cooperate

Abbildungsverweise PlantUML Code

Version 1.0

Vanessa Petrausch

Projektpartner:

Karlsruher Institut für Technologie

FZI

Gefördert durch:

Bundesministerium für Arbeit und Soziales

aus Mitteln des Ausgleichsfonds
Förderkennzeichen: 01KM141108

INHALTSVERZEICHNIS

1 Aufbau des Dokuments

Im nachfolgenden Dokument werden die Grafiken des Schulungsdokumentes als PlantUML Syntax dargestellt. Die Kapitelüberschriften sind dabei die Abbildungsnamen 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 haben die identischen Nummerierungen und Beschriftungen. Um eine bessere Übersicht zu erzielen, wurden die Diagrammtypen als Überschriften gewählt. Kapitel 2 enthält alle Grafiken des Klassendiagramms, Kapitel 3 des Anwendungsfalldiagramms, Kapitel 4 des Aktivitätsdiagramms, Kapitel 5 des Zustandsdiagramms und Kaitel 6 des Sequenzdiagramms. Im Anschluss sind in Kapitel 7 die Beispiellösungen der Modellierungsaufgabe dargestellt.

Diese Dokument dient nicht als Einführung in PlantUML. Es wird davon ausgegangen, dass Leser die Syntax von PlantUML kennen, weshalb diese innerhalb der einzelnen Diagrammtypen nicht näher erklärt wird. Um eine bessere Übersicht der Diagramme zu bieten, werden die Start-und Endtags (@startuml und @enduml) im Beispielcode weggelassen. Dargestellt wird demnach nur der Code dazwischen.

2 KLASSENDIAGRAMM

Abbildung 1: Generische Darstellung einer UML-Klasse (links) und konkrete Ausprägung einer Klasse (rechts)

```
class Klassenname{
        Attribut
        Methoden()
        }
class Buch{
        titel: String
        isbn: Int
        }
```

Abbildung 2: Ungerichtete Assoziation zwischen den Klassen „Buchhandlung“ und „Kunde“

```
class Buchhandlung -- class Kunde
```

Abbildung 3: Bidirektional gerichtete Assoziation zwischen den Klassen „Verkäufer“ und „Kunde“

```
class Verkäufer <--> class Kunde
```

Abbildung 4: Gerichtete Assoziation von der Klasse „Bestellung“ zur Klasse „Buch“

```
class Bestellung --> class Buch
```

Abbildung 5: Komposition der Klassen „Buchhandlung“ und „Raum“

```
class Buchhandlung *-- class Raum
```

Abbildung 6: Aggregation zwischen den Klassen „Buchhandlung“ und „Buch“

```
class Buchhandlung o-- class Buch
```

Abbildung 7: Rollenbezeichnung, die einen „Verkäufer“ in der Beziehung zu einem „Kunden“ als „Bedienung“ identifiziert

```
class Verkäufer "+Bedienung"-- class Kunde
```

Abbildung 8: Abbildung 4.1.8: Beschriftung einer Assoziation mit „berät“ und Leserichtung von „Verkäufer“ zu „Kunde“

```
class Verkäufer -- class Kunde: berät >
```

Abbildung 9: 1:n-Beziehung zwischen der Klasse „Regal“ und der Klasse „Buch“

```
class Regal “1” -- "0..20" class Buch
```

Abbildung 10: Vererbung zwischen der Klasse „Mitarbeiter“ und den Klassen „Verkäufer“ und „Abteilungsleiter“

```
class Verkäufer --|> Mitarbeiter
Abteilungsleiter --|> Mitarbeiter
```

Abbildung 11: Klassendiagramm des Navigationsbeispiels

```
class "NaviGerät" as Navi
class Route{
        Länge:Double
        Dauer: Double
}
class "Routenabschnitt" as Abschnitt{
        Straßenname: String
        Anfangsposition: Position
        Endposition: Position
}
Navi "*" -- "*" Route: berechnet
Route *-- Abschnitt
```

3 ANWENDUNGSFALLDIAGRAMM

Abbildung 12: Ein Anwendungsfall „Buch kaufen“

(Buch kaufen)

Abbildung 13: Strichmännchen- (links) und Rechtecknotation (rechts) zur Darstellung von Akteuren

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: Darstellung eines Gesamtsystems mit einem Akteur, einem System und einem Anwendungsfall

```
actor Kunde
rectangle Buchhandlung {
        Kunde -- (Buch kaufen)
}
```

