Superfast modeling in PlantUML

Workshop PlantUML for creating UML diagrams

Sander van Geloven

Hellebaard

May 11, 2019

Unified Modeling Language (UML)

PlantUML

Introduction

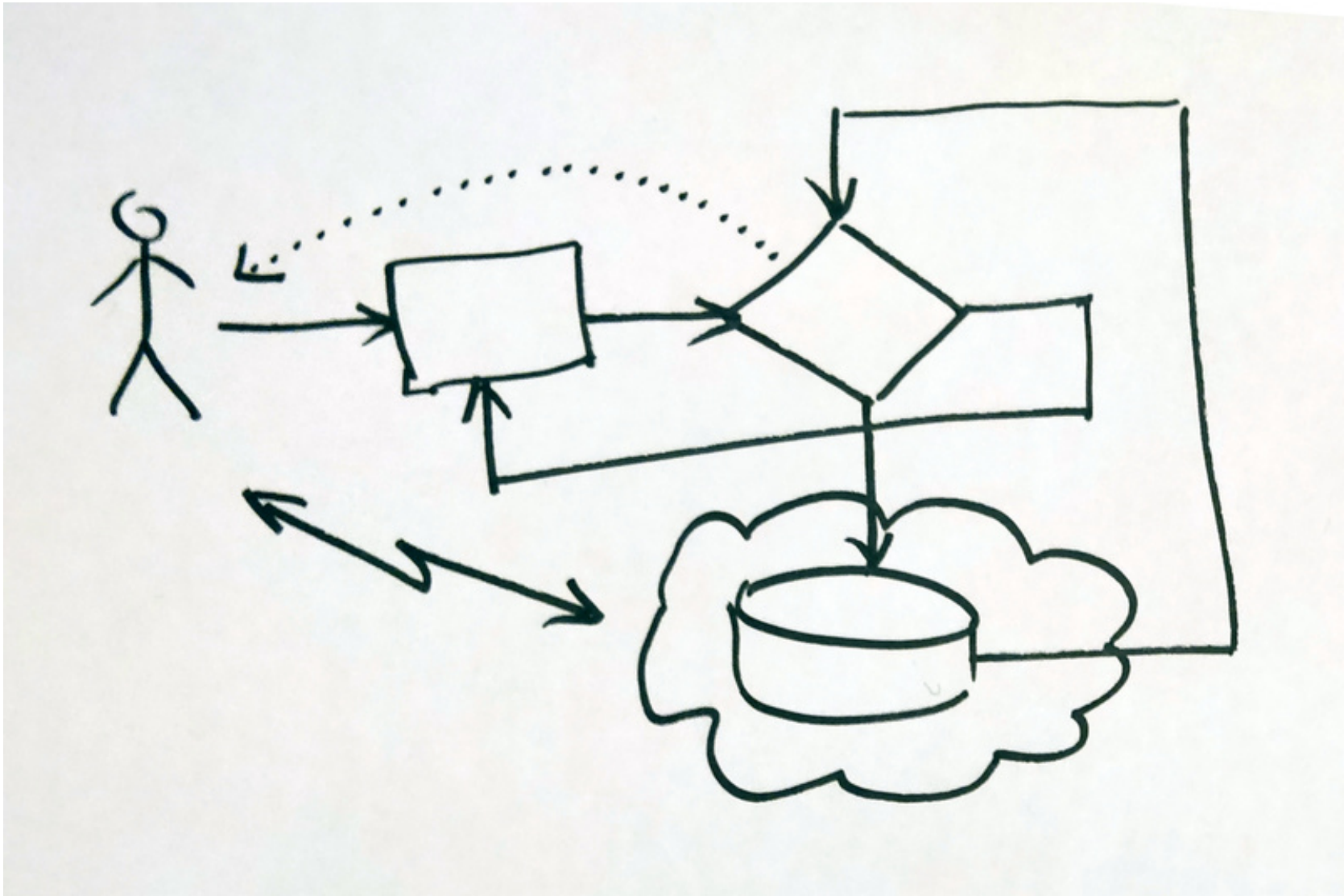
Sander van Geloven

- information analyst / ICT architect
- self-employed at hellebaard.nl
- curriculum vitae at linkedin.com/in/svgeloven
- writing tools for spelling and grammar
  - Dutch spelling and grammar opentaal.org
  - Dutch word list woordenlijst.org
  - spell checker nuspell.github.io

Unified Modeling Language (UML) — What is it?

- modeling language for visualizing system design
- designed in 1994 – 1996 by
  - Grady Booch (1955)
  - Ivar Jacobson (1939)
  - James Rumbaugh (1947)
- diagrams to represent structural and behavior information, including interaction aspects
- version 2.5.1, 5 December 2017, 800 pages, uml.org

UML — No more flow charts on a back of a coaster



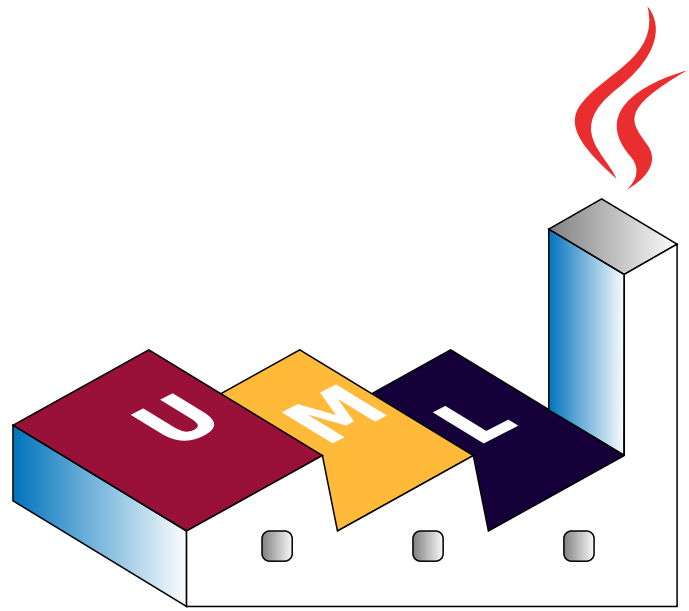
UML — Common problems

Common problems with visual editors for UML diagrams:

1. wasting time positioning and repositioning elements
2. wasting time untangling intersecting lines
3. wasting time setting (partially working) grid snap
4. wasting time applying font and font alignment
5. wasting time and risking RSI while micro-aligning

PlantUML — What is it?

