Dokumentation Clara Tschamon

Verwendung der Konsole siehe README.md

Dokumentation der gewählen Architektur siehe .drawio Datei. Für den Fall, dass Sie draw.io nicht installiert haben, habe ich die Grafik noch als png abgegeben.

Angewandte interaction patterns:

- Zwischen TemperatureEnvironmentSimulator und TemperatureSensor
  Temperatursensor kennt TemperatureEnvironmentSimulator und fragt jede Sekunde die aktuelle Temperatur an. **(=Request Response)**

  So gemacht weil:
  Wenn ich tell-and-forget gewählt hätte, dann müsste ich den Code ändern, wenn ein neuer Sensor hinzukommen würde, der auch Informationen vom Environment bräuchte. Dann müsste also der Actor Environment alle Sensoren kennen („Subscribers“), so müssen aber alle Sensoren ihr Environment kennen, das sie abfragen wollen.

- Zwischen Temperature Sensor und AirCondition
  Nach erhalt der aktuellen Temperatur sendet der TemperatureSensor die Temperatur an die AirCondition **(=Fire and Forget)**.

- Zwischen WeatherEnvironmentSimulator und WeatherSensor:
  WeatherSensor kennt WeatherEnvironmentSimulator und fragt jede Sekunde die aktuelle Temperatur an **(=Request Response)**

- Zwischen WeatherSensor und Blinds
  Im Gegenteil zum Temperatursensor kennt der Weathersensor seinen aktuellen Wert.
  Nur wenn das Wetter geändert wurde, werden die Blinds benachrichtigt. Hierbei sagt der WeatherSensor den Blinds ob das Wetter sonnig ist oder nicht. **(=Fire and Forget)**
  Die Blinds können anhand dieser Information entscheiden, ob sie sich schließen oder öffnen. Wenn beispielsweise ein Film läuft aber das Wetter von sonnig auf schlecht wechselt, sollen sie sich nicht öffnen.

- Zwischen UI und AirCondition
  Tell and Forget

- Zwischen UI und TemperatureEnvironmentSimulator
  Tell and Forget

- Zwischen UI und MediaPlayer
  Tell and Forget

- Zwischen UI und Fridge
  Tell and Forget

- Zwischen Fridge und OrderProcessor
  Der Order Processor ist ein **Per session child Actor**.

Direkt bei erstellen des Order Processors wird anfrage an FridgeSpaceSensor und FridgeWeightSensor gesendet und sobald diese erhalten wurde, analysiert Order Processor ob Order gemacht werden kann und falls ja, informiert dieser den Fridge mit **Tell and Forget**

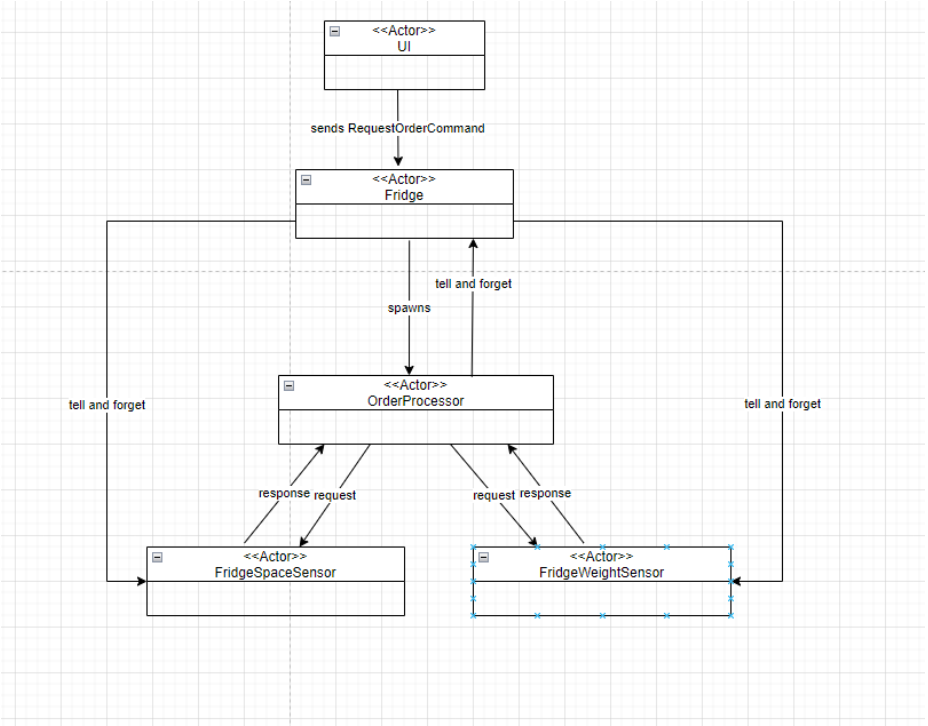
- Zwischen Order Processor und FridgeSpaceSensor&FridgeWeightSensor
  Request-Response

