PlantUML Code der Grafiken

Inhalt

[1 Aufbau des Dokuments 2](#_Toc460502200)

[2 Klassendiagramm 6](#_Toc460502201)

[3 Anwendungsfalldiagramm 11](#_Toc460502202)

[4 Aktivitätsdiagramm 16](#_Toc460502203)

[5 Zustandsdiagramm 27](#_Toc460502204)

[6 Sequenzdiagramm 37](#_Toc460502205)

[7 Anhang Beispiellösungen 38](#_Toc460502206)

# Aufbau des Dokuments

Im nachfolgenden Dokument werden die Grafiken des Schulungsdokumentes als PlantUML Syntax dargestellt. Die Kapitelüberschriften sind dabei die Abbildungsnummern wie diese bei den Grafiken im Schulungsdokument vorkommen. Darunter folgt der PlantUML-Code. Die Grafiken sind in der gleichen Reihenfolge angeordnet wie im Schulungsdokument und sind kapitelweise nach Diagrammtyp geordnet: Kapitel 2 enthält alle Grafiken des Klassendiagramms, Kapitel 3 des Anwendungsfalldiagramms, Kapitel 4 des Aktivitätsdiagramms, Kapitel 5 des Zustandsdiagramms und Kapitel 6 des Sequenzdiagramms. Im Anschluss sind in Kapitel 7 die Beispiellösungen der Modellierungsaufgabe dargestellt. Dieses Dokument dient nicht als Einführung in PlantUML. Es wird davon ausgegangen, dass Leser die Syntax von PlantUML kennen, weshalb diese innerhalb der einzelnen Diagramme nicht näher erklärt wird. Für eine bessere Übersicht werden die Start-und Endtags (@startuml und @enduml) im Beispielcode weggelassen. Da in PlantUML Zeilenumbrüche semantisch relevant sind, ist bei Zeilen, welche aufgrund der Textsetzung umgebrochen werden mit zwei Slashzeichen + kurzer Text angemerkt, dass diese Zeile zur vorhergehenden gehört

(// mit voriger Zeile). Soweit möglich wurde ein Diagramm (oder mehrere kurze Diagramme) auf einer Seite abgebildet. Einige Diagramme verteilen sich jedoch über zwei Seiten. Das gesamte Dokument ist ebenfalls in Schwarzschrift gedruckt.

Diese Mappe wurde erstellt vom Studienzentrum für Sehgeschädigte des Karlsruher Instituts für Technologie in Karlsruhe, 2016.

# Klassendiagramm

## Abbildung 1

class Klassenname{

Attribut

Methoden()

}

class Buch{

titel: String

isbn: Int

}

## Abbildung 2

class Buchhandlung - Kunde

## Abbildung 3

class Verkäufer <-> Kunde

## Abbildung 4

class Bestellung -> Buch

## Abbildung 5

class Buchhandlung \*- Raum

## Abbildung 6

class Buchhandlung o- Buch

## Abbildung 7

class Verkäufer "+Bedienung" - Kunde

## Abbildung 8

class Verkäufer - Kunde: berät >

## Abbildung 9

class Regal "1"”- "0..20" Buch

## Abbildung 10

class Verkäufer -|> Mitarbeiter

Abteilungsleiter -|> Mitarbeiter

## Abbildung 11

class NaviGerät

class Route{

Länge:Double

Dauer: Double

}

class Routenabschnitt{

Straßenname: String

Anfangsposition: Position

Endposition: Position

}

NaviGerät "\*" - "\*" Route: berechnet

Route \*- Routenabschnitt

# Anwendungsfalldiagramm

## Abbildung 12

(Buch kaufen)

## Abbildung 13

Ein maschineller Akteur als Rechteck kann in PlantUML nicht erstellt werden. Es können jedoch Stereotype verwendet werden.