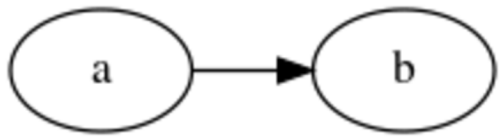
- FOSS tool to create UML diagrams from plain text
- developed 2009 – 2019 by Arnaud Roques
- GPLv3 but also LGPL, GPLv2, Apache License, Eclipse Public License and MIT License
- version 1.2019.5, 23 April 2019, 17 types of diagrams, 12 output file formats, plantuml.com



PlantUML — How it works

PlantUML:

- uses the FOSS graph rendering Graphviz, graphviz.org for example `a -> b` creates



- has text file (`.puml` or `.pu`) as input
- generates diagram in output file or in a GUI
- uses convention over configuration, yet is highly configurable

PlantUML — Supported UML diagrams

UML behavior diagrams:

1. activity diagram
2. sequence diagram
3. timing diagram
4. state diagram
5. usecase diagram

UML structure diagrams:

6. class diagram
7. deployment diagram
8. component diagram
9. object diagram

PlantUML — Supported non-UML diagrams

Supported non-UML diagrams:

1. wireframe graphical interface
2. Archimate diagram
3. Specification and Description Language (SDL) diagram
4. Ditaa diagram
5. Gannt diagram
6. MindMap diagram
7. work breakdown structure diagram
8. mathematic with AsciiMath or JLaTeXMath notation

PlantUML is used by many wikis, forums, text editors, IDE, programming languages and documentation generators.

PlantUML — Supported output formats

1. `gui` - Graphical User-Interface (your screen)
2. `png` - Portable Network Graphics (default)
3. `svg` - Scalable Vector Graphics
4. `eps` - Encapsulated PostScript
5. `pdf` - Portable Document Format
6. `vdx` - Virtual Document eXchange
7. `xmi` - XML Metadata Interchange (class diagram)
8. `txt` - ASCII art
9. `utxt` - ASCII art using Unicode characters
10. `html` - Hypertext Markup Language (class diagram)
11. `scxml` - State Chart XML (state diagram)
12. `latex` - LaTeX/Tikz format
13. `latex:nopreamble` - LaTeX/Tikz format w/o preamble

1 Usecase Diagram — Simple example

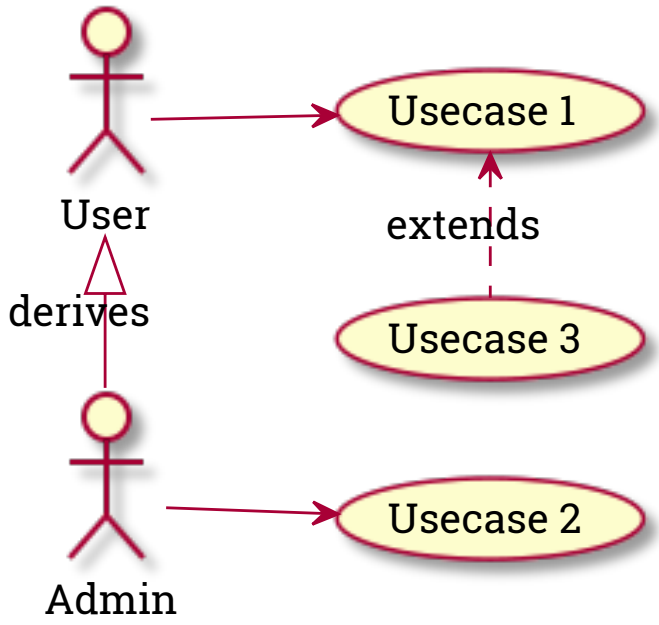
overview of users in relation to functional requirements in usecases

```
left to right direction

User <|-right- Admin : derives

User --> (Usecase 1)
Admin --> (Usecase 2)

(Usecase 3) .left.> (Usecase 1)
 : extends
```



Note that the arrow directions are rotated 90° clockwise.

1 Usecase Diagram — Exercise

Exercise 1/5:

Make a minimal and optimized usecase diagram in PlantUML that describes the the roles of customers and staff eating and working at a restaurant.

1 Usecase Diagram — Restaurant

A customer may:

- order
- eat
- pay

Staff may:

- serve
- cook
- charge

All staff can also use the restaurant in the role of a customer.

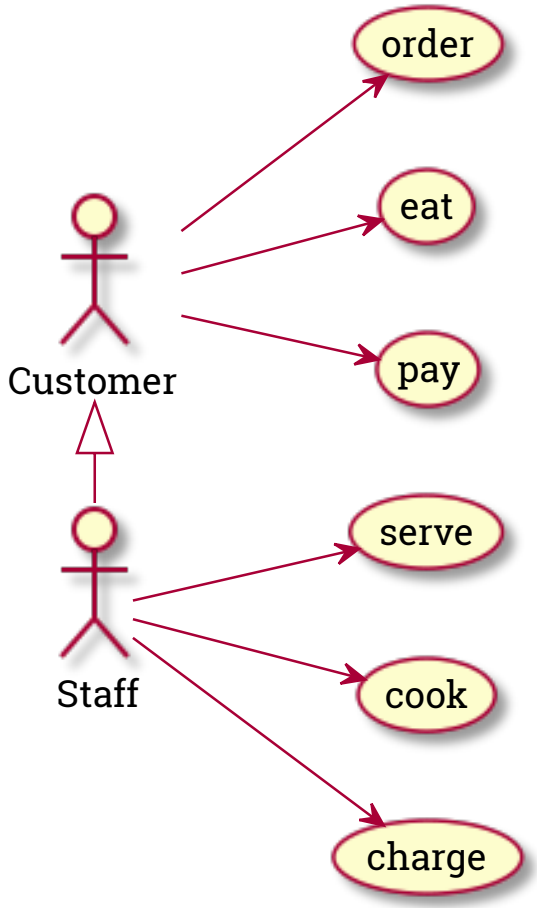
1 Usecase Diagram — Restaurant

```
left to right direction

Customer --> (order)
Customer --> (eat)
Customer --> (pay)
Staff --> (serve)
Staff --> (cook)
Staff --> (charge)

Customer <|-right- Staff
```

