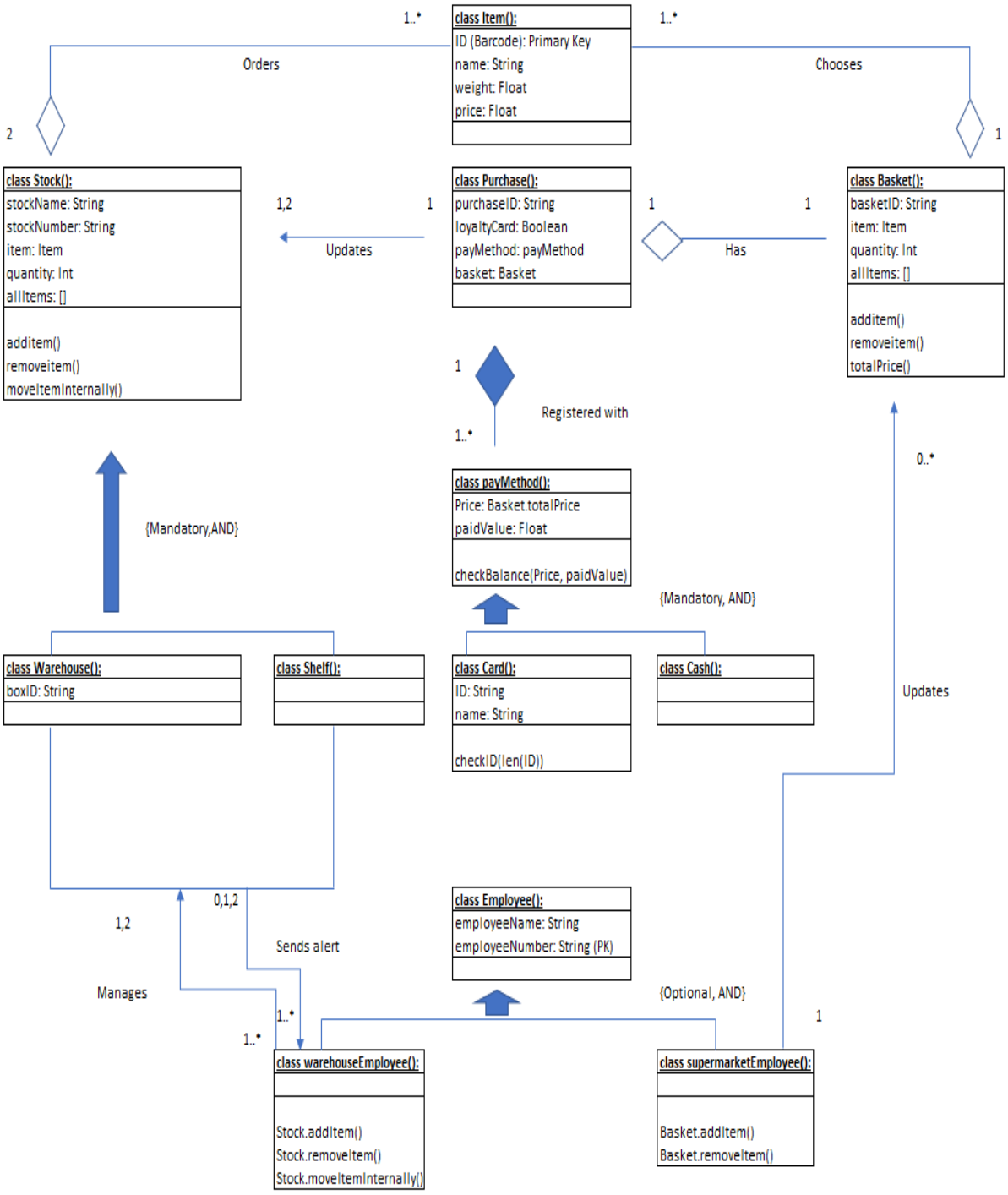
Self-Service Checkout in a Supermarket: System Design

The system design depicted in this report represents a self-service checkout supermarket system. To be clear, the system doesn't cover the idea of an online-shop and therefore entirely depends on the idea that goods/items are directly accessible to customers.