Ohne Stereotyp:

actor Kunde

Mit Stereotyp:

actor Kunde <<Machineller Akteur>>

## Abbildung 14

actor Kunde

rectangle Buchhandlung {

Kunde - (Buch kaufen)

}

## Abbildung 5

actor Verkäufer

actor Kunde

rectangle Buchhandlung{

Verkäufer - "2" Buch kaufen)

Kunde - (Buch kaufen)

}

## Abbildung 16

(Buch kaufen) .> (Bezahlen): <<include>>

## Abbildung 17

(Als Geschenk verpacken).>(Buch kaufen): <<extends>> // mit Zeile vorher

## Abbildung 18

(Mit Kreditkarte zahlen) -|> (Bezahlen)

## Abbildung 19

actor Abteilungsleiter

Abteilungsleiter -|> Mitarbeiter

## Abbildung 20

actor Abteilungsleiter

actor Verkäufer

Abteilungsleiter -|> Mitarbeiter

Verkäufer -|> Mitarbeiter

rectangle Buchhandlung {

Mitarbeiter - (Buch kaufen)

}

## Abbildung 21

actor Person <<Human>>

:GPS-Stallitensystem: as GPS <<Server>>

rectangle Navigationsapp{

Person - (Zum Ziel navigieren)

Person - (Zielort eingeben)

(Zum Ziel navigieren) .> (Zielort eingeben): <<include>> //mit Zeile vorher

(Zum Ziel navigieren) .> (Aktuelle Position): <<include>> //mit Zeile vorher

GPS - (Aktuelle Position)

}

# Aktivitätsdiagramm

## Abbildung 22

:Buch auswählen;

## Abbildung 23

:Buchhandlung betreten;

:Buch auswählen;

## Abbildung 24

start

## Abbildung 25

end

## Abbildung 26

stop

## Abbildung 27

Aktivitäten können nicht modelliert werden: innen gelegene Aktionsfolge:

start

:Buchhandlung betreten;

:Buch auswählen;

:Buch bezahlen;

:Buchhandlung verlassen;

Stop

## Abbildung 28

Bedingungen können nicht innerhalb einer Aktion/Aktivität realisiert werden, sie können jedoch über Notizen, welche rechts angeordnet werden simuliert werden.

start

:Sachbuch kaufen;

note right: <<precondition>> Wissenslücke

:Sachbuch lesen;

note right: <<postcondition>> neues Wissen

stop

## Abbildung 29

if(Genre wählen) then (Fantasy)

:Zustand 1;

elseif (Krimi)

:Zustand 2;

else (sonst)

:Zustand 3;

endif

## Abbildung 30

start

:Buch suchen;

if() then (Fantasy)

:Tolkien wählen;

else (sonst)

:Bestseller wählen;

endif

:Buch kaufen;

stop

## Abbildung 31

Diese Syntax entspricht nicht exakt der Syntax von UML, da Rückwärtsschleifen nicht auf Aktionen weisen können in PlantUML. Daher ist die erste Aktion direkt mit in der While-Schleife enthalten. Programmiertechnisch können nur while- und nicht do-while-Schleifen konstruiert werden.

start

while ( ) is ([Summe < Gutscheinwert])

:Buch wählen;

endwhile ([sonst])

:Bezahlen;

Stop

## Abbildung 32

fork

:Option 1;

fork again

:Option 2;

end fork

## Abbildung 33

start

:Buchhandlung betreten;

fork

:Umsehen;

fork again

:Kaffee trinken;

end fork

:Buchhandlung verlassen;

stop

## Abbildung 34

|Kunde|

start

:Buch suchen;

|Mitarbeiter|

:Regal zeigen;

|Kunde|

:kaufen;

|Mitarbeiter|

## Abbildung 35

start

fork

:Aktuelle Position ermitteln;

fork again

:Zielort eingeben;

if() then (Abkürzungen einbeziehen)

:Wege einbeziehen;

else (sonst)

endif

end fork

:Route berechnen;

:Route ausgeben;

stop

# Zustandsdiagramm

## Abbildung 36

state "In Buchhandlung" as buch

buch: entry/türÖffnen()

buch: do/umschauen()

buch: exit/türSchließen()

## Abbildung 37

[\*] ->

## Abbildung 38

-> [\*]

