Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Semesterproject

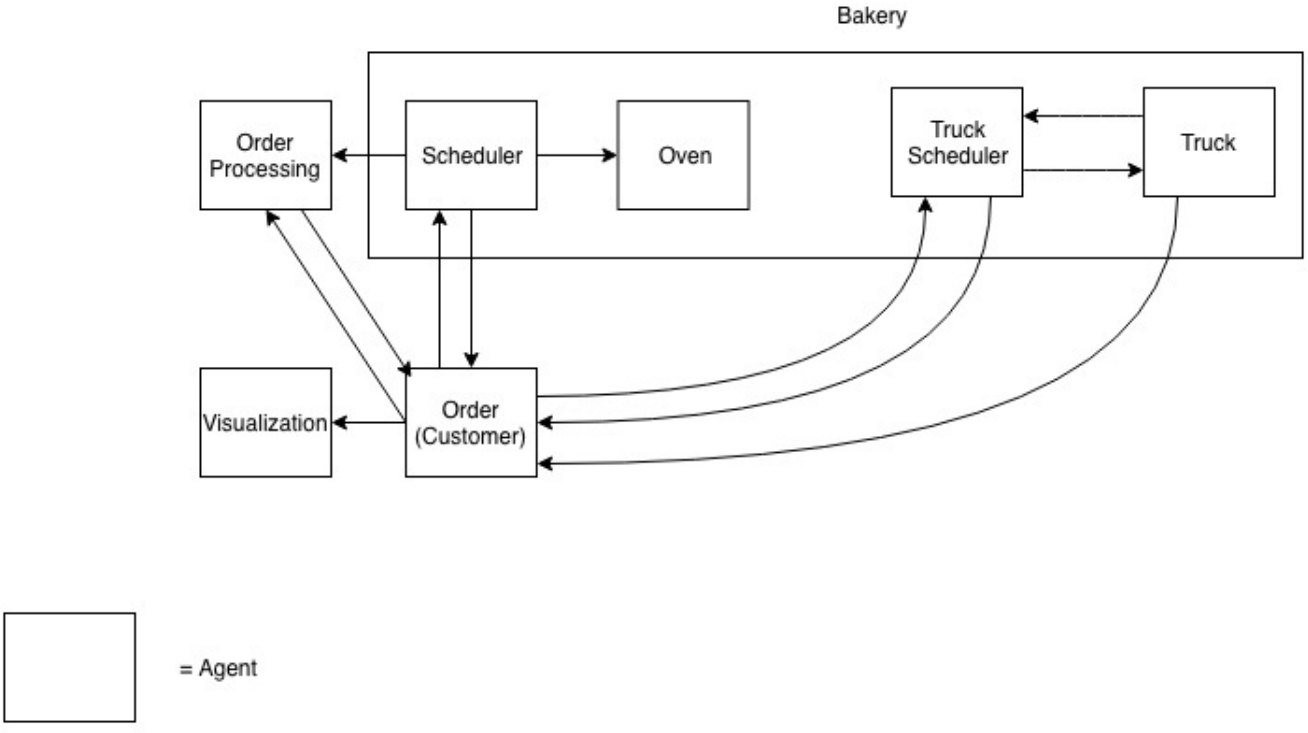
Multi Agent and Agent Systems