1 Usecase Diagram — Restaurant



1 Usecase Diagram — Restaurant

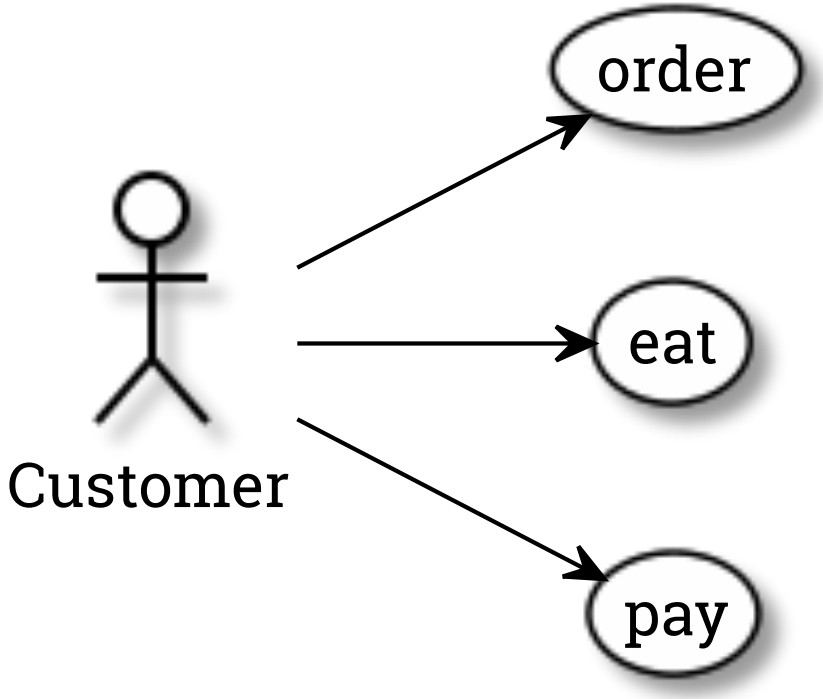
```
skinparam TitleFontStyle Bold
skinparam ArrowColor Black
skinparam ActorBorderColor Black
skinparam UsecaseBorderColor Black
skinparam ActorBackgroundColor White
skinparam UsecaseBackgroundColor White
Title Usecase Diagram - Restaurant

left to right direction

Customer --> (order)
Customer --> (eat)
Customer --> (pay)
```

1 Usecase Diagram — Restaurant

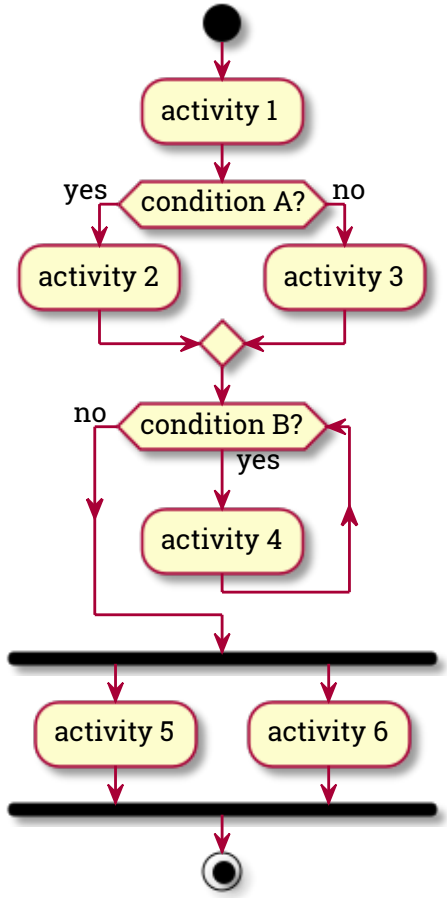
Usecase Diagram — Restaurant



2 Activity Diagram — Simple example

activities in sequence, selection, iteration and parallel

```
start
:activity 1;
if (condition A?) then (yes)
    :activity 2;
else (no)
    :activity 3;
endif
while (condition B?) is (yes)
    :activity 4;
endwhile (no)
fork
    :activity 5;
fork again
    :activity 6;
end fork
stop
```



2 Activity Diagram — Exercise

Exercise 2/5:

Make a minimal and optimized activity diagram in PlantUML of making a pot of tea when all items needed are withing reach.

2 Activity Diagram — Making tea

Required activities:

- choose teabag
- put teabag in teapot
- switch kettle on
- fill kettle with water
- kettle switches heater off
- poor boiled water from kettle in teapot
- kettle heats water

