gruppiert. Mit Hilfe von Trennzeichen können Felder und Methoden aber auch selber geordnet werden. Folgende Trennzeichen sind möglich: `--` (einfache durchgezogene Linie), `..` (einfache unterbrochene Linie), `==` (doppelte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

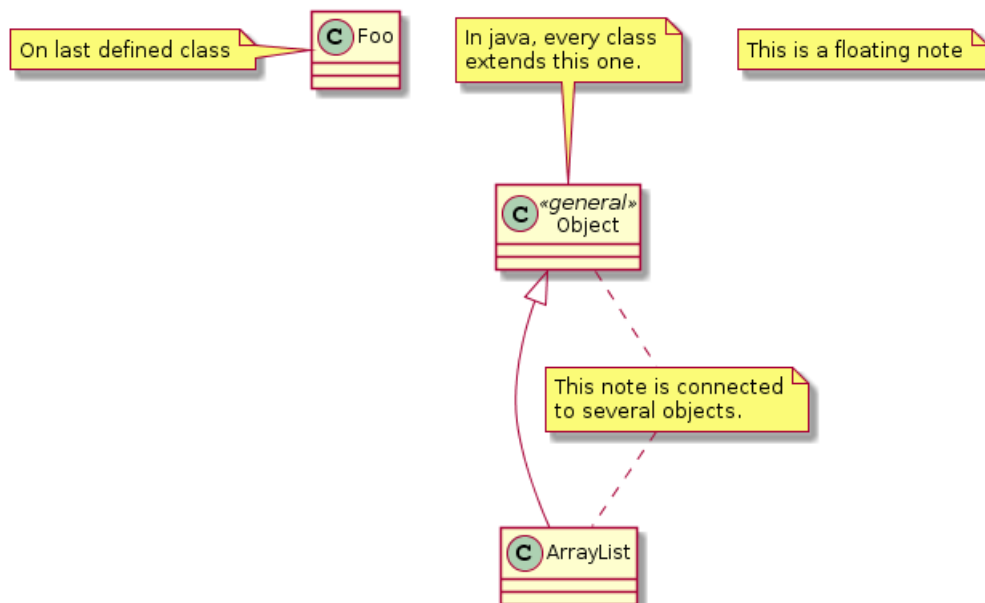
@enduml
```

note text needs to be enclosed in „
EVERY TIME!!
this also makes parsing multiple
lines easier (or possible?)

impl: note name not needed

needs type check

string should be enclosed in „...“



3.8 Mehr zu Notizen

not supported,
part of the rendere ... we assume just some string

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.

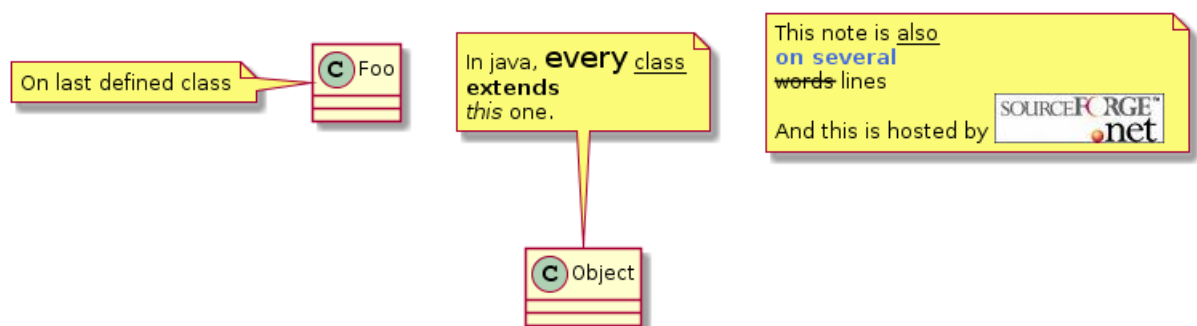
```
@startuml

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
```

enclosed in „...“ then it's ok

```
Dummy --> Foo2 : Another link
```

```
note right on link #blue
```

```
this is my note on right link
```

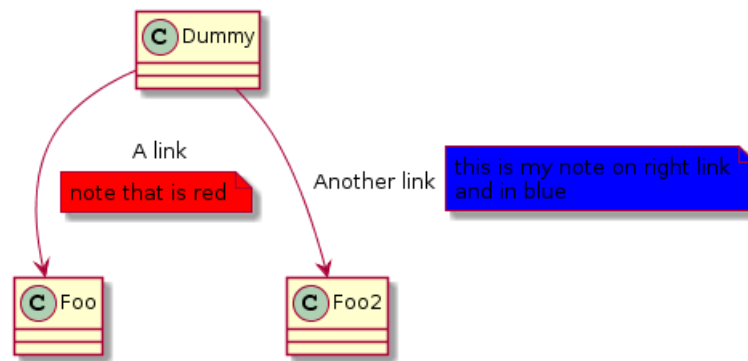
```
and in blue
```

