Hochschule Bonn-Rhein-Sieg

Multi-Agent and Agent Systems

Team : Right brothers

Flying Saucers Bakery

Authors:
Arun Rajendra Prabhu
Dharmin Bakaraniya
Md Zahiduzzaman

January 21, 2019

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Contents

Introduction:

The overall architecture diagram describing the deligation of tasks in the bakery is provided by the figure 1.

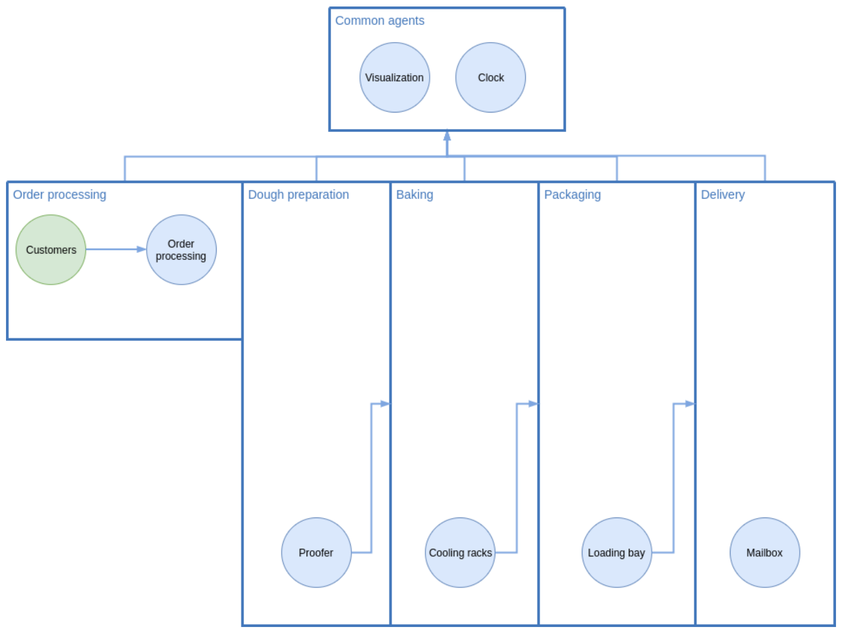


Figure 1: Architecture Diagram

The right brothers team was responsible to provide the **baking** and **packaging** stages. In addition to these stages, it was also responsible for providing the **base agent**, **time keeper** agent and **board visualization** feature. These agents fall in the common agents catagory and we will look at each one of them in detail in the coming sections.

With figure 1 as the starting point, we created a component diagram, figure 2 to describe the relationship between the agents in different stages. This diagram was created from the perspective of the stages which are under the right brother's responsibility, hence the only the interface agents of the other stages are represented in the diagram.

