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

- Sequence diagram,
- Usecase diagram,
- Class diagram,
- Activity diagram,
- Component diagram,
- State diagram,
- Object diagram.

Diagrams are defined using a simple and intuitive language.
1 Sequence Diagram

1.1 Basic examples

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

The sequence "->" is used to draw a message between two participants.

Participants do not have to be explicitly declared.

To have a dotted arrow, you use "-->".

It is also possible to use "<--" and "<--". That does not change the drawing, but may improve readability.

Example:

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

To use asynchronous message, you can use "->>" or "<<-".

```
@startuml
Alice -> Bob: synchronous call
Alice -->> Bob: asynchronous call
@enduml
```
1.2 Declaring participant

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

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

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

You can also change the background color of actor or participant, using html code or color name.

Everything that starts with simple quote ' is a comment.

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

Alice->Bob: Authentication Request
Bob->Alice: Authentication Response
Bob->L: Log transaction
@enduml
```
1.3 Use non-letters in participants

You can use quotes to define participants. And you can use the as keyword to give an alias to those participants.

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

1.4 Message to Self

A participant can send a message to itself.

It is also possible to have multilines using \n.

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

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

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

You can specify

- a startnumber with "autonumber 'start'",
- an increment with "autonumber 'start' 'increment'"

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

You can also use some html tags in the format.

```plantuml
@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
```
1.6 Title

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

```plantuml
@startuml
title Simple communication example
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
@enduml```

**Simple communication example**

![Diagram of simple communication example between Alice and Bob]

- Alice
- Bob
- Authentication Request
- Authentication Response
1.7 Splitting diagrams

The `newpage` keyword is used to split a diagram into several images. You can put a title for the new page just after the `newpage` keyword. This is very handy to print long diagram on several pages.

```
@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
```
1.8 Grouping message

It is possible to group messages together using the following keywords:

- alt/else
- opt
- loop
- par
- break
- critical
- group, followed by a text to be displayed

It is possible to add a text that will be displayed into the header. The `end` keyword is used to close the group. Note that it is possible to nest groups.

```plaintext
@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
```
1.9 Notes on messages

It is possible to put notes on message using:

- **note left** or
- **note right** keywords just after the message.

You can have multiline note using the **end note** keyword.

```plantuml
@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
```
1.10 Some other notes

It is also possible to place notes relative to participant with:

- **note left of**,  
- **note right of** or  
- **note over** keywords.

It is possible to highlight a note by changing its background color. You can also have multilines note using the **end note** keywords.

```plantuml
@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
```

![Sequence Diagram](image)
1.11 Formatting using HTML

It is also possible to use few html tags like:

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

```plantuml
@startuml
participant Alice
participant "The <b>Famous</b> Bob" as Bob
Alice -> Bob : A <i>well formated</i> message
note right of Alice
This is `<size:18>displayed</size>`<u>left of</u> Alice.
end note
note left of Bob
This is `<color:#118888>displayed</color>`<b><color:purple>left of</color> <strike>Alice</strike> Bob</b>.
end note
note over Alice, Bob
This is hosted by `<img:sourceforge.jpg>`
end note
@enduml
```
1.12 Divider

If you want, you can split a diagram using "==" separator to divide your diagram into logical steps.

```plantuml
== Initialization ==
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

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

![Sequence Diagram](image)
1.13 Lifeline Activation and Destruction

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

Once a participant is activated, its lifeline appears.

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

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

```plantuml
@startuml
participant User

User -> A: DoWork
activate A

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

B -> C: DoWork
activate C

C --> B: WorkDone
destroy C

B --> A: RequestCreated
deactivate B

A -> User: Done
deactivate A

@enduml
```
Nested lifeline can be used, and it is possible to add a color on the lifeline.

```plantuml
@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
```
1.14 Participant creation

You can use the `create` keyword just before the first reception of a message to emphasize the fact that this message is actually creating this new object.

```
@startuml
Bob -> Alice : hello
create Other
Alice -> Other : new
create String
Alice -> String
note right : You can also put notes!
Alice --> Bob : ok
@enduml
```
1.15 Incoming and outgoing messages

You can use incoming or outgoing arrows if you want to focus on a part of the diagram.

Use square brackets to denotate the left "[" or the right "]" side of the diagram.

@startuml
[-> A: DoWork
activate A
A -> A: Internal call
activate A
A ->] : << createRequest >>
A<--] : RequestCreated
deactivate A
[<- A: Done
deactivate A
@enduml
1.16 Stereotypes and Spots

It is possible to add stereotypes to participants using "<<" and ">>". In the stereotype, you can add a spotted character in a colored circle using the syntax "(X,color)".

```plantuml
@startuml
participant "Famous Bob" as Bob <<< Generated >>>
participant Alice <<< (C,#ADD1B2) Testable >>> #EEEEEE
Bob->Alice: First message
@enduml
```

```plantuml
@startuml
participant Bob <<< (C,#ADD1B2) >>>
participant Alice <<< (C,#ADD1B2) >>>
Bob->Alice: First message
@enduml
```
1.17 More information on titles

You can use some HTML tags in the title.

```plantuml
@startuml
title <u>Simple</u> communication example
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
@enduml
```

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

```plantuml
@startuml
title <u>Simple</u> communication example\non several lines
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
@enduml
```

```
Simple communication example

Alice

Authentication Request

Authentication Response

Bob

Simple communication example
on several lines

Alice

Authentication Request

Authentication Response

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

```plantuml
@startuml

<title>
 Simple communication example
 on several lines and using html
 This is hosted by SourceForge.jpg
</title>

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

@enduml
```
1.18 Participants englober

It is possible to draw a box around some participants, using `box` and `end box` commands.