```
end note
```

multiple allowed on same relation?
impl: yes

color not supporte

```
@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
```

interface stereotype
impl: allowed

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

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

interface attributes allowed?

impl: no

visibility modifier?

impl: no (because all are public)

{field}, {method}?

impl: no

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

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

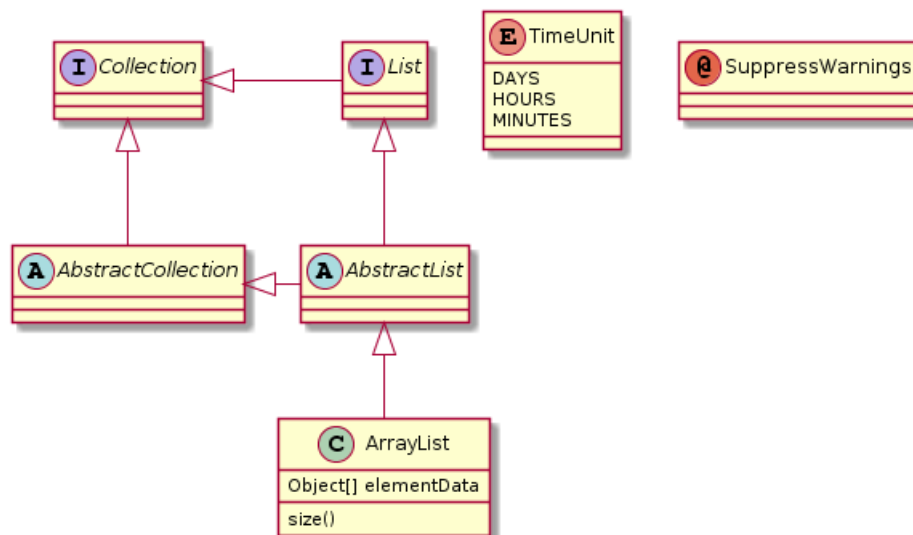
enum impl:

allow DAYS = 0 default values

stereotypes: true (e.g. to determine string enum or int...)

```
annotation SuppressWarnings
```

```
@enduml
```



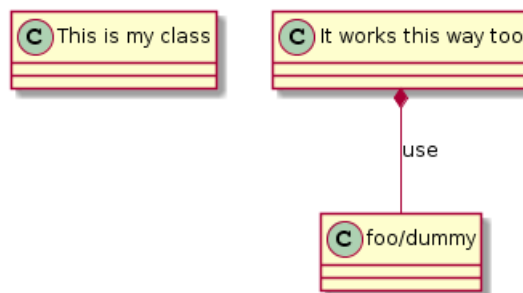
3.11 Verwendung von Sonderzeichen

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()
}
```

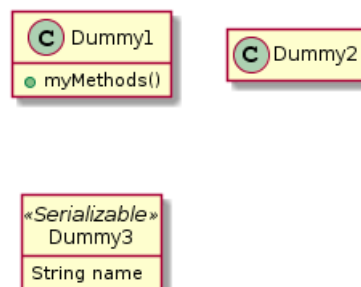
```
class Dummy2 {
+hiddenMethod()
}
```

```
class Dummy3 <<Serializable>> {
String name
}
```

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

```
@enduml
```

not supported,
renderer can do this
or some other ui



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.

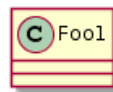
```
@startuml
class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```

not supported,
render/ui can do this



3.14 Verwenden von Generics

interaction with stereotype, (re)naming...??

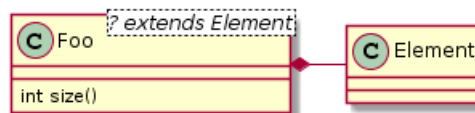
Mit spitzen Klammern (< und >) kann die Verwendung von Generics dargestellt werden.

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

@enduml
```

auch erlaubt bei enum/interface?
impl: interface yes, enum no

generics in methods??
impl: yes



Man kann diese Darstellung mittels des Befehls `skinparam genericDisplay old` ausschalten.

3.15 Besondere Hervorhebungen

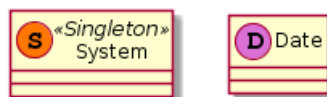
Normalerweise werden Klassen, Schnittstellen, Enums und abstrakte Klassen mit einem hervorgehobenen Buchstaben gekennzeichnet (C, I, E or A).