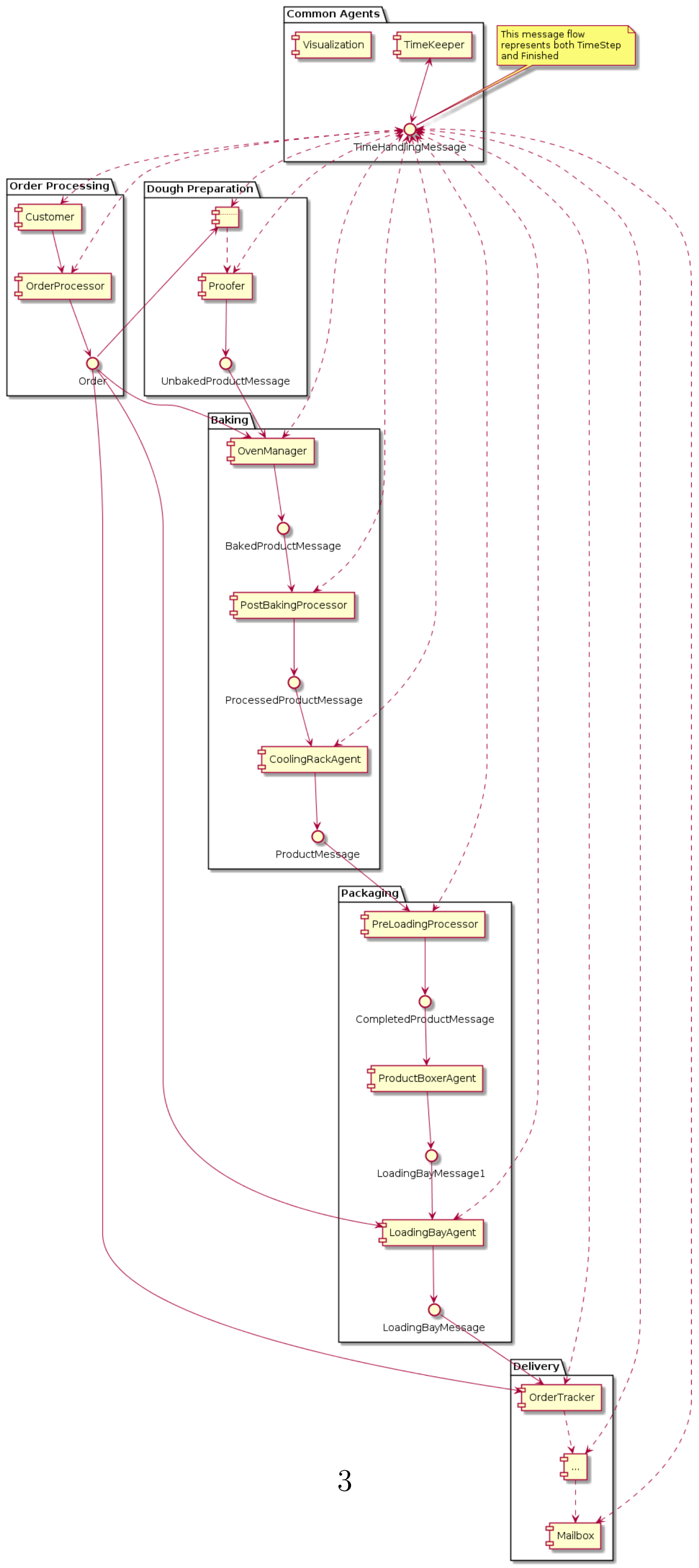


Figure 2: Component diagram representation

All the agents in our stages are created statically at startup. To illustrate this decision, consider the baking stage. Each bakery has a fixed number of ovens. In our design, we have a single agent called as OvenManager looking after all the ovens. Hence as per our design, it does not make sense to use dynamic agent creation for OvenManager. Also static agent creation, simplifies the overall interoperability to a cetain extent.

1 Baking Stage:

- **OvenManager**: We assign a single Agent to manage all the ovens in the bakery. It receives the `Order` from `OrderProcessor` agent. It receives `UnbakedProductMessage` from `Proofer`. It bakes the products it received from `Proofer` and sends `BakedProductMessage` to `PostBakingProcessor`. It aggregates products of same type if the product is not being baked.
- **PostBakingProcessor**: It receives `BakedProductMessage` from `OvenManager`. It processes all the steps in the recipe of that product which occur between `Baking` and `Cooling` steps. It sends the `ProcessedProductMessage` to `CoolingRackAgent`.
- **CoolingRackAgent**: It receives `ProcessedProductMessage` from `PostBakingProcessor`. It performs cooling step of the recipe corresponding to that product. It sends `ProductMessage` to `PreLoadingProcessor` in the packaging stage.

2 Packaging Stage:

- **PreLoadingProcessor**: It recieves `ProductMessage` from `CoolingRackAgent` from Baking stage. It performs all steps in recipe of that product which lie between `Cooling` and `Packaging`. It sends `CompletedProductMessage` to `ProductBoxerAgent`.
- **ProductBoxerAgent**: It receives `CompletedProductMessage` from the `PreLoadingProcessor`. It also receives the `Order` from `OrderPro-`

`cessor`. It then prioritizes the orders based on the delivery times. It supports two types of priorities,

  - Hard priority: Waits for the entire order of higher priority is completed before it moves on to process the next high priority order.
  - Soft priority: Checks if the products available in the inventory can be used to satisfy the pending orders according to their priority. If suppose there are 5 donuts in inventory and order1(high priority) needs 10 donuts and order2(lower priority) needs 5 donuts, then the 5 donuts are used to satisfy order2. However if there were 10 donuts in the inventory then the 10 donuts would have been used to satisfy order1.

  `LoadingBayMessage` is sent to the `LoadingBay` agent under two conditions,

  - If a box is filled.
  - If a box is not full, however, all the required number of items of a product type in an order have been received.