You can add an optional title or a optional background color, after the `box` keyword.

```
@startuml
box "Internal Service" #LightBlue
    participant Bob
    participant Alice
end box
participant Other

Bob -> Alice : hello
Alice -> Other : hello
@enduml
```
1.19 Removing Footer

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

```plantuml
hide footbox
title Footer removed

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

Footer removed

Alice

Authentication Request

Authentication Response

Bob
1.20 Skinparam

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

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

```plantuml
@startuml
skinparam backgroundColor #EEE8DC
skinparam sequenceArrowColor DeepSkyBlue
skinparam sequenceParticipantBorderColor DeepSkyBlue
skinparam sequenceLifeLineBorderColor blue
skinparam sequenceParticipantBackgroundColor DodgerBlue
skinparam sequenceParticipantFontName Impact
skinparam sequenceParticipantFontSize 17
skinparam sequenceParticipantFontColor #A9DCDF
skinparam sequenceActorBackgroundColor aqua
skinparam sequenceActorFontColor DeepSkyBlue
skinparam sequenceActorFontSize 17
skinparam sequenceActorFontName Aapex
skinparam sequenceLifeLineBackgroundColor #A9DCDF

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
```
1.21 Skin

Use the keyword `skin` to change the look of the generated diagram. There are only two skins available today (Rose, which is the default, and BlueModern), but it is possible to write your own skin.

```plantuml
@startuml
skin BlueModern
actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C
User -> A: DoWork
activate A
A -> B: << createRequest >>
activate B
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C
B --> A: Request <u>Created</u>
deactivate B
A --> User: Done
deactivate A
@enduml
```
2 Use Case Diagram

2.1 Usecases

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

You can also use the `usecase` keyword to define a usecase.

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

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

Actor are enclosed using between two points.

You can also use the `actor` keyword to define an actor.

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

We will see than actor definitions is optional.

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

First Actor  Another actor  Men3  Last actor
2.3 Basic example

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

The more dashes "-" in the arrow, the longer the arrow.

You can add a label on the arrow, by adding a ":" character in the arrow definition.

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

@startuml
User --> (Start)
User --> (Use the application) : A small label
:Main Admin: --> (Use the application) : This is\nyet another\nlabel
@enduml
2.4 Extension

If one actor/use case extends another one, you can use the symbol `<|--` (which stands for `←`).

As for smiley, when you turn your head, you will see the symbol ↑

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

User `<|-- Admin
(Start) `<|-- (Use)
@enduml
2.5 Using notes

You can use the:

- note left of,
- note right of,
- note top of,
- note bottom of

keywords to define notes related to a single object. A note can be also define alone with the note keywords, then linked to other objects using the .. symbol.

```plantuml
@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
```

![PlantUML Diagram](image-url)
2.6 Stereotypes

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

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

Diagram:

- **<< Application >>**
- **<< Human >>**
- **<< One Shot >>**
- Main Database
- User
- **<< Main >>**
- Use the application

Diagram created with PlantUML.
2.7 Changing arrows direction

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

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

You can also change directions by reversing the link:

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

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

```plantuml
@startuml
:user: -left-> (dummyLeft)
:user: -right-> (dummyRight)
:user: -up-> (dummyUp)
:user: -down-> (dummyDown)
@enduml
```
2.8 Title the diagram

The title keywords is used to put a title.

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

```plantuml
@startuml
title Simple <b>Usecase</b> 
with one actor

usecase (Use the application) as (Use)
User -> (Use)
@enduml
```

Simple Usecase
with one actor

User --> Use the application
2.9 Left to right direction

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

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

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

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

You can use the **skinparam** command to change colors and fonts for the drawing. You can use this command:

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

```plantuml
@startuml
skinparam usecaseBackgroundColor DarkSeaGreen
skinparam usecaseArrowColor Olive
skinparam actorBorderColor black
skinparam usecaseBorderColor DarkSlateGray
skinparam usecaseActorFontName Courier

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

User -> (Start)
User --> (Use)
MySql --> (Use)
@enduml
```
3 Class Diagram

3.1 Relations between classes

Relations between classes are defined using the following symbols:

| Extension | <|-- |
| Composition | *-- |
| Aggregation | o-- |

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

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

```plantuml
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
Class11 <.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..> Class18
Class19 <--- Class20
@enduml
```
3.2 Label on relations

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

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

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

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

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

@startuml
Object <|-- ArrayList

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

It is also possible to group between brackets {} all fields and methods.

@startuml
class Dummy {
   String data
   void methods()
}
@enduml
3.4 Defining visibility

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

<table>
<thead>
<tr>
<th>Character</th>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>private</td>
</tr>
<tr>
<td>#</td>
<td>protected</td>
</tr>
<tr>
<td>-</td>
<td>package private</td>
</tr>
<tr>
<td>+</td>
<td>public</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

note top of Object : In java, every class extends this one.
note "This is a floating note" as N1
note "This note is connected to several objects." as N2
Object .. N2
N2 .. ArrayList
@enduml
```
3.6 More on notes

It is also possible to use few html tags like:

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

You can also have a note on several lines.

```plantuml
@startuml
note top of Object
    In java, every <u>class</u> <b>extends</b> <i>this</i> one.
end note
note as N1
    This <size:10>note</size> is <u>also</u> <b><color:royalBlue>on several</color> <s>words</s> lines</b>
    And this is hosted by <img:sourceforge.jpg>
end note
@enduml
```
3.7 Abstract class and interface

You can declare a class as abstract using "abstract" or "abstract class" keywords. The class will be printed in italic.