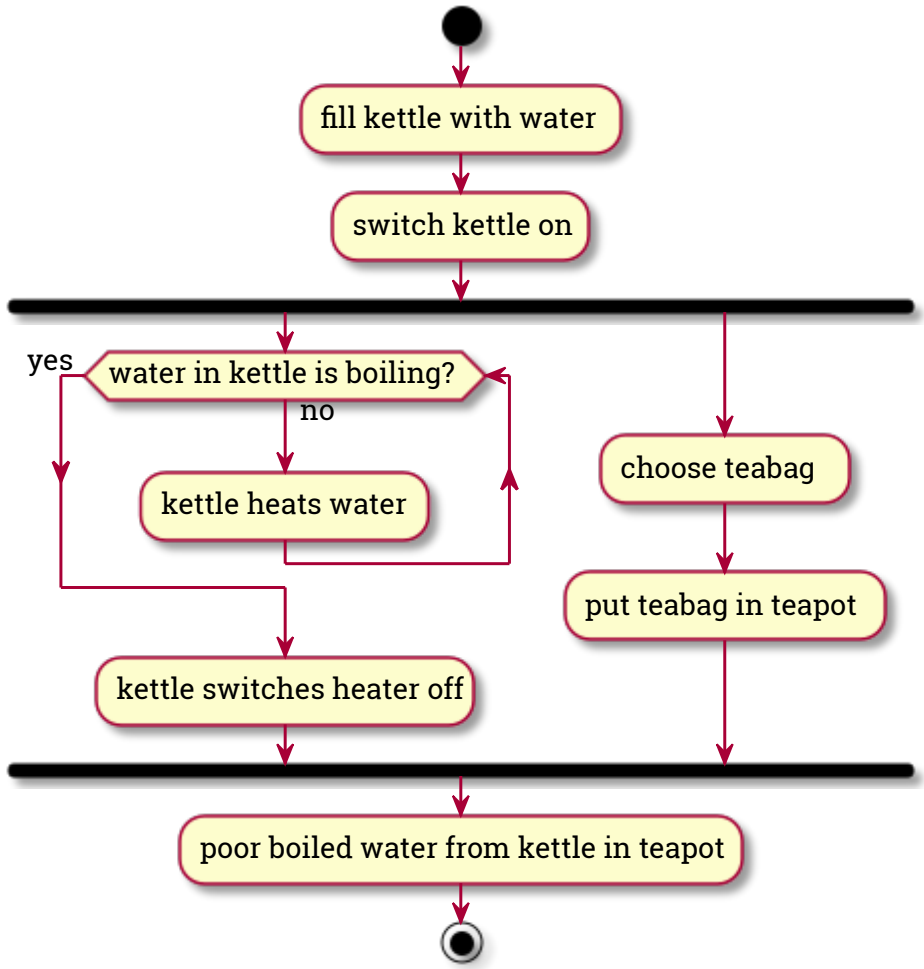
Required conditions:

- water in kettle is boiling?

2 Activity Diagram — Making tea

```
start
:fill kettle with water;
:switch kettle on;
fork
    while (water in kettle is boiling?) is (no)
        :kettle heats water;
    endwhile (yes)
    :kettle switches heater off;
fork again
    :choose teabag;
    :put teabag in teapot;
end fork
:poor boiled water from kettle in teapot;
stop
```

2 Activity Diagram — Making tea

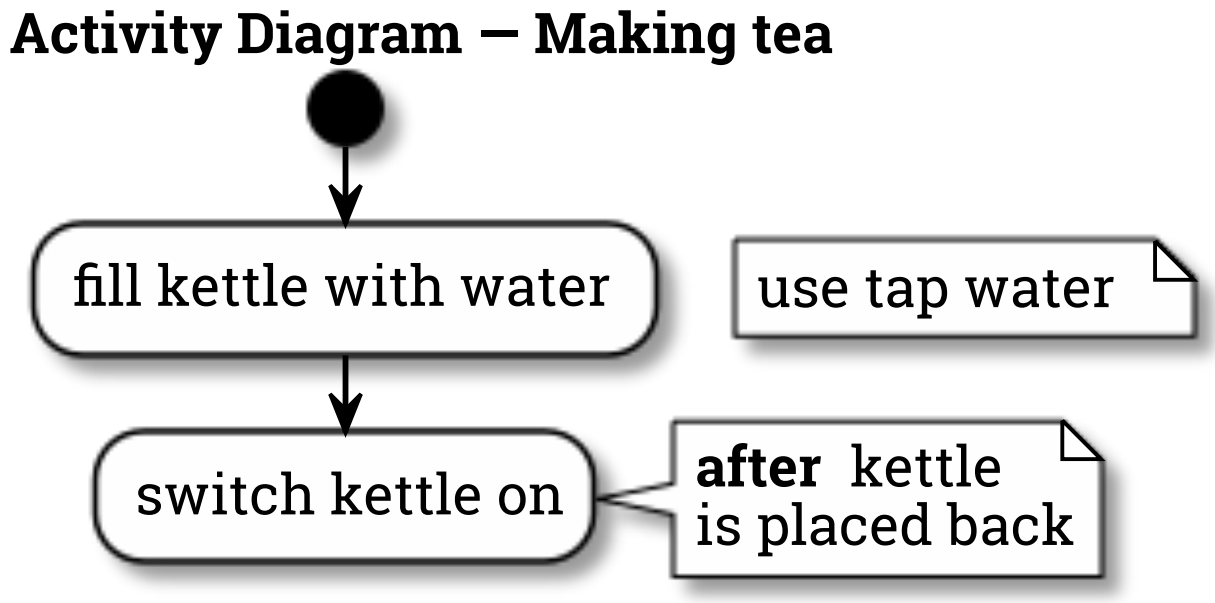


2 Activity Diagram – Making tea

```
skinparam TitleFontStyle Bold
skinparam ArrowColor Black
skinparam ActivityBorderColor Black
skinparam NoteBorderColor Black
skinparam ActivityBackgroundColor White
skinparam NoteBackgroundColor White
Title Activity Diagram - Making tea

start
:fill kettle with water;
floating note right: use tap water
:switch kettle on;
note right
    <b>after</b> kettle
    is placed back
end note
```