- **LoadingBayAgent**: It recieves `LoadingBayMessage` from `ProductBoxerAgent`. It then checks for order completion. If any orders are complete, it sends the message to the `OrderTracking/OrderAggregator` agent of the `Delivery Stage`.

3 Common Agents:

- **TimeKeeper**: It is responsible for the movement of time in the entire bakery eco-system. This is responsible for providing all the other agents with a common time reference so that everyone are in sync except the visualization agents. This also gets feedback from all the agents about the status of their tasks. If all the agents are done with whatever task they were supposed to finish in the time step, the TimeKeeper increments the time step. It is also responsible for shutting down the platform when the simulation time ends.
- **BaseAgent**: BaseAgent is a parent to all agents in bakery simulation.

It is mainly responsible for registering the agent to yellow pages and talking to TimeKeeper. It is also responsible for sending the message to visualization agents. Every agent that inherits BaseAgent has to call `finished` when their task is finished for that time step.

- **Visualization**: VisualizationAgent receives the output messages of the interface agents. It is a JADE agent extended from **BaseAgent**. It creates an instance of JavaFX application instance **Visualizer** during initialization and forwards all interface agent messages to the Visualizer instance. Please refer Figure 3 for a snapshot of the visualiser gui in action and refer Figure 4 for the class diagram of the visualiser agent.

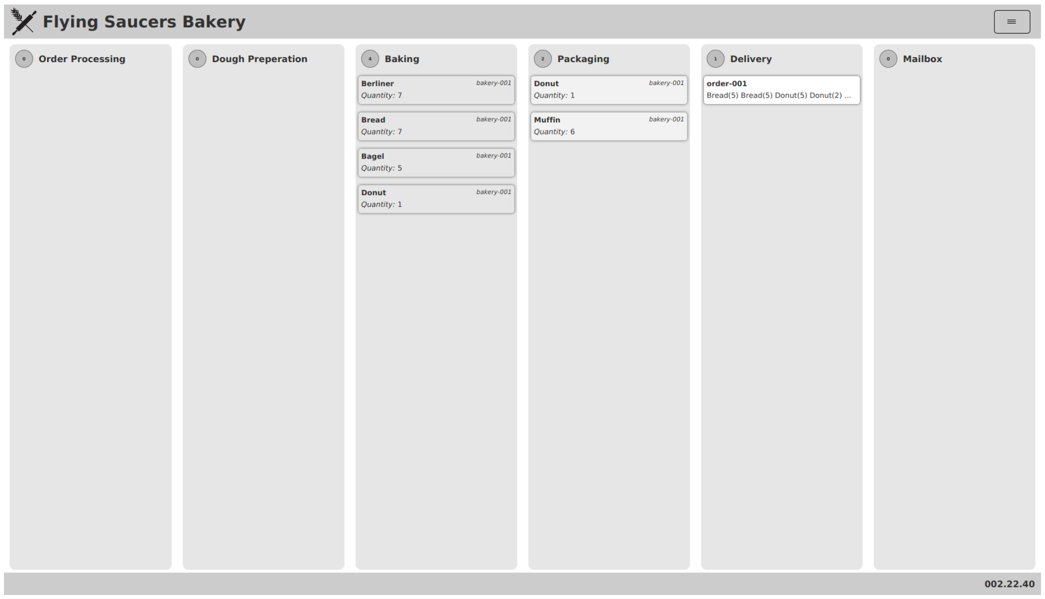
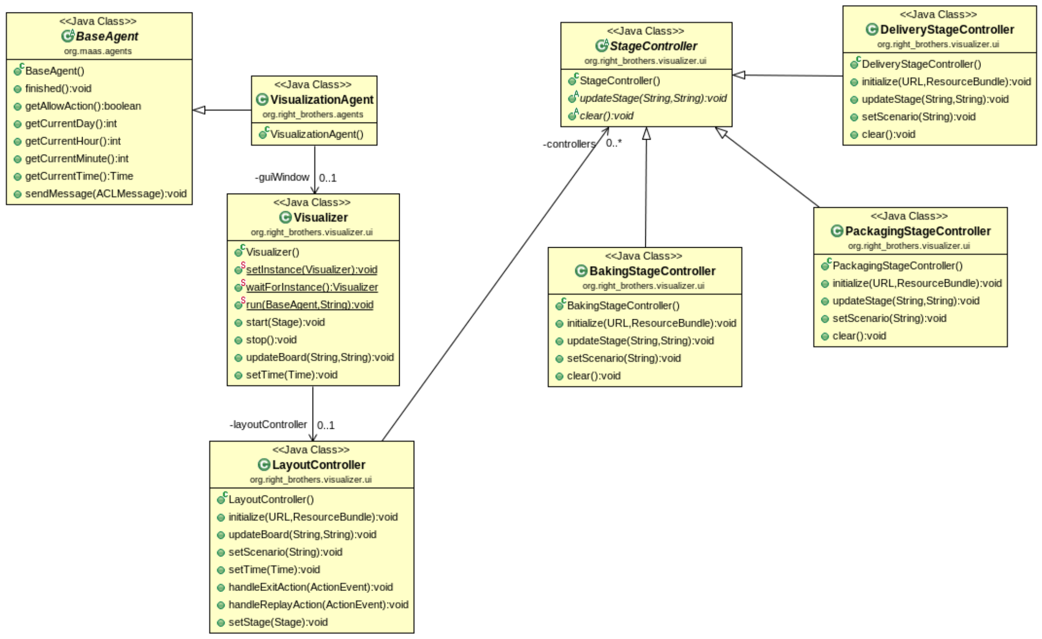