You can use the interface and enum keywords too.

```plantuml
@startuml
abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection
List <|-- AbstractList
Collection <|-- AbstractCollection
Collection <|-- List
AbstractCollection <|-- AbstractList
AbstractList <|-- ArrayList
ArrayList : Object[] elementData
ArrayList : size()

enum TimeUnit
TimeUnit : DAYS
TimeUnit : HOURS
TimeUnit : MINUTES
@enduml
```
3.8 Using non-letters

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

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

```plantuml
@startuml
class "This is my class" as class1
class class2 as "It works this way too"
class2 *-- "foo/dummy" : use
@enduml
```
3.9 Hide attributes, methods...

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

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

Instead of empty members, you can use:

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

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

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

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

```plantuml
@startuml
class Dummy1 {  
  +myMethods()
}
class Dummy2 {  
  +hiddenMethod()
}
class Dummy3 <<Serializable>> {  
  String name
}
hide members
hide <<Serializable>> circle
show Dummy1 method
show <<Serializable>> fields
@enduml
```
3.10 Specific Spot

Usually, a spotted character (C, I, E or A) is used for classes, interface, enum and abstract classes. But you can define your own spot for a class when you define the stereotype, adding a single character and a color, like in this example:

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

```plantuml
D  Date
S  << Singleton >>
    System
```
3.11 Packages

You can define a package using the `package` keyword, and optionally declare a background color for your package (Using a html color code or name). When you declare classes, they are automatically put in the last used package, and you can close the package definition using the `end package` keyword. You can also use brackets `{ }`.

Note that package definitions can be nested.

```plantuml
@startuml
package "Classic Collections" #DDDDDD {
    Object <|-- ArrayList
}
package net.sourceforge.plantuml #Snow
    Object <|-- Demo1
    Demo1 *- Demo2
end package
@enduml
```

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

```plantuml
@startuml
package foo1.foo2
end package
package foo1.foo2.foo3 {
    class Object
}
foo1.foo2 +-- foo1.foo2.foo3
@enduml
```
3.12 Namespaces

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

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

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

```plantuml
@startuml

class BaseClass

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

    .BaseClass <|- Meeting

end namespace

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

    net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml
```
3.13 Changing arrows direction

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

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

You can also change directions by reversing the link:

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

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

```plantuml
@startuml
foo -left-> dummyLeft
foo -right-> dummyRight
foo -up-> dummyUp
foo -down-> dummyDown
@enduml
```
You can shorten the arrow by using only the first character of the direction (for example, -d- instead of -down-) or the two first characters (-do-)
Please note that you should not abuse this functionality: GraphViz gives usually good results without tweaking.

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

- bar ()- foo,
- bar ()-- foo,
- foo ()- bar

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

3.15 Title the diagram
The title keywords is used to put a title. You can use title and end title keywords for a longer title, as in sequence diagrams.

```plantuml
@startuml
title Simple <b>example</b>
of title
Object <|-- ArrayList
@enduml
```
3.16 Association classes

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

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

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

You can define it in another direction:

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

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

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

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

```plantuml
@startuml
skinparam classBackgroundColor PaleGreen
skinparam classArrowColor SeaGreen
skinparam classBorderColor SpringGreen
skinparam stereotypeCBackgroundColor YellowGreen

Class01 "1" *-- "many" Class02 : contains
Class03 o-- Class04 : aggregation
@enduml
```
3.18 Splitting large files

Sometimes, you will get some very large image files. You can use the "page (hpages)x(vpages)" command to split the generated image into several files:

- **hpages** is a number that indicated the number of horizontal pages,
- **vpages** is a number that indicated the number of vertical pages.

```plantuml
@startuml
| Split into 4 pages |
page 2x2

class BaseClass

namespace net.dummy 
    .BaseClass <|-- Person
    Meeting o-- Person
    .BaseClass <|-- Meeting
end namespace

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

BaseClass <|-- net.unused.Person
@enduml
```
4 Activity Diagram

4.1 Simple Activity

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

Use --> for arrows.

Example:

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

4.2 Label on arrows

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

You can put a label on a arrow using brackets [ and ] just after the arrow definition.

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

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

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

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

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

Please note that you should not abuse this functionality: GraphViz gives usually good results without tweaking.
4.4 Branches

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

```plantuml
@startuml
(•) --> "Initialisation"

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

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

It is also possible to nest branches.

```plantuml
@startuml
(*) --> if "Some Test" then

  -->[true] "activity 1"
  if "" then
  -> "activity 3" as a3
  else
  if "Other test" then
    -left--> "activity 5"
  else
    --> "activity 6"
  endif
  endif

else
  -->[false] "activity 2"
endif

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

![Diagram of activity flow](image-url)
4.6 Synchronization

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

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

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

--> (⋆)
@enduml
```

![Activity Diagram]

---

4.7 Long activity description

When you declare activities, you can span on several lines the description text. You can also add \n in the description. It is also possible to use few html tags like:

- `<b>`
- `<i>`
- `<font size="nn">` or `<size:nn>` to change font size
- `<font color="#AAAAAA">` or `<font color="colorName">`
- `<color:#AAAAAA>` or `<color:colorName>`
- `<img:file.png>` to include an image

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

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

![Activity Diagram](sourceforge.png)
4.8 Notes

You can add notes on a activity using the commands:

- note left,
- note right,
- note top,
- note bottom,

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

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

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

This note is on
several lines
4.9 Partition

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

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

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