Abbildung 1: Verbindung eines Anwendungsfalls mit zwei verschiedenen Akteuren

```
actor Verkäufer
actor Kunde
rectangle Buchhandlung{
        Verkäufer – “2“(Buch kaufen)
        Kunde -- (Buch kaufen)
 }
```

Abbildung 16: Include-Assoziation vom Anwendungsfall „Buch kaufen“ zu „Bezahlen“

(Buch kaufen) .> (Bezahlen) : include

Abbildung 17: Extend-Assoziation vom Anwendungsfall „Als Geschenk verpacken“ zu „Buch kaufen“

(Als Geschenk verpacken) .> (Buch kaufen) : extends

Abbildung 18: Der Anwendungsfall „Mit Kreditkarte zahlen“ erbt vom Anwendungsfall „Bezahlen“

(Mit Kreditkarte zahlen) --|> (Bezahlen)

Abbildung 19: Der Akteur „Abteilungsleiter“ erbt vom Akteur „Mitarbeiter“

```
actor Abteilungsleiter
Abteilungsleiter --|> Mitarbeiter
```

Abbildung 20: Verhalten bei der Ausführung eines Anwendungsfalls bei Vererbung von Akteuren

```
actor Abteilungsleiter
actor Verkäufer
Abteilungsleiter --|> Mitarbeiter
Verkäufer --|> Mitarbeiter
rectangle Buchhandlung {
        Mitarbeiter -- (Buch kaufen)
}
```

Abbildung 21: Anwendungsfalldiagramm des Navigationsbeispiels

```
actor :GPS-Stallitensystem: as GPS <<Server>>
actor Person <<Human>>
rectangle Navigationsapp{
        Person - (Zum Ziel navigieren)
        Person - (Zielort eingeben)
        (Zum Ziel navigieren) .> (Zielort eingeben) : include
        (Zum Ziel navigieren) .> (Aktuelle Position) : include
        GPS - (Aktuelle Position)
}
```

4 AKTIVITÄTSDIAGRAMM

Abbildung 22: Einzelne Aktion „Buch auswählen“

```
:Buch auswählen;
```

Abbildung 23: Beispiel einer Transition zwischen zwei Aktionen „Buchhandlung betreten“ und „Buch auswählen“

```
:Buchhandlung betreten;
:Buch auswählen;
```

Abbildung 24: Startknoten

```
start
```

Abbildung 25: Flussende

```
end
```

Abbildung 26: Endknoten

```
stop
```

Abbildung 27: Beispiel der Aktivität „Buch kaufen“ bestehend aus mehreren Aktionen

Aktivitäten können nicht modelliert werden: innen gelegene Aktionsfolge:

```
Start
:Buchhandlung betreten;
:Buch auswählen;
:Buch bezahlen;
:Buchhandlung verlassen;
stop
```

Abbildung 28: Beispiel einer Aktivität mit Vor- und Nachbedingung

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: Darstellung einer bedingten Verzweigung mit drei Alternativen

```
if(Genre wählen) then (Fantasy)
        :Zustand 1;
elseif (Krimi)
        :Zustand 2;
else (sonst)
        :Zustand 3;
endif
```

Abbildung 30: Beispiel einer bedingten Verzweigung

```
start
:Buch suchen;
if() then (Fantasy)
        :Tolkien wählen;
else (sonst)
        :Bestseller wählen;
endif
:Buch kaufen;
Stop
```

Abbildung 31: Darstellung einer Schleife durch bedingte Verzweigungen

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: Darstellung der Parallelisierung und Synchronisierung von Kontrollflüssen

```
fork
        :Option 1;
fork again
        :Option 2;
end fork
```

Abbildung 33: Beispiel paralleler Kontrollflüsse

```
start
:Buchhandlung betreten;
fork
        :Umsehen;
fork again
        :Kaffee trinken;
end fork
:Buchhandlung verlassen;
Stop
```

Abbildung 34: Beispiel von zwei Aktivitätsbereichen "Kunde" und "Mitarbeiter"

```
|Kunde|
start
:Buch suchen;
|Mitarbeiter|
:Regal zeigen;
```

```
|Kunde|
:kaufen;
|Mitarbeiter|
```

Abbildung 35: Aktivitätsdiagramm des Navigationsbeispiels

```
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
```

5 ZUSTANDSDIAGRAMM

Abbildung 36: Zustand „In Buchhandlung“ mit drei internen Aktionen

```
state "In Buchhandlung" as buch
buch: entry/türÖffnen()
buch: do/umschauen()
buch: exit/türSchließen()
```