2 Activity Diagram — UML diagrams

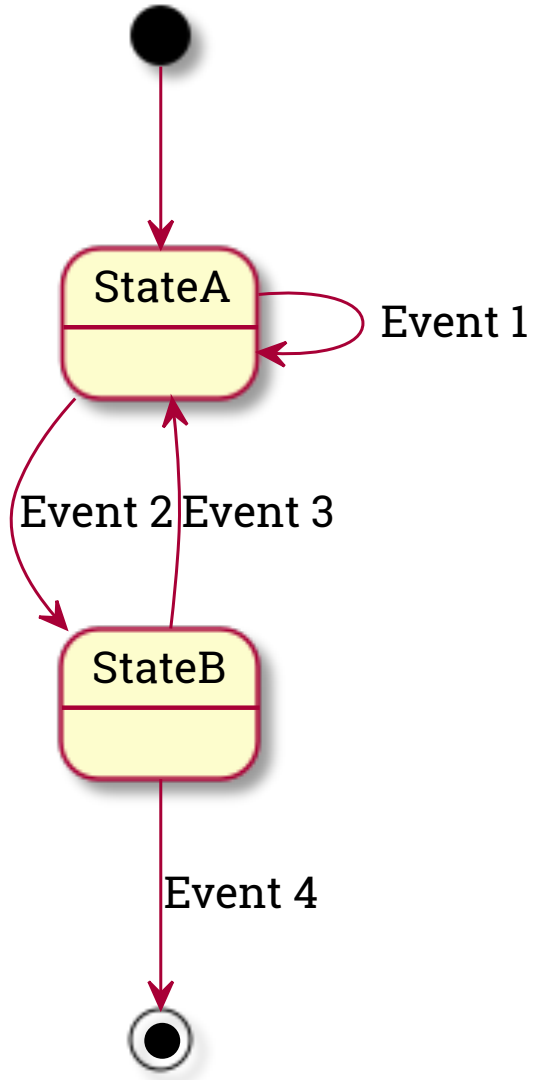


3 State Diagram — Simple example

behavioral state machine
with events that trigger state
transitions

```
[*] -down-> StateA
StateA --> StateA : Event 1
StateA -down-> StateB : Event 2
StateB -up-> StateA : Event 3
StateB -down-> [*] : Event 4
```

Note the `down` and `up` arrow directions and that the initial transition has no event.



3 State Diagram — Exercise

Exercise 3/5:

Make a minimal and optimized state diagram in PlantUML that describes the states and events of a swimming pool locker under normal circumstances.

3 State Diagram — Swimming pool locker

Possible events:

- open or close door
- insert coin
- insert or retract key
- turn key clockwise or counterclockwise

3 State Diagram — Swimming pool locker

Possible aspects of states:

- door is open, closed or locked
- coin is inserted or not
- key is inserted, turned or retracted

3 State Diagram — Swimming pool locker

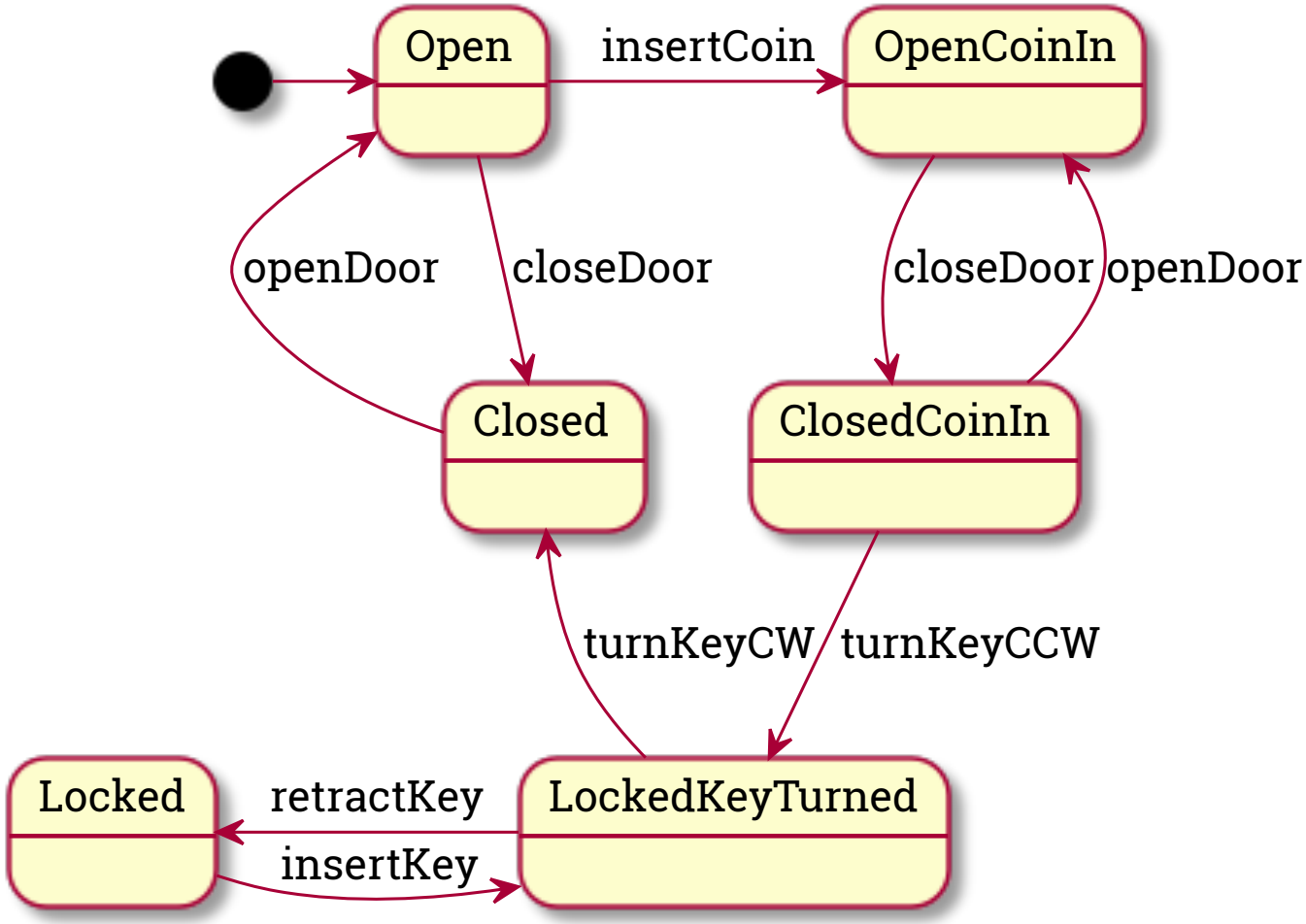
```
[*] -right-> Open
Open -right-> OpenCoinIn : insertCoin

OpenCoinIn -down-> ClosedCoinIn : closeDoor
ClosedCoinIn -up-> OpenCoinIn : openDoor

ClosedCoinIn -down-> LockedKeyTurned : turnKeyCCW
LockedKeyTurned -left-> Locked : retractKey
Locked -right-> LockedKeyTurned : insertKey
LockedKeyTurned -left-> Closed : turnKeyCW

Closed -up-> Open : openDoor
Open -down-> Closed : closeDoor
```