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
```

not supported



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.

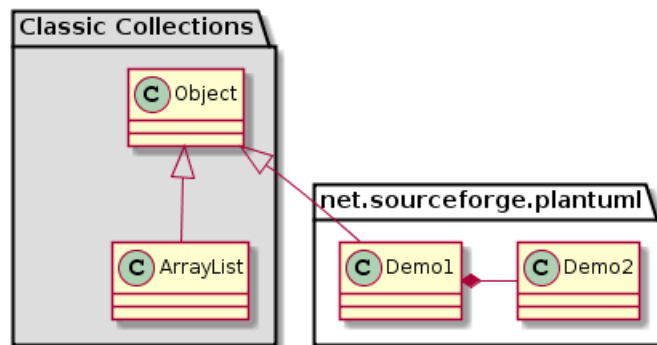
empty packages?
do we need the {} then??

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

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

@enduml
```

only identifier as package name supported,
separated by ::



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
```

impl: general stereotypes not only these





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

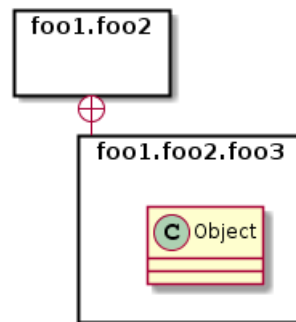
```
@startuml
skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Namensraum

use package... not implemented because
what is the difference between package and namespace??

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 voll qualifizierten 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
}
```

separator is :: because that's implemented in c++, c#, ruby, ...

also better because . is already a horizontal line



```

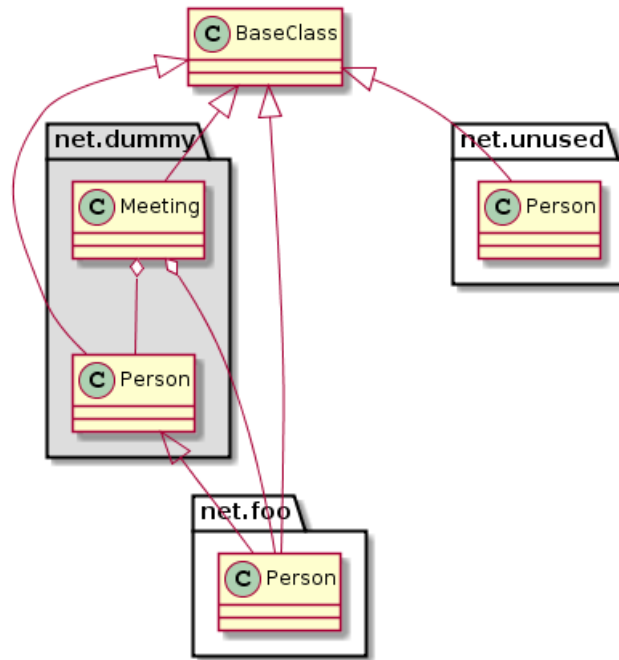
.BaseClass <|-- Person

net.dummy.Meeting o-- Person
}

.BaseClass <|-- net.unused.Person

@enduml

```



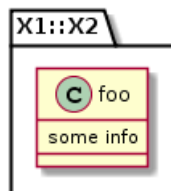
3.19 Automatische Erzeugung eines Namensraums

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

```

@startuml
                                not supported
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 {

```

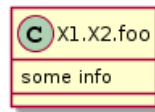


```

some info
}

@enduml

```



3.20 Lollipop Schnittstellen

Mit der folgenden Syntax kann man Schnittstellen von Klassen definieren:

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

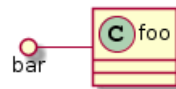
TODO

```

@startuml
class foo
bar ()- foo
@enduml

```

source must be a class,
target must be an interface



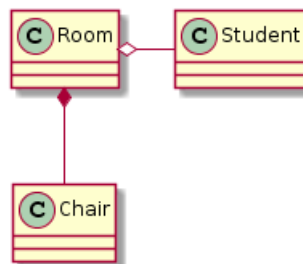
3.21 Ändern der Pfeilrichtung

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

```

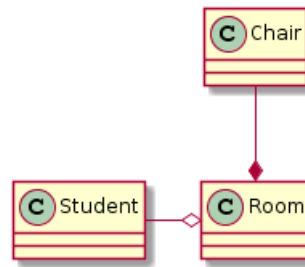


Man kann die Richtung auch durch das Umdrehen der Verbindung ändern:

```

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