Abbildung 37: Startzustand

```
[*] -->
```

Abbildung 38: Endzustand

```
--> [*]
```

Abbildung 39: Terminator

Momentan nicht möglich mit PlantUML

Abbildung 40: Transition zwischen zwei Zuständen mit einem Event

```
state "Buch nicht vorhanden" as weg
state "Buch vorhanden" as da
weg --> da: bestellen
```

Abbildung 41: Transition zwischen zwei Zuständen mit Events und Guards

```
state "Buch nicht vorhanden" as weg
state "Buch vorhanden" as da
weg --> da: bestellen[verfügbar]
weg --> weg: bestellen[nichtverfügbar]
```

Abbildung 42: Transition zwischen zwei Zuständen mit Event, Guard und Aktion

```
state "Buch nicht vorhanden" as weg
state "Buch vorhanden" as da
weg --> da: bestellen[verfügbar]
weg --> weg: bestellen[nichtverfügbar]/drucken()
```

Abbildung 43: Darstellung sieben hierarchischer Zustände als Baum

Kein UML-Diagramm, daher nicht mit PlantUML umsetzbar

Abbildung 44: Beispielhafte Darstellung der hierarchischen Zustände des Baumes als Zustandsdiagramm

```
state Offen{
        state Arbeit
        state Pause
}
state Geschlossen{
        state Verlassen
        state Inventur
}
```

Abbildung 45: Standardeintritt in einen zusammengesetzten Zustandes

```
state Offen {
        [*]--> Arbeit
        Arbeit --> Pause
}
Geschlossen --> Offen
```

Abbildung 46: Zweite Variante des Standard-Eintritts

```
state Offen {
        [*]--> Arbeit
        Arbeit --> Pause
Pause -> Offen
}
```

Abbildung 47: Explizites Betreten eines Unterzustands in einem zusammengesetzten Zustand

```
state Offen {
        [*]--> Arbeit
        Arbeit --> Pause
}
Geschlossen -> Pause
```

Abbildung 48: Verlassen eines zusammengesetzten Zustands mithilfe eines modellierten Endzustand

```
state Offen {
        [*]--> Arbeit
        Arbeit --> Pause
        Arbeit -> [*]
}
Offen -> Geschlossen
```

Abbildung 49: Verlassen eines zusammengesetzten Zustands mithilfe eines Events

```
state Offen {
        [*]--> Arbeit
        Arbeit --> Pause
        Arbeit -> [*]
}
Offen -> Abgebrannt: Feuerausbruch/rausgehen
```

Abbildung 50: Direktes Verlassen eines zusammengesetzten Zustands

```
state Offen {
        [*]--> Arbeit
        Arbeit --> Pause
        Arbeit -> [*]
}
Pause -> Feierabend
```

Abbildung 51: Orthogonaler Zustand „In Buchhandlung“ mit zwei Regionen „Geist“ und „Körper“

```
state "In Buchhandlung" as buchhandlung {
        [*] -> Suchend
        Suchend -> [*]
        --
        [*] -> Gehend
        Gehend -> [*]
}
```

Abbildung 52: Zustandsdiagramm des Navigationsbeispiels

```
state "Navi Aus" as NaviAus
state "Navi Ein" as NaviEin
[*] --> NaviAus
NaviEin --> NaviAus: aus
NaviAus --> NaviEin: ein
state NaviEin{
        state "kein GPS Signal" as keinGPSSignal
        state "GPS Signal" as GPSSignal
        [*] --> keinGPSSignal
        keinGPSSignal --> GPSSignal: GPS empfangen
        GPSSignal -->keinGPSSignal: kein GPS Signal\n erreichbar/ Hinweis \n ausgeben
}
```

6 SEQUENZDIAGRAMM

Abbildung 53: Die Zeit- und Interaktionsachse mit zwei Teilnehmern

Die Doppelpunktnotation kann in PlantUML nur in Verbindung mit einem Alias verwendet werden, da ein Doppelpunkt die Beschriftung einer Assoziation einleitet. Das Schlüsselwort „actor“ erzeug einen menschlichen Teilnehmer als Strichmännchen, das Schlüsselwort „participant“ ein Rechteck.

```
actor ”Alice:Kunde“ as Alice
participant ”:System” as System
```

Abbildung 54: Drei Varianten der Lebenslinien mit Ausführungsbalken und Beendigung

Das Schlüsselwörter „activate“ beginnt einen Ausführungsbalken einer Lebenslinie, „deactivate“ beendet diesen und „destroy“ beendet ihn und zerstört die Lebenslinie. Eine doppelte Aktivierung schafft geschachtelte Ausführunsbalken.

```
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: Die Zustandsinvariante von Lebenslinien

Invarianten können nur mithilfe von Notizen realisiert werden. Diese werden mit folgender Syntax direkt auf die Lebenslinie gelegt.