## Abbildung 39

Momentan nicht möglich mit PlantUML

## Abbildung 40

state "Buch nicht vorhanden" as weg

state "Buch vorhanden" as da

weg -> da: bestellen

## Abbildung 41

state "Buch nicht vorhanden" as weg

state "Buch vorhanden" as da

weg -> da: bestellen[verfügbar]

weg -> weg: bestellen[nicht verfügbar]

## Abbildung 42

state "Buch nicht vorhanden" as weg

state "Buch vorhanden" as da

weg -> da: bestellen[verfügbar]

weg -> weg: bestellen[nicht verfügbar]/drucken() // mit Zeile vorher

## Abbildung 43

Kein UML-Diagramm, daher nicht mit PlantUML umsetzbar.

## Abbildung 44

state Offen{

state Arbeit

state Pause

}

state Geschlossen{

state Verlassen

state Inventur

}

## Abbildung 45

state Offen{

[\*]-> Arbeit

Arbeit -> Pause

}

Geschlossen -> Offen

## Abbildung 46

state Offen{

[\*]-> Arbeit

Arbeit -> Pause

Pause -> Offen

}

## Abbildung 47

state Offen{

[\*]-> Arbeit

Arbeit -> Pause

}

Geschlossen -> Pause

## Abbildung 48

state Offen{

[\*]-> Arbeit

Arbeit -> Pause

Arbeit -> [\*]

}

Offen -> Geschlossen

## Abbildung 49

state Offen{

[\*]-> Arbeit

Arbeit -> Pause

Arbeit -> [\*]

}

Offen -> Abgebrannt: Feuerausbruch/rausgehen //mit Zeile vorher

## Abbildung 50

state Offen{

[\*]-> Arbeit

Arbeit -> Pause

Arbeit -> [\*]

}

Pause -> Feierabend

## Abbildung 51

state "In Buchhandlung" as buchhandlung{

[\*] -> Suchend

Suchend -> [\*]

--

[\*] -> Gehend

Gehend -> [\*]

}

## Abbildung 52

state "Navi Aus" as NaviAus

state "Navi Ein" as NaviEin

[\*]-> NaviAus

NaviAus -> NaviEin: ein

NaviEin -> NaviAus: aus

state NaviEin{

state "kein GPS Signal" as keinGPSSignal

state "GPS Signal" as GPSSignal

[\*] -> keinGPSSignal

keinGPSSignal -> GPSSignal: GPS empfangen //mit Zeile vorher

GPSSignal -> keinGPSSignal: kein GPS Signal erreichbar/Hinweis ausgeben //mit Zeile vorher

}

# Sequenzdiagramm

## Abbildung 53

## actor "Alice:Kunde" as Alice

## participant”":System" as System

## Abbildung 54

actor "Alice:Kunde" as Alice

actor "Bob:Kunde" as Bob

actor "Carol:Kunde" as Carol

activate Alice

activate Bob

activate Bob

activate Carol

deactivate Alice

deactivate Bob

deactivate Bob

destroy Carol

Abbildung 55

actor ":Kasse" as Kasse

hnote over Kasse: {befüllt==true}

## Abbildung 56

actor ":Kunde" as Kunde

actor ":Mitarbeiter" as Mitarbeiter

Kunde -> Mitarbeiter: synchron rufen

Kunde ->> Mitarbeiter: asynchron rufen

## Abbildung 57

actor "Alice:Kunde" as Alice

activate Alice

Alice -> Alice: Buch wählen

activate Alice

## Abbildung 58

actor ":Kunde" as Kunde

actor ":Mitarbeiter" as Mitarbeiter

Kunde -> Mitarbeiter: rufen

Mitarbeiter -->> Kunde: rufen

## Abbildung 59

actor ":Kunde" as Kunde

actor ":Mitarbeiter" as Mitarbeiter

[o-> Kunde: Geschenk kaufen

activate Kunde

Kunde ->> Mitarbeiter: Beratung anfordern

activate Mitarbeiter

Mitarbeiter ->o]: ignorieren

## Abbildung 60

Schräge Nachrichtenverbindungen sind in PlantUML nicht möglich. Eine Zeitspanne mit der Dauer kann jedoch modelliert werden. Die schräge Verbindung ist in folgendem Code deshalb nicht enthalten.