```plantuml
.partition Conductor
(\*) --> "Climbs on Platform"
--> === S1 ===
--> Bows
.end partition

.partition Audience #LightSkyBlue
=== S1 === --> Applauds

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

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

![Diagram of partition in PlantUML](image)
4.10 Title the diagram

The `title` keywords is used to put a title. You can use `title` and `end title` keywords for a longer title, as in sequence diagrams.

```plantuml
@startuml
title Simple example
(**) --> "First activity"
--> (*)
@enduml```

```plaintext
Simple example
of title

First activity

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

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

```plantuml
@startuml
skinparam backgroundColor #AAFFFF
skinparam activityStartColor red
skinparam activityBarColor SaddleBrown
skinparam activityEndColor Silver
skinparam activityBackgroundColor Peru
skinparam activityBorderColor Peru
skinparam activityFontName Impact

(*) --> "Climbs on Platform"
---> === S1 ===
---> Bows
---> === S2 ===
---> WavesArmes
---> (*)&
@enduml
```
4.12 Complete example

@startuml
'\http://click.sourceforge.net/images/activity-diagram-small.png

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
  endif
else
  --->[false] ===REDIRECT_CHECK===
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
4.12 Complete example

5 Component Diagram

5.1 Components

Components must be bracketed.

You can also use the `component` keyword to defines a component.

And you can define an alias, using the `as` keyword.

This alias will be used latter, when defining relations.

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

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

You can also use the `interface` keyword to defines a usecase.

And you can define an alias, using the `as` keyword.

This alias will be used latter, when defining relations.

We will see latter that interface declaration is optional.

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

```
First Interface  Another interface  Interf3  Last interface
```
5.3 Basic example

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

```plantuml
@startuml
DataAccess - [First Component]
[First Component] ..> HTTP : use
@enduml
```
5.4 Using notes

You can use the

- note left of,
- note right of,
- note top of,
- note bottom of,

keywords to define notes related to a single object.

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

```plantuml
@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
```
5.5 Grouping Components

You can use the `package` keyword to group components and interfaces together.

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

package "Other Groups" {
    FTP - [Second Component]
    [First Component] --> FTP
}
@enduml
```
5.6 Changing arrows direction

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

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

You can also change directions by reversing the link:

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

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

@startuml
[Component] -left-> left
[Component] -right-> right
[Component] -up-> up
[Component] -down-> down
@enduml
You can shorten the arrow by using only the first character of the direction (for example, -d- instead of -down-) or the two first characters (-do-).

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

### 5.7 Title the diagram

The **title** keywords is used to put a title. You can use **title** and **end title** keywords for a longer title, as in sequence diagrams.

```
@startuml
title Very simple component\ndiagram

interface "Data Access" as DA

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

Very simple component
diagram

Data Access  First Component

use

HTTP
5.8 Skinparam

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

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

```plantuml
@startuml
skinparam componentFontSize 13
skinparam interfaceBackgroundColor RosyBrown
skinparam interfaceBorderColor black
skinparam componentBackgroundColor gold
skinparam componentBorderColor orange
skinparam componentArrowColor #FF6655
skinparam componentArrowFontColor #777777
skinparam componentFontName Courier
skinparam componentArrowFontName Impact

() "Data Access" as DA
DA - [First Component]
[First Component] ..> () HTTP : use
@enduml
```

![Diagram showing a component named 'Data Access' interacting with a component named 'First Component']
6 State Diagram

6.1 Simple State

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

Use --> for arrows.

```plantuml
@startuml
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string
State1 --> State2
State2 --> [*]
@enduml
```

![State Diagram](image-url)
6.2 Composite state

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

```plantuml
[*] --> 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
```
6.3 Long name

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

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

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

```plantuml
[*] --> Active

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

6.5 Arrow direction

You can use \( \rightarrow \) for horizontal arrows. It is possible to force arrow’s direction using the following syntax:

- \( \text{down}\rightarrow \) (default arrow)
- \( \text{right}\rightarrow \) or \( \rightarrow \)
- \( \text{left}\rightarrow \)
- \( \text{up}\rightarrow \)

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

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

Please note that you should not abuse this functionality: GraphViz gives usually good results without tweaking.
6.6 Note

You can also define notes using:

- note left of,
- note right of,
- note top of,
- note bottom of

keywords. You can also define notes on several lines.

@startuml
[*] --> Active
Active --> Inactive

note left of Active : this is a short\nnote

note right of Inactive
  A note can also
  be defined on
  several lines
end note

@enduml
6.7 More in notes

You can put notes on composite states.

@startuml
[*] --> NotShooting

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

note right of NotShooting : This is a note on a composite state
@enduml
7 Objects Diagram

7.1 Definition of objects
You define instance of objects using the object keywords.

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

7.2 Relations between objects
Relations between objects are defined using the following symbols:

- Extension: `<|--`
- Composition: `*--`
- Aggregation: `o--`

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

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

@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
7.3 Adding fields

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

```
@startuml
object user

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

@enduml
```

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

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

8.1 Footer and header

You can use the commands `header` or `footer` to add a footer or a header on any generated diagram.

You can optionally specify if you want a center, left or right footer/header, by adding a keyword.

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

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

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

header
<br>
Warning: This is a demonstration diagram.<br>Do not use in production.
endheader

center footer Generated for demonstration
@enduml
```
8.2 Zoom

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

- scale 1.5,
- scale 2/3,
- scale 200 width,
- scale 200 height,
- scale 200*100

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

Sometimes, and especially for printing, you may want to rotate the generated image, so that it fits better in the page. You can use the `rotate` command for this.

```plantuml
@startuml
rotate

title Simple Usecase
"Use the application" as (Use)
User -> (Use)
@enduml
```
9 Changing fonts and colors