Flying Saucers Bakery
0 Order Processing
0 Dough Preperation
4 Baking
Berliner bakery-001 Quantity: 7
Bread bakery-001 Quantity: 7
Bagel bakery-001 Quantity: 5
Donut bakery-001 Quantity: 1
2 Packaging
Donut bakery-001 Quantity: 1
Muffin bakery-001 Quantity: 6
1 Delivery
order-001 Bread(5) Bread(5) Donut(5) Donut(2) ...
0 Mailbox
002.22.40

Figure 3: Board visualization in progress



<<Java Class>>
BaseAgent
org.maas.agents
BaseAgent()
finished():void
getAllowAction():boolean
getCurrentDay():int
getCurrentHour():int
getCurrentMinute():int
getCurrentTime():Time
sendMessage(ACLMessage):void
<<Java Class>>
VisualizationAgent
org.right_brothers.agents
VisualizationAgent()
-guiWindow
0..1
<<Java Class>>
Visualizer
org.right_brothers.visualizer.ui
Visualizer()
setInstance(Visualizer):void
waitForInstance():Visualizer
run(BaseAgent,String):void
start(Stage):void
stop():void
updateBoard(String,String):void
setTime(Time):void
-layoutController
0..1
<<Java Class>>
LayoutController
org.right_brothers.visualizer.ui
LayoutController()
initialize(URL,ResourceBundle):void
updateBoard(String,String):void
setScenario(String):void
setTime(Time):void
handleExitAction(ActionEvent):void
handleReplayAction(ActionEvent):void
setStage(Stage):void
-controllers
0..*
<<Java Class>>
StageController
org.right_brothers.visualizer.ui
StageController()
updateStage(String,String):void
clear():void
<<Java Class>>
DeliveryStageController
org.right_brothers.visualizer.ui
DeliveryStageController()
initialize(URL,ResourceBundle):void
updateStage(String,String):void
setScenario(String):void
clear():void
<<Java Class>>
BakingStageController
org.right_brothers.visualizer.ui
BakingStageController()
initialize(URL,ResourceBundle):void
updateStage(String,String):void
setScenario(String):void
clear():void
<<Java Class>>
PackagingStageController
org.right_brothers.visualizer.ui
PackagingStageController()
initialize(URL,ResourceBundle):void
updateStage(String,String):void
setScenario(String):void
clear():void

Figure 4: Class diagram of the major classes of the visualizer

4 Mockup Agents:

For testing the individual stages for proper functioning, we create a number of agents basically as mockup agents to provided the necessary input messages to the interface messages of our stages. These are listed below along with their functionality.