- Zwischen Fridge und FridgeWeightSensor&FridgeSpaceSensor
  Wenn Order gemacht wird oder Product konsumiert wird, informiert der Fridge seine Sensoren mit **Tell and Forget** darüber, dass produkt dazu gekommen ist, damit diese ihre werte aktualisieren können.

Scenario: Change Temperature
Im TemperatureEnvironment löst ein Timer alle x sekunden ein AutomaticTemperatureChangeCommand aus. Somit wird die Temperatur des TemperatureEnvironmentSimulators alle x Sekunden um 0 bis 1°C geändert.

Alle y Sekunden sendet der TemperatureSensor einen Request an die TemperatureEnvironment und bekommt die aktuelle Temperatur als Anwort.
Diese wird sofort mit Tell and Forget an die AirCondition weitergeleitet.
Wenn die Aircondition nicht eingeschaltet ist, ignoriert die AirCondition diese Information.
Ansonsten überprüft sie, ob die Temperatur über den schwellwert 20°C geändert hat oder darunter gefallen ist. Jenachdem wird sie aktiviert oder ausgeschaltet.

Scenario: Order Product
Von UI aus wird an den Fridge ein RequestOrderCommand gesendet. Dieser spawned einen OrderProcessor für jeden RequestOrderCommand. Referenzen des WeightSensors und SpaceSensors werden übergeben.
In Konstruktor des Order Processors wird ein Request an den SpaceSensor und den WeightSensor gesendet.
Hier: Per session Child Pattern:
Wenn beispielsweise jetzt der WeightSensor schon geantwortet hat, der WeightSensor aber noch nicht, wird das Selbe Behavior returned. Sobald beide geantwortet haben, wird der Fridge bei positiver darüber informiert, dass die Order gemacht werden kann (falls möglich).
Dann wird das Behavior des OrderProcessors gestoppt.
Der Fridge kann also nun das Product zu den current Products hinzufügen und die Sensoren informieren, dass sich space und weight geändert haben. Außerdem nimmt der Fridge die Bestellung in die OrderHistory auf gibt das Reciept aus.



Anmerkung:
Wenn ich das Behavior der AirCondition auf ausgeschaltet geändert habe und für den ChangedTemperatureCommand keine Methode fetgelegt habe, habe ich in der Konsole die Meldung „unhandled command“ oder so ähnlich bekommen weil ja der Temperatursensor trotzdem commands sendet. Das habe ich ursprünglich so gelöst, dass ich die onIgnoreMessage Methode festgelegt habe im augeschalteten zustand.
Dann hat mir aber Ida gezeigt, dass dies nicht nötig ist, sondern stattdessen diese 2 Zeilen ins application.conf file geschrieben werden können.

AirCondition.java application.conf

```
akka {
  log-dead-letters-during-shutdown = off
  log-dead-letters = 0
}
```


AirCondition.java
10 3 8
this.poweredOn = false;
return Behaviors.receive(AirConditionCommand.class)
//.onMessage(ChangedTemperatureCommand.class, this::onIgnoreMessage)
.onMessage(PowerAirConditionCommand.class, this::onPowerAirConditionOn)
.onSignal(PostStop.class, signal -> onPostStop())
.build();
}
1 usage new *
private Behavior<AirConditionCommand> powerOn() {
poweredOn = true;
return Behaviors.receive(AirConditionCommand.class)
.onMessage(ChangedTemperatureCommand.class, this::onReadTemperature)
.onMessage(PowerAirConditionCommand.class, this::onPowerAirConditionOff)
.onSignal(PostStop.class, signal -> onPostStop())
.build();
}
/*private Behavior<AirConditionCommand> onIgnoreMessage(ChangedTemperatureCommand r) {
//ignore messages when power is off. otherwise I get the info in the console that the message was unhanc
return this;
}*/
3 usages ClaraTschamon *
private AirCondition onPostStop() {
getContext().getLog().info("TemperatureSensor actor stopped");
return this;
}
}