9.1 Usage

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

```
skinparam backgroundColor yellow
```

You can use this command:

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

You can use either standard color name or RGB code.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Default Value</th>
<th>Color</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>white</td>
<td></td>
<td>Background of the page</td>
</tr>
<tr>
<td>activityArrowColor</td>
<td>#A80036</td>
<td></td>
<td>Color of arrows in activity diagrams</td>
</tr>
<tr>
<td>activityBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of activities</td>
</tr>
<tr>
<td>activityBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Color of activity borders</td>
</tr>
<tr>
<td>activityStartColor</td>
<td>black</td>
<td></td>
<td>Starting circle in activity diagrams</td>
</tr>
<tr>
<td>activityEndColor</td>
<td>black</td>
<td></td>
<td>Ending circle in activity diagrams</td>
</tr>
<tr>
<td>activityBarColor</td>
<td>black</td>
<td></td>
<td>Synchronization bar in activity diagrams</td>
</tr>
<tr>
<td>usecaseArrowColor</td>
<td>#A80036</td>
<td></td>
<td>Color of arrows in usecase diagrams</td>
</tr>
<tr>
<td>actorBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Head's color of actor in usecase diagrams</td>
</tr>
<tr>
<td>actorBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Color of actor borders in usecase diagrams</td>
</tr>
<tr>
<td>usecaseBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of usecases</td>
</tr>
<tr>
<td>usecaseBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Color of usecase borders in usecase diagrams</td>
</tr>
<tr>
<td>classArrowColor</td>
<td>#A80036</td>
<td></td>
<td>Color of arrows in class diagrams</td>
</tr>
<tr>
<td>classBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of classes/interface/enum in class diagrams</td>
</tr>
<tr>
<td>classBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Borders of classes/interface/enum in class diagrams</td>
</tr>
<tr>
<td>packageBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of packages in class diagrams</td>
</tr>
<tr>
<td>packageBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Borders of packages in class diagrams</td>
</tr>
<tr>
<td>stereotypeCBackgroundColor</td>
<td>#ADD1B2</td>
<td></td>
<td>Background of class spots in class diagrams</td>
</tr>
<tr>
<td>stereotypeABackgroundColor</td>
<td>#A9DCAF</td>
<td></td>
<td>Background of abstract class spots in class diagrams</td>
</tr>
<tr>
<td>stereotypeIBackgroundColor</td>
<td>#B1A7E5</td>
<td></td>
<td>Background of interface spots in class diagrams</td>
</tr>
<tr>
<td>stereotypeEBackgroundColor</td>
<td>#EB937F</td>
<td></td>
<td>Background of enum spots in class diagrams</td>
</tr>
<tr>
<td>componentArrowColor</td>
<td>#A80036</td>
<td></td>
<td>Color of arrows in component diagrams</td>
</tr>
<tr>
<td>componentBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of components</td>
</tr>
<tr>
<td>componentBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Borders of components</td>
</tr>
<tr>
<td>interfaceBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of interface in component diagrams</td>
</tr>
<tr>
<td>interfaceBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Border of interface in component diagrams</td>
</tr>
<tr>
<td>noteBackgroundColor</td>
<td>#FBFB77</td>
<td></td>
<td>Background of notes</td>
</tr>
<tr>
<td>noteBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Border of notes</td>
</tr>
<tr>
<td>stateBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of states in state diagrams</td>
</tr>
<tr>
<td>stateBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Border of states in state diagrams</td>
</tr>
<tr>
<td>stateArrowColor</td>
<td>#A80036</td>
<td></td>
<td>Colors of arrows in state diagrams</td>
</tr>
<tr>
<td>sequenceArrowColor</td>
<td>#A80036</td>
<td></td>
<td>Color of arrows in sequence diagrams</td>
</tr>
<tr>
<td>sequenceActorBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Head’s color of actor in sequence diagrams</td>
</tr>
<tr>
<td>sequenceActorBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Border of actor in sequence diagrams</td>
</tr>
<tr>
<td>sequenceGroupBackgroundColor</td>
<td>#EEEEEE</td>
<td></td>
<td>Header color of alt/opt/loop in sequence diagrams</td>
</tr>
<tr>
<td>sequenceLifeLineBackgroundColor</td>
<td>white</td>
<td></td>
<td>Background of life line in sequence diagrams</td>
</tr>
<tr>
<td>sequenceLifeLineBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Border of life line in sequence diagrams</td>
</tr>
<tr>
<td>sequenceParticipantBackgroundColor</td>
<td>#FFFEFE</td>
<td></td>
<td>Background of participant in sequence diagrams</td>
</tr>
<tr>
<td>sequenceParticipantBorderColor</td>
<td>#A80036</td>
<td></td>
<td>Border of participant in sequence diagrams</td>
</tr>
</tbody>
</table>
## 9.3 Font color, name and size

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

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

You can also change the default font for all fonts using `skinparam defaultFontName`.