- **DummyAgent**: The purpose of this agent was to test the TimeKeeper agent. It prints a message every time step till the platform shuts down.
- **DummyCoolingRackAgent**: This dummy agent get the order messages from order processing agent and creates ProductMessage objects. This product message is converted into a json string message and sent to PreLoadingBayAgent. This dummy is used in the test for packaging stage.
- **DummyOrderProcessor**: This dummy agent reads a client.json file from a scenario directory and create order messages from that infor-

mation. This order message is then sent to all the agents of its bakery. The dummy order processor takes all the order for its bakery so there cannot be multiple order processing agent reading the same file.

- **DummyProofer**: Similar to DummyCoolingRackAgent, this agent receives order messages from order processing agent and converts the order to UnbakedProductMessage and sends it to the OvenManager of its bakery.
- **DummyReceiverAgent**: This agent is made for the sole purpose of receiving INFORM messages from a particular agent. It takes 3 arguments, namely, guid of the bakery it is part of, the name of the agent it will receive messages from and the conversation id of those messages. This agent is also used for testing different stages. It is the last agent in the chain and once it receives a message, we assert that the stage is working correctly.

Following section will provided class descriptions for the different agents discussed in this section.

5 Class descriptions

5.1 TimeKeeper

- **Stage**: Common Agent
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: SendTimeStep (OneShot), TimeStepConfirmationBehaviour (Cyclic)
- **Messages in**:
  - finished (Sender: all agents)
- **Messages out**:
  - TimeStep (Receiver: all agents)

5.2 BaseAgent

- **Stage**: NA
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: PermitAction (Cyclic)
- **Messages in**:
  - TimeStep (Sender: TimeKeeper)
- **Messages out**:
  - finished (Receiver: TimeKeeper)
  - all messages(Receiver: Visualization Agents)

5.3 Visualization agent

- **Stage**: Common Agents
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: MessageServer (Cyclic)
- **Messages in**:
  - all messages (Sender: all agents)

5.4 OvenManager

- **Stage**: Baking
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: Bake (Cyclic), OrderServer (Cyclic), UnbakedProductsServer (Cyclic)
- **Messages in**:
  - Order (Sender: OrderProcessor)
  - UnbakedProductMessage (Sender: Proofer)
- **Messages out**:
  - BakedProductMessage (Receiver: PostBakingProcessor)

5.5 PostBakingProcessor

- **Stage**: Baking
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: BakedProductsServer (Cyclic), Process (Cyclic)
- **Messages in**:
  - BakedProductMessage (Sender: OvenManager)
- **Messages out**:
  - ProcessedProductMessage (Receiver: CoolingRackAgent)

5.6 CoolingRackAgent

- **Stage**: Baking
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: ProcessedProductServer (Cyclic), CoolProducts (Cyclic)
- **Messages in**:
  - ProcessedProductMessage (Sender: PostBakingProcessor)
- **Messages out**:
  - ProductMessage (Receiver: PreLoadingProcessor)

5.7 PreLoadingProcessor

- **Stage**: Packaging
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: Process (Cyclic), CooledProductServer (Cyclic), OrderServer (Cyclic)
- **Messages in**:
  - ProductMessage (Sender: CoolingRackAgent)
  - Order (Sender: OrderProcessor)
- **Messages out**:
  - CompletedProductMessage (Receiver: LoadingBayAgent)

5.8 ProductBoxerAgent

- **Stage**: Packaging
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: OrderReceiver (Cyclic), CompletedProductReceiver (Cyclic)
- **Messages in**:
  - CompletedProductMessage (Sender: PreLoadingProcessor)
- **Messages out**:
  - LoadingBayMessage (Receiver: LoadingBayAgent)

5.9 LoadingBayAgent

- **Stage**: Packaging
- **Agent/Object**: Agent
- **Static/Dynamic**: Static
- **Behaviour**: OrderDetailsReceiver (Cyclic), ProductDetailsReceiver (Cyclic)
- **Messages in**:
  - LoadingBayAgent (Sender: ProductBoxerAgent)
- **Messages out**:
  - LoadingBayMessage (Receiver: OrderTracker/OrderAggregator)