Department of Autonomous Systems
Referent: Prof. Dr. Gerhard Kraetzschmar

Submitted by: Team PJT

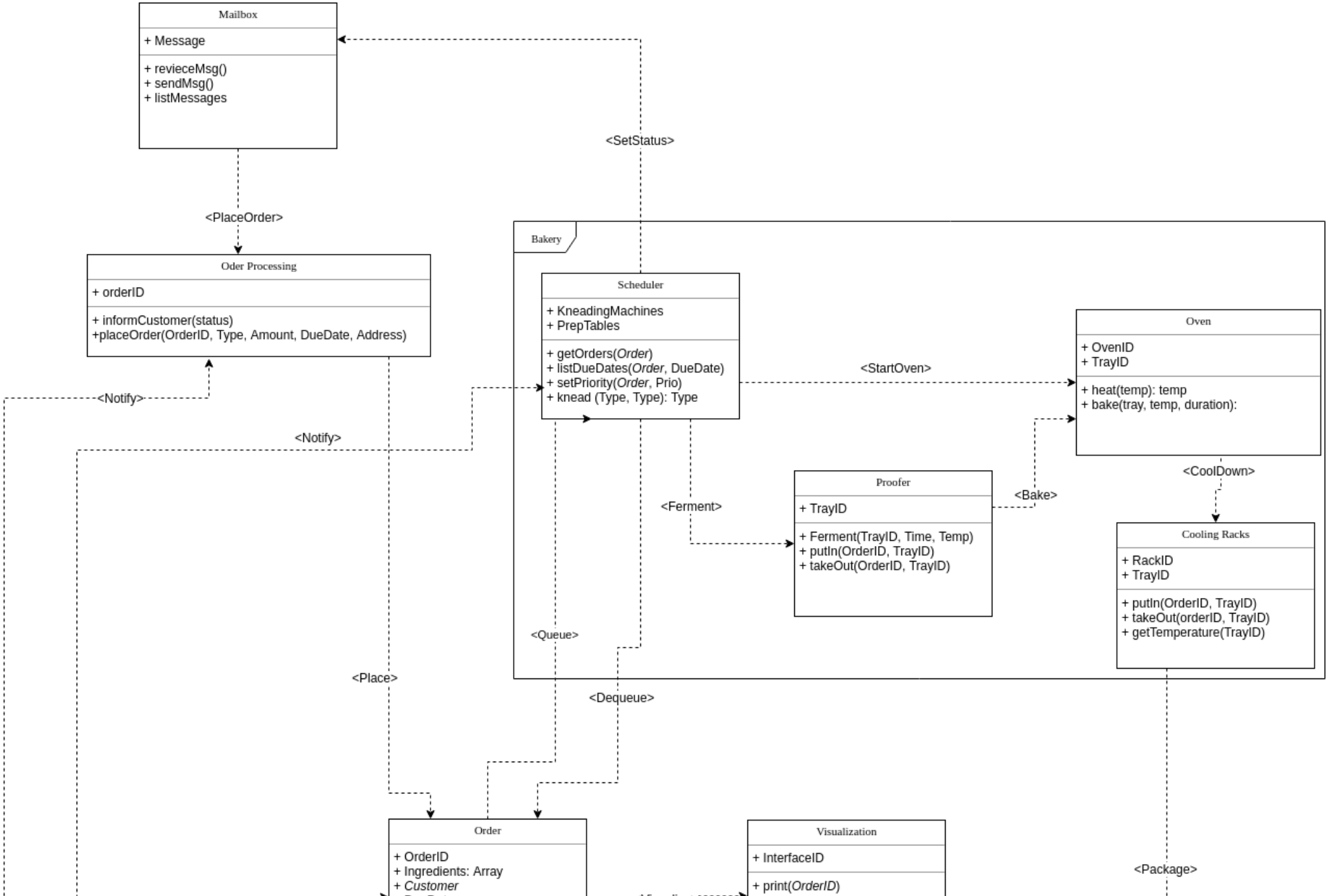
Sankt Augustin 12. November 2018