Example:
```
skinparam defaultFontName Aapex
```

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>activityFontColor</td>
<td>black</td>
<td>Used for activity box</td>
</tr>
<tr>
<td>activityFontSize</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>activityFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>activityFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>activityArrowFontColor</td>
<td>black</td>
<td>Used for text on arrows in activity diagrams</td>
</tr>
<tr>
<td>activityArrowFontSize</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>activityArrowFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>activityArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>circledCharacterFontColor</td>
<td>black</td>
<td>Used for text in circle for class, enum and others</td>
</tr>
<tr>
<td>circledCharacterFontSize</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>circledCharacterFontStyle</td>
<td>bold</td>
<td></td>
</tr>
<tr>
<td>circledCharacterFontName</td>
<td>Courier</td>
<td></td>
</tr>
<tr>
<td>circledCharacterRadius</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>classArrowFontColor</td>
<td>black</td>
<td>Used for text on arrows in class diagrams</td>
</tr>
<tr>
<td>classArrowFontSize</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>classArrowFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>classArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classAttributeFontColor</td>
<td>black</td>
<td>Class attributes and methods</td>
</tr>
<tr>
<td>classAttributeFontSize</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>classAttributeIconSize</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>classAttributeFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>classAttributeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classArrowFontColor</td>
<td>black</td>
<td>Used for classes name</td>
</tr>
<tr>
<td>classFontSize</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>classFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>classFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classStereotypeFontColor</td>
<td>black</td>
<td>Used for stereotype in classes</td>
</tr>
<tr>
<td>classStereotypeFontSize</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>classStereotypeFontStyle</td>
<td>italic</td>
<td></td>
</tr>
<tr>
<td>classStereotypeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentFontColor</td>
<td>black</td>
<td>Used for components name</td>
</tr>
<tr>
<td>componentFontSize</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>componentFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>componentFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentStereotypeFontColor</td>
<td>black</td>
<td>Used for stereotype in components</td>
</tr>
<tr>
<td>componentStereotypeFontSize</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>componentStereotypeFontStyle</td>
<td>italic</td>
<td></td>
</tr>
<tr>
<td>componentStereotypeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentArrowFontColor</td>
<td>black</td>
<td>Used for text on arrows in component diagrams</td>
</tr>
<tr>
<td>componentArrowFontSize</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>componentArrowFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>componentArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Font Attribute</td>
<td>Color</td>
<td>Size</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td><code>noteFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>noteFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>noteFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>noteFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>packageFontColor</code></td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td><code>packageFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>packageFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>packageFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceActorFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>sequenceActorFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceActorFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceActorFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceDividerFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>sequenceDividerFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceDividerFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceDividerFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceArrowFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>sequenceArrowFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceArrowFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceArrowFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceGroupingFontColor</code></td>
<td>black</td>
<td>11</td>
</tr>
<tr>
<td><code>sequenceGroupingFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceGroupingFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceGroupingFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceGroupingHeaderFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>sequenceGroupingHeaderFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceGroupingHeaderFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceGroupingHeaderFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceParticipantFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>sequenceParticipantFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceParticipantFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceParticipantFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceTitleFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>sequenceTitleFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceTitleFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sequenceTitleFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>titleFontColor</code></td>
<td>black</td>
<td>18</td>
</tr>
<tr>
<td><code>titleFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>titleFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>titleFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateFontColor</code></td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td><code>stateFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateArrowFontColor</code></td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td><code>stateArrowFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateArrowFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateArrowFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateAttributeFontColor</code></td>
<td>black</td>
<td>12</td>
</tr>
<tr>
<td><code>stateAttributeFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateAttributeFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stateAttributeFontName</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseFontColor</code></td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td><code>usecaseFontSize</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseFontStyle</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseFontName</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 9.3 Font color, name and size

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Color</th>
<th>Size</th>
<th>Style</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usecaseStereotypeFontColor</code></td>
<td>black</td>
<td>14</td>
<td>italic</td>
<td></td>
<td>Used for stereotype in usecase</td>
</tr>
<tr>
<td><code>usecaseStereotypeFontSize</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseStereotypeFontStyle</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseStereotypeFontName</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseActorFontColor</code></td>
<td>black</td>
<td>14</td>
<td>plain</td>
<td></td>
<td>Used for actor labels in usecase diagrams</td>
</tr>
<tr>
<td><code>usecaseActorFontSize</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseActorFontStyle</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseActorFontName</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseActorStereotypeFontColor</code></td>
<td>black</td>
<td>14</td>
<td>italic</td>
<td></td>
<td>Used for stereotype for actor</td>
</tr>
<tr>
<td><code>usecaseActorStereotypeFontSize</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseActorStereotypeFontStyle</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseActorStereotypeFontName</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseArrowFontColor</code></td>
<td>black</td>
<td>13</td>
<td>plain</td>
<td></td>
<td>Used for text on arrows in usecase diagrams</td>
</tr>
<tr>
<td><code>usecaseArrowFontSize</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseArrowFontStyle</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>usecaseArrowFontName</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>footerFontColor</code></td>
<td>black</td>
<td>10</td>
<td>plain</td>
<td></td>
<td>Used for footer</td>
</tr>
<tr>
<td><code>footerFontSize</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>footerFontStyle</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>footerFontName</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>headerFontColor</code></td>
<td>black</td>
<td>10</td>
<td>plain</td>
<td></td>
<td>Used for header</td>
</tr>
<tr>
<td><code>headerFontSize</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>headerFontStyle</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>headerFontName</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9.4 Black and White

You can force the use of a blackwhite output using the `skinparam monochrome true` command.

```plantuml
@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
```
10 Preprocessing

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

Those functionalities are very similar to the C language preprocessor, except that the special character "#" has been changed to the exclamation mark "!".

10.1 Including files

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

Imagine you have the very same class that appears in many diagrams. Instead of duplicating the description of this class, you can define a file that contains the description.

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

File List.iuml:

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

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

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

```plantuml
!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
```

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

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

You can use \!ifdef XXX and \!endif directives to have conditional drawings.
The lines between those two directives will be included only if the constant after the \!ifdef directive has been defined before.