6 Message Object Description

6.1 UnbakedProductMessage

- Recipient: OvenManager(Baking Stage)
- Sender: Proofer(Dough Preparation Stage)
- Class description:

```java
public class UnbakedProductMessage implements java.io.Serializable {
    private String productType;
    private Vector<String> guids;
    private Vector<Integer> productQuantities;

    public UnbakedProductMessage() {
        this.guids = new Vector<String> ();
        this.productQuantities = new Vector<Integer> ();
    }

    public void setProductType(String productType) {
        this.productType = productType;
    }

    public String getProductType() {
        return productType;
    }

    public void setGuids(Vector<String> guids) {
        this.guids = guids;
    }

    public Vector<String> getGuids() {
        return guids;
    }

    public void setProductQuantities(Vector<Integer> productQuantities) {
        this.productQuantities = productQuantities;
    }

    public Vector<Integer> getProductQuantities() {
        return productQuantities;
    }
}
```

6.2 BakedProductMessage

- Recipient: PostBakingProcessor(Baking Stage)
- Sender: OvenManager(Baking Stage)
- Class description:

```
public class BakedProductMessage implements java.io.Serializable {
    private String guid;
    private int coolingDuration;
    private Vector<Step> intermediateSteps;
    private int quantity;

    public BakedProductMessage() {
        this.intermediateSteps = new Vector<Step> ();
    }
    public void setGuid(String id) {
        this.guid = id;
    }

    public String getGuid() {
        return this.guid;
    }

    public void setCoolingDuration(int coolingDuration) {
        this.coolingDuration = coolingDuration;
    }

    public int getCoolingDuration() {
        return coolingDuration;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setIntermediateSteps(Vector<Step> intermediateSteps) {
        this.intermediateSteps = intermediateSteps;
    }

    public Vector<Step> getIntermediateSteps() {
        return intermediateSteps;
    }
}
```

6.3 ProcessedProductMessage

- Recipient: CoolingRackAgent(Baking Stage)
- Sender: PostBakingProcessor(Baking Stage)

- Class description:

```
public class ProcessedProductMessage implements java.io.Serializable {
    private String guid;
    private int coolingDuration;
    private int quantity;

    public void setGuid(String id) {
        this.guid = id;
    }

    public String getGuid() {
        return this.guid;
    }

    public void setCoolingDuration(int coolingDuration) {
        this.coolingDuration = coolingDuration;
    }

    public int getCoolingDuration() {
        return coolingDuration;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public int getQuantity() {
        return quantity;
    }

}
```

6.4 ProductMessage

- Recipient: PreLoadingProcessor(Packaging Stage)
- Sender: CoolingRackAgent(Baking Stage)

- Class description:

```
public class ProductMessage implements java.io.Serializable {
    private Hashtable<String,Integer> products;

    public ProductMessage() {
        this.setProducts(new Hashtable<>());
    }

    public void setProducts(Hashtable<String,Integer> products) {
        this.products = products;
    }

    public Hashtable<String,Integer> getProducts() {
        return products;
    }
}
```

6.5 CompletedProductMessage

- Recipient: ProductBoxerAgent(Packaging Stage)
- Sender: PreLoadingProcessor(Packaging Stage)
- Class description:

```
public class CompletedProductMessage implements java.io.Serializable {
    private String guid;
    private int quantity;

    public void setGuid(String id) {
        this.guid = id;
    }

    public String getGuid() {
        return this.guid;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public int getQuantity() {
        return quantity;
    }

}
```

6.6 LoadingBayMessage

- Recipient: LoadingBayAgent(Packaging Stage)
- Sender: ProductBoxerAgent(Packaging Stage)
- Class description:

```
public class LoadingBayMessage {
    private String orderId;
    private List<LoadingBayBox> boxes;

    public LoadingBayMessage() {
        boxes = new ArrayList<LoadingBayBox>();
    }

    public LoadingBayMessage(String orderId, List<LoadingBayBox> boxes) {
        this();

        this.orderId = orderId;
        this.boxes = boxes;
    }
    @JsonProperty("OrderID")
    public String getOrderId() {
        return orderId;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }
    @JsonProperty("Boxes")
    public List<LoadingBayBox> getBoxes() {
        return boxes;
    }
    public void setBoxes(List<LoadingBayBox> boxes) {
        this.boxes = boxes;
    }
}
```

7 Testing

- For testing TimeKeeper
  `gradle run`
- For testing baking stage
  `gradle run --args="-baking"`
- For testing packaging stage
  `gradle run --args="-packaging"`
- For testing baking and packaging stage
  `gradle run --args="-baking -packaging"`
- For testing baking and packaging and visualisation stage
  `gradle run --args="-baking -packaging -visualization"`