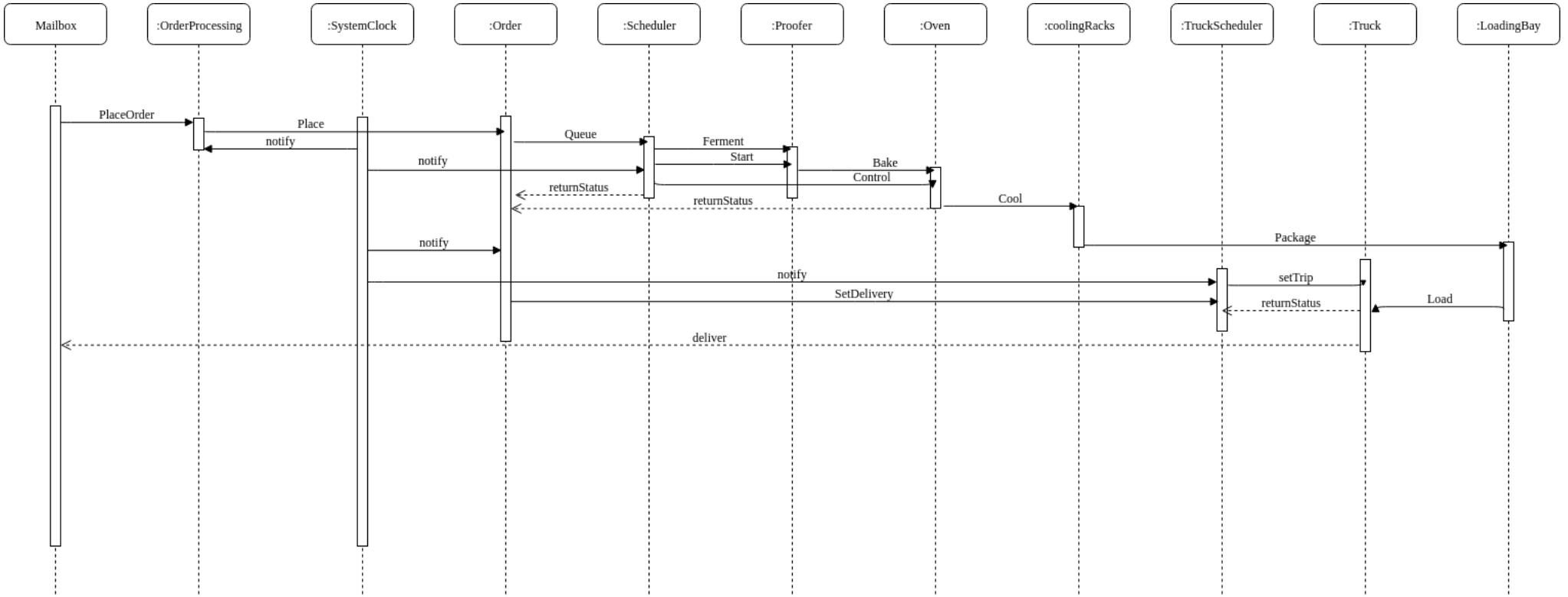
1 Architecture



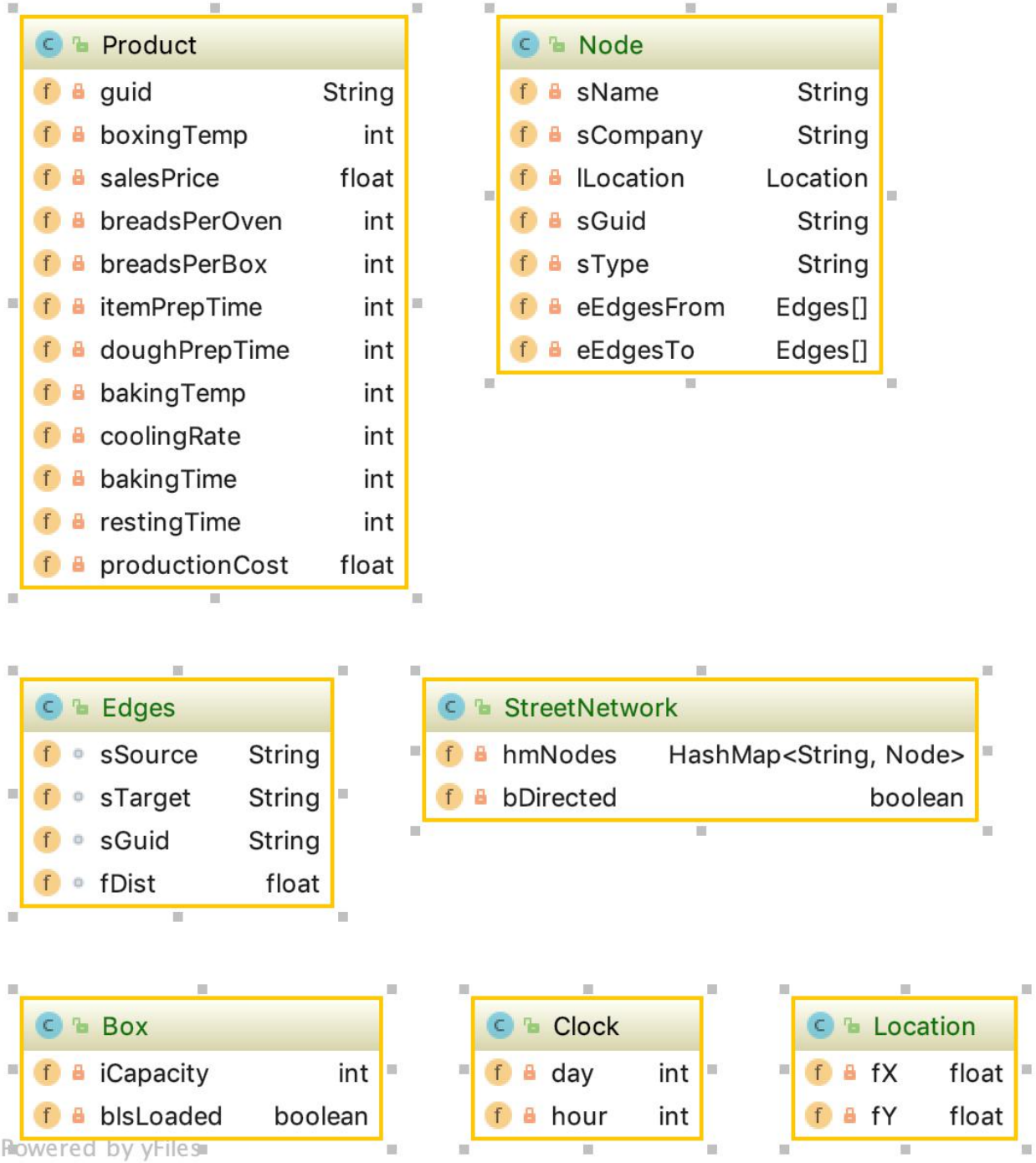
To which stage do these agents belong?