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

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

File ArrayList.iuml:

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

You can then use the \!define directive to activate the conditional part of the diagram.

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

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

```
11 Internationalization

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

@startuml
skinparam backgroundColor #EEEBDC
actor 使用者
participant “頭等艙” as A
participant “第二類” as B
participant “最後一堂課” as 別的東西

使用者 -> A: 完成這項工作
activate A
A -> B: 創建請求
activate B
B -> 別的東西: 創建請求
activate 別的東西
別的東西 --> B: 這項工作完成
destroy 別的東西
B --> A: 請求創建
deactivate B
A --> 使用者: 做完
deactivate A
@enduml

11.1 Charset

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

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

Or, with the ant task:

```xml
<!-- Put images in c:/images directory -->
<target name="main">
```
Depending of your Java installation, the following charset should be available: ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.

11.2 Font Issues

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

```
skinparam defaultFontName MS Mincho
```

![Diagram](image-url)
Here is the list of colors recognized by PlantUML. Note that color names are case insensitive.

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Color Name</th>
<th>Color Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliceBlue</td>
<td>GhostWhite</td>
<td>NavajoWhite</td>
</tr>
<tr>
<td>AntiqueWhite</td>
<td>GoldenRod</td>
<td>Navy</td>
</tr>
<tr>
<td>Aquamarine</td>
<td>Gold</td>
<td>OldLace</td>
</tr>
<tr>
<td>Aqua</td>
<td>Gray</td>
<td>OliveDrab</td>
</tr>
<tr>
<td>Azure</td>
<td>GreenYellow</td>
<td>Olive</td>
</tr>
<tr>
<td>Beige</td>
<td>Green</td>
<td>OrangeRed</td>
</tr>
<tr>
<td>Bisque</td>
<td>HoneyDew</td>
<td>Orange</td>
</tr>
<tr>
<td>Black</td>
<td>HotPink</td>
<td>Orchid</td>
</tr>
<tr>
<td>BlanchedAlmond</td>
<td>IndianRed</td>
<td>PaleGoldenRod</td>
</tr>
<tr>
<td>BlueViolet</td>
<td>Indigo</td>
<td>PaleGreen</td>
</tr>
<tr>
<td>Blue</td>
<td>Ivory</td>
<td>PaleTurquoise</td>
</tr>
<tr>
<td>Brown</td>
<td>Khaki</td>
<td>PaleVioletRed</td>
</tr>
<tr>
<td>BurlyWood</td>
<td>LavenderBlush</td>
<td>PapayaWhip</td>
</tr>
<tr>
<td>CadetBlue</td>
<td>Lavender</td>
<td>PeachPuff</td>
</tr>
<tr>
<td>Chartreuse</td>
<td>LawnGreen</td>
<td>Peru</td>
</tr>
<tr>
<td>Chocolate</td>
<td>LemonChiffon</td>
<td>Pink</td>
</tr>
<tr>
<td>Coral</td>
<td>LightBlue</td>
<td>Plum</td>
</tr>
<tr>
<td>CornflowerBlue</td>
<td>LightCoral</td>
<td>PowderBlue</td>
</tr>
<tr>
<td>Cornsilk</td>
<td>LightCyan</td>
<td>Purple</td>
</tr>
<tr>
<td>Crimson</td>
<td>LightGoldenRodYellow</td>
<td>Red</td>
</tr>
<tr>
<td>Cyan</td>
<td>LightGreen</td>
<td>RosyBrown</td>
</tr>
<tr>
<td>DarkBlue</td>
<td>LightGrey</td>
<td>RoyalBlue</td>
</tr>
<tr>
<td>DarkCyan</td>
<td>LightPink</td>
<td>SaddleBlue</td>
</tr>
<tr>
<td>DarkGoldenRod</td>
<td>LightSalmon</td>
<td>Salmon</td>
</tr>
<tr>
<td>DarkGray</td>
<td>LightSeaGreen</td>
<td>SandyBrown</td>
</tr>
<tr>
<td>DarkGreen</td>
<td>LightSkyBlue</td>
<td>SeaGreen</td>
</tr>
<tr>
<td>DarkKhaki</td>
<td>LightSlateGray</td>
<td>SeaShell</td>
</tr>
<tr>
<td>DarkMagenta</td>
<td>LightSteelBlue</td>
<td>Sienna</td>
</tr>
<tr>
<td>DarkOliveGreen</td>
<td>LightYellow</td>
<td>Silver</td>
</tr>
<tr>
<td>DarkOrchid</td>
<td>LimeGreen</td>
<td>SkyBlue</td>
</tr>
<tr>
<td>DarkRed</td>
<td>Lime</td>
<td>SlateBlue</td>
</tr>
<tr>
<td>DarkSalmon</td>
<td>Linen</td>
<td>SlateGray</td>
</tr>
<tr>
<td>DarkSeaGreen</td>
<td>Magenta</td>
<td>Snow</td>
</tr>
<tr>
<td>DarkSlateBlue</td>
<td>Maroon</td>
<td>SpringGreen</td>
</tr>
<tr>
<td>DarkSlateGray</td>
<td>MediumAquamarine</td>
<td>SteelBlue</td>
</tr>
<tr>
<td>DarkTurquoise</td>
<td>MediumBlue</td>
<td>Tan</td>
</tr>
<tr>
<td>DarkViolet</td>
<td>MediumOrchid</td>
<td>Teal</td>
</tr>
<tr>
<td>Darkorange</td>
<td>MediumPurple</td>
<td>Thistle</td>
</tr>
<tr>
<td>DeepPink</td>
<td>MediumSeaGreen</td>
<td>Tomato</td>
</tr>
<tr>
<td>DeepSkyBlue</td>
<td>MediumSlateBlue</td>
<td>Turquoise</td>
</tr>
<tr>
<td>DimGray</td>
<td>MediumSpringGreen</td>
<td>Violet</td>
</tr>
<tr>
<td>DodgerBlue</td>
<td>MediumTurquoise</td>
<td>Wheat</td>
</tr>
<tr>
<td>Firebrick</td>
<td>MediumVioletRed</td>
<td>WhiteSmoke</td>
</tr>
<tr>
<td>FloralWhite</td>
<td>MidnightBlue</td>
<td>White</td>
</tr>
<tr>
<td>ForestGreen</td>
<td>MintCream</td>
<td>YellowGreen</td>
</tr>
<tr>
<td>Fuchsia</td>
<td>MistyRose</td>
<td>Yellow</td>
</tr>
<tr>
<td>Gainsboro</td>
<td>Moccasin</td>
<td></td>
</tr>
</tbody>
</table>
## Contents

1 **Sequence Diagram**
   1.1 Basic examples  .................................................. 1
   1.2 Declaring participant ........................................... 2
   1.3 Use non-letters in participants  ............................ 3
   1.4 Message to Self .................................................. 3
   1.5 Message sequence numbering  ................................. 4
   1.6 Title ............................................................... 6
   1.7 Splitting diagrams .............................................. 7
   1.8 Grouping message ................................................ 8
   1.9 Notes on messages .............................................. 9
   1.10 Some other notes ................................................ 10
   1.11 Formatting using HTML ....................................... 11
   1.12 Divider ........................................................... 12
   1.13 Lifeline Activation and Destruction ..................... 13