```
actor “:Kasse“ as Kasse
hnote over Kasse: {befüllt==true}
```

Abbildung 56: Darstellung synchroner und asynchroner Kommunikation zwischen einem „Kunden“ und einem „Mitarbeiter“

```
actor “:Kunde“ as Kunde
actor “:Mitarbeiter“ as Mitarbeiter
Kunde -> Mitarbeiter: synchron rufen
Kunde ->> Mitarbeiter: asynchron rufen
```

Abbildung 57: Reflexive Nachricht "Buch auswählen"

```
actor “Alice:Kunde” as Alice
```

```
activate Alice
Alice -> Alice: Buch wählen
activate Alice
```

Abbildung 58: Die Antwortnachricht

```
actor ":Kunde" as Kunde
actor ":Mitarbeiter" as Mitarbeiter
Kunde -> Mitarbeiter: rufen
Mitarbeiter -->> Kunde: rufen
```

Abbildung 59: Gefundene und verlorene Nachrichten

```
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: Modellierung von Zeitbedingungen von Nachrichten

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: Argumentübertragung bei Nachrichten

```
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: Rückgabewerte von Nachrichten mit Argumenten

```
actor ":Kunde" as Kunde
actor ":Mitarbeiter" as Mitarbeiter
activate Kunde
Kunde -> Mitarbeiter: Preise addieren(10,8)
activate Mitarbeiter
Mitarbeiter -->> Kunde: Preise addieren(10,8):18
deactivate Mitarbeiter
Kunde -> Mitarbeiter: Preise addieren(10,8)
activate Mitarbeiter
Mitarbeiter -->> Kunde: sum= Preise addieren(10,8)
deactivate Mitarbeiter
Kunde -> Mitarbeiter: Preise addieren(10,8, out sum)
activate Mitarbeiter
Mitarbeiter -->> Kunde: Preise addieren(10,8, out sum:18)
deactivate Mitarbeiter
Kunde -> Mitarbeiter: Preise addieren(10,8)
activate Mitarbeiter
Mitarbeiter -->> Kunde: Preise addieren(-,-):18
deactivate Mitarbeiter
Kunde -> Mitarbeiter: *
activate Mitarbeiter
Mitarbeiter -->> Kunde: *
deactivate Mitarbeiter
```

Abbildung 63: Erstellung und Zerstörung eines „Buch"-Objekts

```
actor ":Autor" as Autor
create Buch
Autor ->> Buch: schreiben
Autor -> Buch: zerstören
destroy Buch
```

Abbildung 64: Darstellung von alternativen Nachrichtenflüssen

```
actor ":Kunde" as Kunde
participant ":Buchhandlung" as Buchhandlung
```

```
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: Optionale Nachrichtenflüsse anhand von Bedingungen

```
actor “:Kunde“ as Kunde
participant “:Buchhandlung“ as Buchhandlung
opt Empfehlung==gut
        Kunde -> Buchhandlung: kaufen
end
```

Abbildung 66: Darstellung von parallelen Nachrichtenflüssen

```
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: Darstellung eines nicht unterbrechbaren Bereichs "critical"

```
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: Mehrfache Durchführung von Interaktionen: die Schleife

```
actor ":Abteilungsleiter"
actor "MA:Mitarbeiter" as Mitarbeiter
loop MA.einräumen!=true
        Abteilungsleiter-> Mitarbeiter: einräumen(Buch)
        activate Abteilungsleiter
        activate Mitarbeiter
        deactivate Abteilungsleiter
        deactivate Mitarbeiter
end
```

Abbildung 69: Sequenzdiagramm des Navigationsbeispiels

```
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
activate GPS
GPS -->> Navigerät: Aktuelle Position ermitteln: Pos(X,Y)
destroy GPS
Navigerät -->> Person: Ziel bestimmen
destroy Navigerät
loop
        Person -> Person: Zum Ziel navigieren
        activate Person
        deactivate Person
        deactivate Person
end
```

7 Anhang Beispielslösungen

Abbildung A.1. Lösungsvorschlag für das Klassendiagramm

```
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. Lösungsvorschlag für das Anwendungsfalldiagramm

```
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. Lösungsvorschlag für das Aktivitätsdiagramm

```
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. Lösungsvorschlag für das Zustandsdiagramm

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

```
[*] -> 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. Lösungsvorschlag für das Sequenzdiagramm

```
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
Apotheker -->> Kunde: Rezept: Medikamente geben
```