actor ":Kunde" as Kunde

actor ":Mitarbeiter" as Mitarbeiter

activate Kunde

Kunde -> Mitarbeiter: rufen d=dauer

activate Mitarbeiter

deactivate Mitarbeiter

Kunde -> Mitarbeiter: rufen{0..1sec}

activate Mitarbeiter

deactivate Mitarbeiter

Kunde -> Mitarbeiter: rufen

activate Mitarbeiter

deactivate Mitarbeiter

...{0..5min}...

Kunde -> Mitarbeiter: anschreien

activate Mitarbeiter

deactivate Mitarbeiter

## Abbildung 61

actor ":Kunde" as Kunde

actor ":Mitarbeiter" as Mitarbeiter

activate Kunde

Kunde -> Mitarbeiter: rufen

activate Mitarbeiter

deactivate Mitarbeiter

Kunde -> Mitarbeiter: Preise addieren(10,8)

activate Mitarbeiter

deactivate Mitarbeiter

Kunde -> Mitarbeiter: addieren(sum1=10,sum2=8)

activate Mitarbeiter

deactivate Mitarbeiter

Kunde -> Mitarbeiter: Buch bestellen(titel=-)

activate Mitarbeiter

deactivate Mitarbeiter

Kunde -> Mitarbeiter: \*

activate Mitarbeiter

deactivate Mitarbeiter

## Abbildung 62

actor ":Kunde" as Kunde

actor ":Mitarbeiter" as Mitarbeiter

activate Kunde

Kunde -> Mitarbeiter: Preise addieren(10,8) //mit Zeile vorher

activate Mitarbeiter

Mitarbeiter -->> Kunde: Preise addieren(10,8):18 //mit Zeile vorher

deactivate Mitarbeiter

Kunde -> Mitarbeiter: Preise addieren(10,8) //mit Zeile vorher

activate Mitarbeiter

Mitarbeiter -->> Kunde: sum= Preise addieren(10,8) //mit Zeile vorher

deactivate Mitarbeiter

Kunde -> Mitarbeiter: Preise addieren (10,8, out sum) //mit Zeile vorher

activate Mitarbeiter

Mitarbeiter -->> Kunde: Preise addieren (10,8, out sum:18) //mit Zeile vorher

deactivate Mitarbeiter

Kunde -> Mitarbeiter: Preise addieren(10,8) //mit Zeile vorher

activate Mitarbeiter

Mitarbeiter -->> Kunde: Preise addieren(-,-):18 //mit Zeile vorher

deactivate Mitarbeiter

Kunde -> Mitarbeiter: \*

activate Mitarbeiter

Mitarbeiter -->> Kunde: \*

deactivate Mitarbeiter

## Abbildung 63

actor ":Autor" as Autor

create Buch

Autor ->> Buch: schreiben

Autor -> Buch: zerstören

destroy Buch

## Abbildung 64

actor ":Kunde" as Kunde

participant ":Buchhandlung" as Buchhandlung //mit Zeile vorher

activate Kunde

Kunde -> Buchhandlung: Buchserie suchen

activate Buchhandlung

Buchhandlung -->> Kunde: Buchserie suchen

alt Buch vorhanden

Kunde -> Buchhandlung: kaufen

else sonst

Kunde -> Buchhandlung: bestellen

end

## Abbildung 65

actor ":Kunde" as Kunde

participant ":Buchhandlung" as Buchhandlung //mit Zeile vorher

opt Empfehlung==gut

Kunde -> Buchhandlung: kaufen

end

## Abbildung 66

actor ":Abteilungsleiter" as Leiter

actor "MA1:Mitarbeiter" as m1

actor "MA2:Mitarbeiter" as m2

par

Leiter ->> m1: 1.1 Bücher einräumen

activate Leiter

activate m1

deactivate m1

Leiter ->> m2: 1.2 Bücher einräumen

activate m2

deactivate m2

deactivate Leiter

else

m1-> m2: 2.1 Bücher diskutieren

activate m1

activate m2

m2-->> m1: 2.2 Bücher diskutieren

deactivate m1

deactivate m2

end

## Abbildung 67

actor "MA1:Mitarbeiter" as m1

actor "MA2:Mitarbeiter" as m2

participant ":Ambulanz" as Ambulanz

m2->> m1: Unfall melden

activate m1

activate m2

deactivate m2

critical

m1 -> m1: Erste Hilfe leisten(MA2)