3 State Diagram — Swimming pool locker



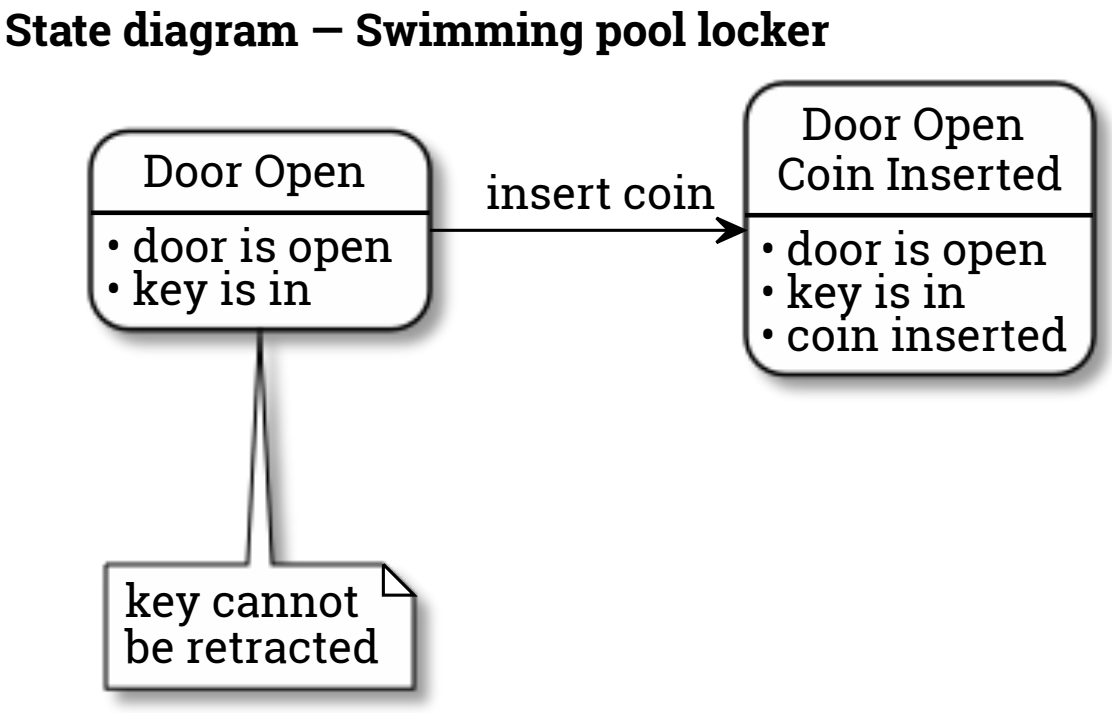
3 State Diagram — Swimming pool locker

```
skinparam TitleFontStyle Bold
skinparam ArrowColor Black
skinparam StateBorderColor Black
skinparam NoteBorderColor Black
skinparam StateBackgroundColor White
skinparam NoteBackgroundColor White
title State diagram - Swimming pool locker
Open -right-> OpenCoinIn : insert coin

state "Door Open" as Open
Open : • door is open
Open : • key is in
note bottom of Open : key cannot\nbe retracted

state "Door Open\nCoin Inserted" as OpenCoinIn
OpenCoinIn : • door is open\n• key is in\n• coin inserted
```

3 State Diagram — Swimming pool locker

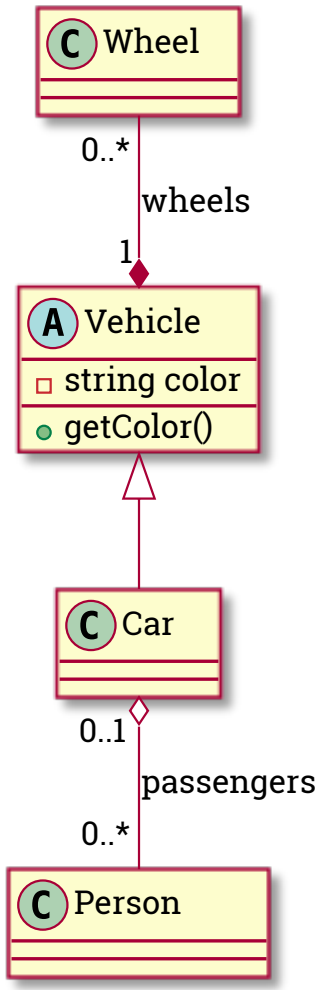


4 Class Diagram — Simple example

structure of classes with attributes
and methods and object relations

```
abstract class Vehicle
Vehicle : -string color
Vehicle : +getColor()
Vehicle "1" *-up- "0..*" Wheel : wheels
Vehicle <|-- Car
Car "0..1" o-- "0..*" Person
 : passengers
```

Note the `up` arrow direction and that attribute and method visibility.



4 Class Diagram — Exercise

Exercise 4/5:

Make a minimal and optimized class diagram in PlantUML that describes the hierarchy of the following UML class diagrams:

- state diagrams
- object diagrams
- activity diagrams
- component diagrams
- usecase diagrams
- class diagrams

4 Class Diagram — UML diagrams

A usecase diagram is a behavior diagram.
An activity diagram is a behavior diagram.
A state diagram is a behavior diagram.

A class diagram is a structure diagram.
An object diagram is a structure diagram.
A component diagram is a structure diagram.

A behavior diagram is a diagram.
A structure diagram is a diagram.

All but leaf classes are abstract.