   1.14 Participant creation ......................................... 15
   1.15 Incoming and outgoing messages  ......................... 16
   1.16 Stereotypes and Spots ...................................... 17
   1.17 More information on titles ................................ 18
   1.18 Participants englober ....................................... 20
   1.19 Removing Footer ............................................... 21
   1.20 Skinparam ........................................................ 22
   1.21 Skin ............................................................ 23

2 **Use Case Diagram**
   2.1 Usecases .......................................................... 24
   2.2 Actors ............................................................. 25
   2.3 Basic example .................................................... 26
   2.4 Extension .......................................................... 27
   2.5 Using notes ....................................................... 28
   2.6 Stereotypes ........................................................ 29
   2.7 Changing arrows direction .................................. 30
   2.8 Title the diagram .............................................. 31
   2.9 Left to right direction ....................................... 32
   2.10 Skinparam ........................................................ 33

3 **Class Diagram**
   3.1 Relations between classes .................................... 34
   3.2 Label on relations .............................................. 35
   3.3 Adding methods .................................................. 36
   3.4 Defining visibility .............................................. 37
   3.5 Notes and stereotypes ....................................... 38
   3.6 More on notes ..................................................... 39
   3.7 Abstract class and interface ............................... 40
   3.8 Using non-letters ............................................. 41
   3.9 Hide attributes, methods... ................................. 42
   3.10 Specific Spot ...................................................... 43
   3.11 Packages .......................................................... 44
   3.12 Namespaces ....................................................... 45
   3.13 Changing arrows direction ................................ 46
   3.14 Lollipop interface ............................................ 47
   3.15 Title the diagram ............................................. 47
   3.16 Association classes ......................................... 48
   3.17 Skinparam ........................................................ 49
   3.18 Splitting large files ....................................... 50
## 4 Activity Diagram
4.1 Simple Activity ........................................ 51
4.2 Label on arrows ....................................... 51
4.3 Changing arrow direction ............................... 52
4.4 Branches ............................................... 53
4.5 More on Branches ...................................... 54
4.6 Synchronization ....................................... 55
4.7 Long activity description ............................... 56
4.8 Notes ................................................ 57
4.9 Partition .............................................. 58
4.10 Title the diagram ..................................... 59
4.11 Skinparam ............................................ 60
4.12 Complete example ..................................... 61

## 5 Component Diagram
5.1 Components ............................................ 63
5.2 Interfaces ............................................. 64
5.3 Basic example .......................................... 65
5.4 Using notes ........................................... 66
5.5 Grouping Components .................................. 67
5.6 Changing arrows direction ............................. 68
5.7 Title the diagram ...................................... 69
5.8 Skinparam ............................................. 70

## 6 State Diagram
6.1 Simple State ............................................ 71
6.2 Composite state ........................................ 72
6.3 Long name ............................................. 73
6.4 Concurrent state ....................................... 74
6.5 Arrow direction ........................................ 75
6.6 Note .................................................. 76
6.7 More in notes .......................................... 77

## 7 Objects Diagram
7.1 Definition of objects ................................... 78
7.2 Relations between objects .............................. 78
7.3 Adding fields .......................................... 79

## 8 Common commands
8.1 Footer and header ...................................... 80
8.2 Zoom ................................................ 81
8.3 Rotation .............................................. 82

## 9 Changing fonts and colors
9.1 Usage ................................................ 83
9.2 Color ............................................... 84
9.3 Font color, name and size .............................. 85
9.4 Black and White ....................................... 88

## 10 Preprocessing
10.1 Including files ........................................ 89
10.2 Constant definition ................................... 90
10.3 Conditions ........................................... 91

## 11 Internationalization
11.1 Charset .............................................. 92
11.2 Font Issues .......................................... 93

## 12 Color Names
12 Color Names ........................................... 94