activate m1

deactivate m1

m1-> Ambulanz: anfordern

activate Ambulanz

Ambulanz -->> m1

deactivate Ambulanz

deactivate m1

end

## Abbildung 68

actor ":Abteilungsleiter"

actor "MA:Mitarbeiter" as Mitarbeiter

loop MA.einräumen!=true

Abteilungsleiter-> Mitarbeiter: einräumen(Buch)\\mit Zeile vorher

activate Abteilungsleiter

activate Mitarbeiter

deactivate Abteilungsleiter

deactivate Mitarbeiter

end

## Abbildung 69

actor ":Person" as Person

participant ":Navigerät" as Navigerät

participant ":GPS-System" as GPS

activate Person

Person -> Navigerät: Ziel bestimmen

activate Navigerät

Navigerät -> GPS: Aktuelle Position ermitteln //mit Zeile vorher

activate GPS

GPS -->> Navigerät: Aktuelle Position ermitteln: Pos(X,Y) //mit Zeile vorher

destroy GPS

Navigerät -->> Person: Ziel bestimmen

destroy Navigerät

loop

Person -> Person: Zum Ziel navigieren

activate Person

deactivate Person

deactivate Person

end

# Anhang Beispiellösungen

## Abbildung A.1.

class Arznei{

Name: String

heilen()

}

class Krankheit{

Name: String

}

class Patient{

Name: String

}

Arzt -|> Patient

Privatperson -|> Patient

Arznei "\*" - "1..\*" Krankheit

Krankheit "\*" - "\*" Patient

Patient "\*" - "0..50" Arznei: kauft >

## Abbildung A.2.

actor Kunde

actor Apotheker

Arzt -|> Kunde

Privatperson -|> Kunde

rectangle Apotheke{

(Verkauf) .> (Beratung): <<include>>

(Arznei herstellen) - "2" Apotheker

(Arznei bestellen) - Apotheker

(Verkauf) - Apotheker

(Verkauf) - Kunde

}

## Abbildung A.3.

start

:Apotheke besuchen;

:Krankheit beschreiben;

:Naturprodukt wählen;

if() then (Naturprodukt vorrätig)

:Produkt kaufen;

else (Naturprodukt nicht vorrätig)

fork

:Standardprodukt kaufen;

fork again

:Naturprodukt bestellen;

end fork

endif

:Apotheke verlassen;

stop

## Abbildung A.4.

Dieses Diagramm ist ohne bedingte Verzweigung modelliert, da diese in Zustandsdiagrammen momentan nicht möglich sind. Im Gegenzug dazu wurden die Zustandsübergänge detaillierter beschriftet. Der Code ist auf der nächsten Seite.

[\*] -> Gesund

Gesund -> Erkältet: Im Regen spazieren

Erkältet -> Gesund: ausruhen

Erkältet -> Krank: arbeiten

Krank -> [\*]

Krank -> Gesund: Arznei nehmen [wirkt]

Krank -> Krank: Arznei nehmen[wirkt nicht]

## Abbildung A.5.

actor ":Kunde"” as Kunde

actor ":Apotheker"” as Apotheker

participant ":Apotheke"” as Apotheke

Kunde -> Apotheker: Rezept

activate Kunde

activate Apotheker

loop Artikel!=0

Apotheker -> Apotheke: Medikament holen

activate Apotheke

Apotheke -->> Apotheker: \*

destroy Apotheke

end

Apotheker ->o]: Fehlende Medikamente bestellen \\ mit Zeile vorher

Apotheker -->> Kunde: Rezept: Medikamente geben \\ mit Zeile vorher