4 Class Diagram — UML diagrams

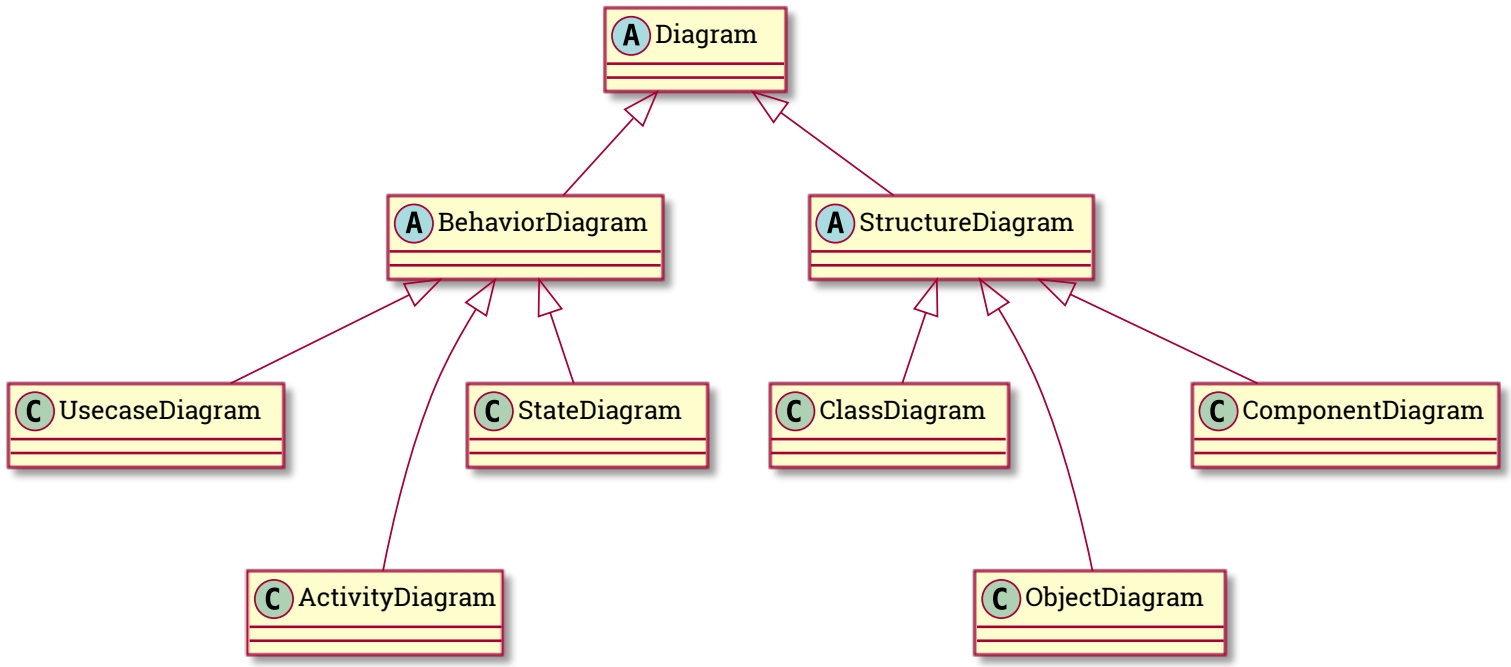
```
abstract class Diagram as Diagram
abstract class BehaviorDiagram
abstract class StructureDiagram

Diagram <|-- BehaviorDiagram
Diagram <|-- StructureDiagram

BehaviorDiagram <|-- UsecaseDiagram
BehaviorDiagram <|--- ActivityDiagram
BehaviorDiagram <|-- StateDiagram

StructureDiagram <|-- ClassDiagram
StructureDiagram <|--- ObjectDiagram
StructureDiagram <|-- ComponentDiagram
```

4 Class Diagram — UML diagrams



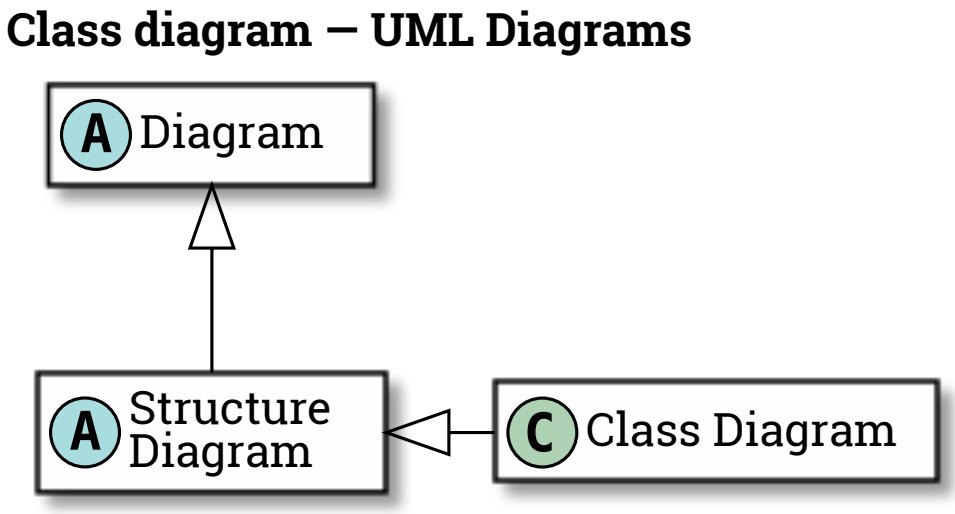
4 Class Diagram — UML diagrams

```
skinparam TitleFontStyle Bold
skinparam ArrowColor Black
skinparam ClassBorderColor Black
skinparam ClassBackgroundColor White
hide empty members
title Class diagram - UML Diagrams

abstract class "Diagram" as diagram
abstract class "Structure\lDiagram" as structure
class "Class Diagram" as class

diagram <|-- structure
structure <|-right- class
```