- Order processing
  - Customer
  - Order Processing
  - Scheduler
  - Order
- Dough preparation
  - Order
- Baking and Cooking
  - Scheduler
  - Oven
  - Order
- Packing and Loading
  - Truck Scheduler
  - Order
- Delivery
  - Truck Scheduler
  - Truck
  - Order


Mailbox
+ Message
+ revieceMsg()
+ sendMsg()
+ listMessages
<SetStatus>
<PlaceOrder>
Bakery
Oder Processing
+ orderID
+ informCustomer(status)
+placeOrder(OrderID, Type, Amount, DueDate, Address)
Scheduler
+ KneadingMachines
+ PrepTables
+ getOrders(Order)
+ listDueDates(Order, DueDate)
+ setPriority(Order, Prio)
+ knead (Type, Type): Type
<StartOven>
Oven
+ OvenID
+ TrayID
+ heat(temp): temp
+ bake(tray, temp, duration):
<Notify>
<Notify>
<Ferment>
Proofer
+ TrayID
+ Ferment(TrayID, Time, Temp)
+ putIn(OrderID, TrayID)
+ takeOut(OrderID, TrayID)
<Bake>
<CoolDown>
Cooling Racks
+ RackID
+ TrayID
+ putIn(OrderID, TrayID)
+ takeOut(orderID, TrayID)
+ getTemperature(TrayID)
<Queue>
<Place>
<Dequeue>
Order
+ OrderID
+ Ingredients: Array
+ Customer
Visualization
+ InterfaceID
+ print(OrderID)
<Package>


Mailbox
:OrderProcessing
:SystemClock
:Order
:Scheduler
:Proofer
:Oven
:coolingRacks
:TruckScheduler
:Truck
:LoadingBay
PlaceOrder
Place
notify
Queue
Ferment
Start
notify
Bake
Control
returnStatus
returnStatus
Cool
notify
Package
notify
setTrip
SetDelivery
returnStatus
Load
deliver

2 Objects



3 Aggregation of order data

Aggregation of order data can be done in the following manner:

- **An aggregation of a customer's orders for each day or each date <ddd.hh>**
  → It depends which data structure you could use

- If it is really important to you that you access date by given data format you could use a hashmap. Key is date value is order. Worst performance of searching a hashmap is $O(n) = log(n)$

```
Hashmap<Date, Order> hmMapDaily = new Hashmap<Date, Order>();
hmMapDaily.put(new Date(), new Order());
Order co = hmMapDaily.get(date);
```

- If it is not that important to use the given dateformat you could use an array. Index is day of a year. That means here worst performance of searching an array given that you know which day you want to search is $O(n) = 1$

- **An aggregation of all orders for a particular product for each day or each date** $\rightarrow$ Hashmap of Hashmaps. One entry within Hashmap represents one product. Key is product value is a hashmap. One Hashmap within Hashmap has as key a date, as value an array of orders.

```
Hashmap<ProductId, Hashmap<Date, Orders[]>> hMapProduct;
hMapProduct.put(new ProductId(), Hashmap<Date, Orders[]>);
Hashmap<Date, Orders[]> hmDate = hMapProduct.get(ProductId);
```

So hMapProduct would look the following way:

$$hMapProduct = \begin{pmatrix} \{ProductId, Hashmap < Date, Orders[] >\} \\ . \\ . \\ . \\ \{ProductId, Hashmap < Date, Orders[] >\} \end{pmatrix}$$