```



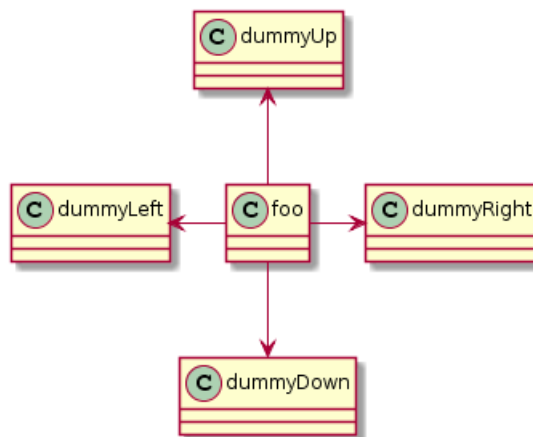
Außerdem ist es möglich, die Richtung der Pfeile durch Hinzufügen der `left`, `right`, `up` oder `down` Schlüsselwörter innerhalb der Pfeile zu verändern:

```

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

```

not supported



not supported (1 character encode)

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

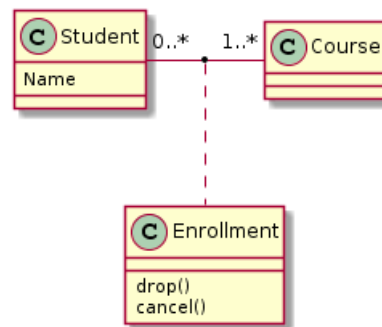
Nach dem man eine Beziehung zwischen zwei Klassen definiert hat, kann man eine *association class* definieren. Hierzu ein Beispiel:

```

@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

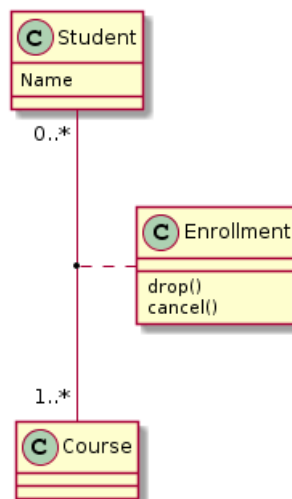
class Enrollment {
drop()
cancel()
}
@enduml

```

use .. because we might use . elsewhere

which package for association class??

impl: all association classes are in default package



3.23 Der Skinparam-Befehl not supported (all below too)

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
}

```



```

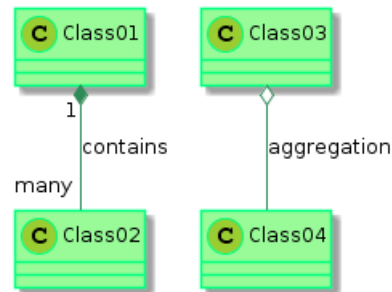
skinparam stereotypeCBackgroundColor YellowGreen

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
}

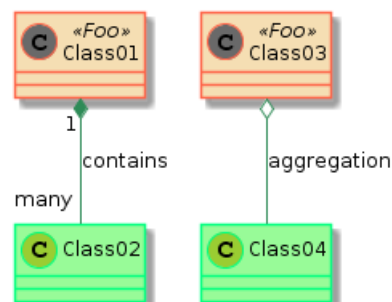
skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

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.

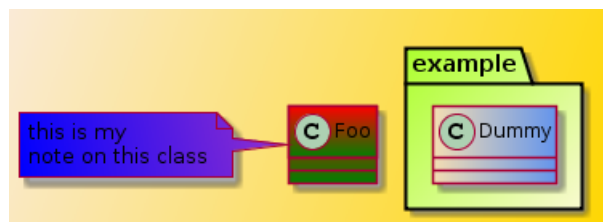
So könnte dies zum Beispiel aussehen:

```
@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

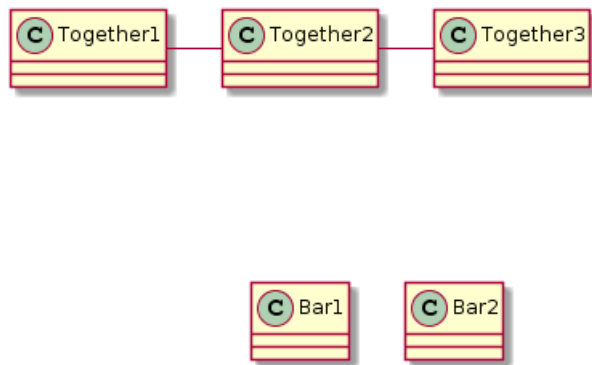
Sometimes, the default layout is not perfect...

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.

Die Verwendung von `skinparam` Definitionen, ermöglicht die Darstellung von Außenrahmen für mehrseitige Bilder. (Siehe nachfolgendes Beispiel)

```

@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

```