4 Class Diagram — UML diagrams



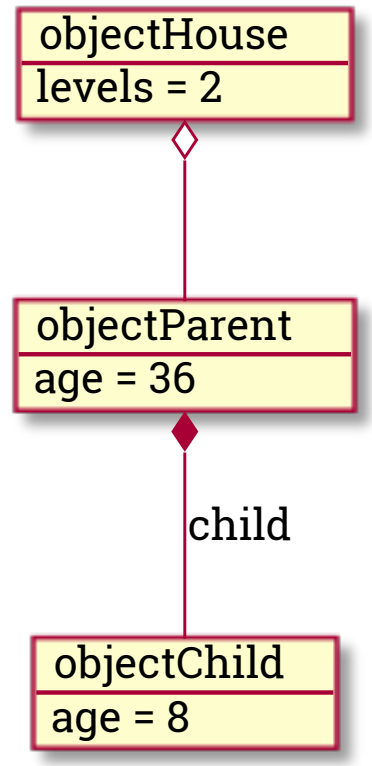
5 Object Diagram — Simple example

overview of users in relation to functional requirements in usecases

```
object objectHouse
object objectParent
object objectChild

objectHouse o-- objectParent
objectParent *-- objectChild : child

objectHouse : levels = 2
objectParent : age = 36
objectChild : age = 8
```



5 Object Diagram — Exercise

Exercise 5/5:

Make a minimal and optimized object diagram of my red car that is composed of:

- wheel left front
- wheel right front
- wheel left rear
- wheel right rear

Create object according to the class diagram related vehicle discussed earlier.

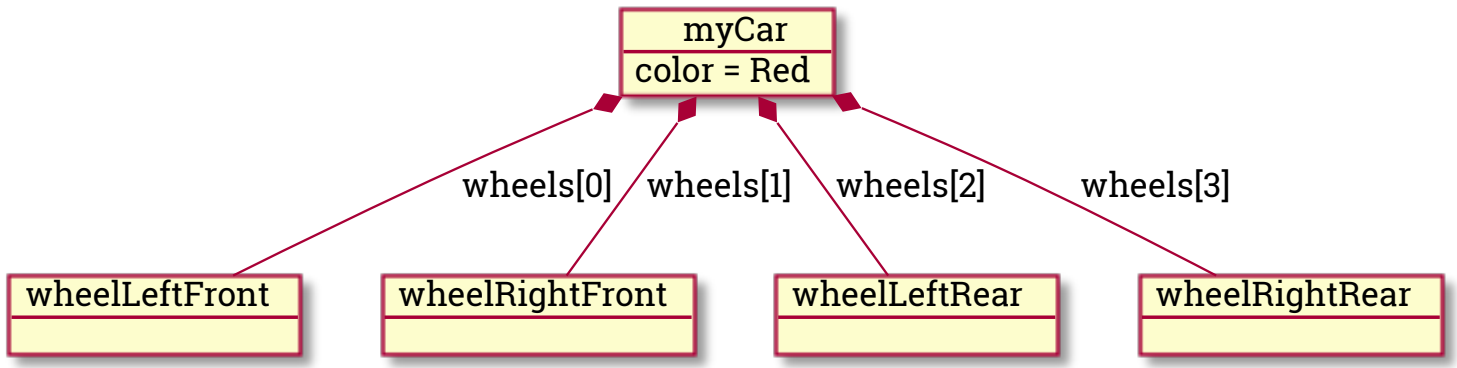
5 Object Diagram — Car

```
object myCar
myCar : color = Red

object wheelLeftFront
object wheelRightFront
object wheelLeftRear
object wheelRightRear

myCar *-- wheelLeftFront : wheels[0]
myCar *-- wheelRightFront : wheels[1]
myCar *-- wheelLeftRear : wheels[2]
myCar *-- wheelRightRear : wheels[3]
```

5 Object Diagram — Car



5 Object Diagram — Car

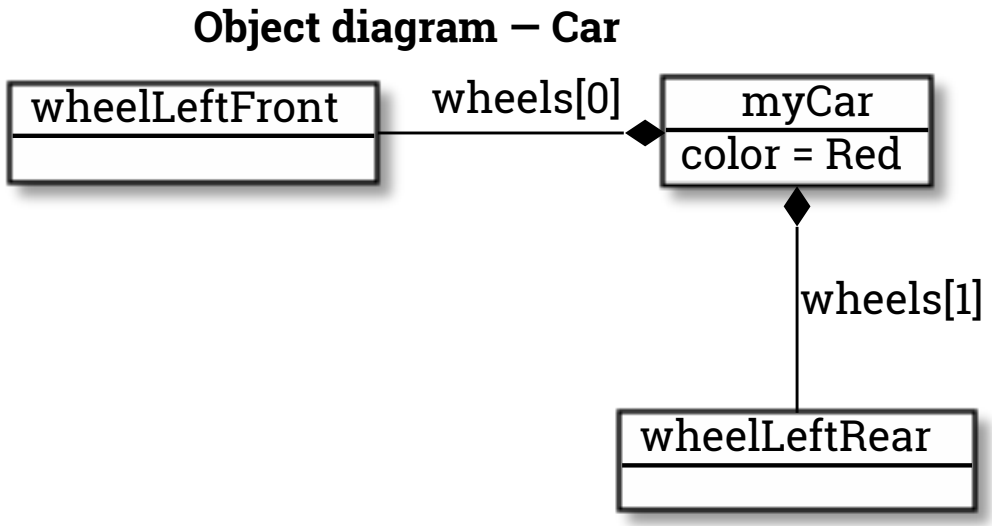
```
skinparam TitleFontStyle Bold
skinparam ArrowColor Black
skinparam ObjectBorderColor Black
skinparam ObjectBackgroundColor White
title Object diagram - Car

object myCar {
    color = Red
}

object wheelLeftFront
object wheelLeftRear

myCar *-left- wheelLeftFront : wheels[0]
myCar *-- wheelLeftRear : wheels[1]
```

5 Object Diagram — Car



See also

github.com/PanderMusubi/plantuml-workshop

Some other of my FOSS projects:

- Dutch keyboard for Android
  github.com/opentaal/LanguagePack/tree/Dutch
- Dutch garbage collection ICS calendars
  github.com/PanderMusubi/afvalophaaldata
- Dutch holiday ICS calendars
  github.com/PanderMusubi/dutch-holidays
- Compose Key Sequence Reference Guide
  amazon.com/Compose-Sequence-Reference-Guide-2012/dp/1468141104
- English locale for the Netherlands
  github.com/PanderMusubi/locale-en-nl