The most important object of a supermarket are goods (Figure 1 represents the entire structure). The representing class of "Item" is the class that includes all the metadata of an item such as barcode ID, name, weight, price and is a static class (= no methods). Only in having items, a supermarket could make profit. Beside items, the next important object is the physical supermarket which is in this system design represented by the "Stock" class which includes shelves and the warehouse (for storing goods) as subclasses. Looking inside the stock class, one can see that different stocks have different stock numbers and that different methods allow to add/remove items partially or fully to/from the stock (Methods "addItem", "removeItem" take an item and its desired quantity and add/remove it to the list of "allItems" by "stockNumber"). Furthermore, it is possible to use methods which internally transfer items from shelf to warehouse or vice versa ("moveItemInternally"). Aside from all the attributes and methods of the stock class, the warehouse class also includes as variable an ID for every box in the warehouse so that employees can better navigate in this place.

Figure 1: System Design of a Self-Service Checkout in a Supermarket



The third step consists of including customers into this system (without using specifically a customer class). The “Purchase” class represents this idea and it includes as variables an ID, a “loyaltyCard” (Boolean), a “payMethod” and a “basket”. The basket class should represent the process before the payment when customers are in the scanning phase where (quantities of) items can still be added and removed

to the list of “allItems”. Note, the purchase class has a class into it, “payMethod”. The payMethod class has as own attributes the “Price” of a basket and the “paidValue” by the customer. The class then allows for checking whether the payment was greater than, equal to or smaller than the price (“checkBalance”). The subclass “Card” then also adds a string variable (name) which differentiates between different kind of cards (Bank Card, Apple Pay, Loyalty Points, etc.) and it allows to check whether the length of variable “ID” is reasonable. Going back to the basket class, supermarket employees have the right to manually change a basket when for example a barcode hasn’t been correctly scanned. This idea is captured by the inclusion of a “supermarketEmployee” class (as a subclass of “Employee”) and the methods are “addItem” and “removeItem”.

When a financial transaction has been finalized, the stock system simultaneously gets the information that items left the supermarket and the system sends alerts to employees of the warehouse system if needed (sometimes a shelf stays half-filled). One can see that only a “warehouseEmployee” has the ability to add and remove items from the stock or to make an internal transfer (warehouse -> shelf or vice versa).

For further explanation, Table 1 describes every class more in detail whereas Table 2 reflects all the relationships used in Figure 1 and the reason of their inclusion (see below).

Table 1:Self-Service Checkout Supermarket: Classes and Description

Class	Description
Item	Contains all the Metdadata of every Item.
Basket	Represents the pre process of a purchase. Here, like a real basket, customers can add/remove items and quantities of items.
Purchase	It stands for the moment when customers scanned the basket of items and they are now in the paying process.
payMethod	This class takes the total price of a basket and the paid value as attributes and its method allows to check whether the received value is greater, smaller or equal to the price.
Cash/Card	The decision for their inclusion is based on required additional information for cards (ID number) and a method that checks for reasonable ID's. (Class "Cash" has no own attributes)
Stock	This class consists of key variables and methods to create and maintain a stock (similar to "Basket").
Shelf/Warehouse	The explicit use of these two classes is explained by the reason that a warehouse often consists of a box system that helps employees in their navigation. On the other side, a shelf is more flexible and changes more often the structure (no systematic structure).
Employee	This class with its two attributes (Name, Number) contains all the Metadata of all employees.
supermarketEmployee/ warehouseEmployee	The creation of these two classes is argued by different methods that are applicable by different staff members. Supermarket staff can add and remove items from a basket list(when inaccurately used by clients) whereas warehouse employees can use all the methods of the stock class (add, remove and remove internally).

Table 2:Self-Service Checkout Supermarket: Relationships of Classes and Description

Relationship	Detail	Description
Inheritance	payMethod <---------- Cash (Superclass) {Mandatory, AND} Card (Subclass)	The only payment method accessible to clients are the represented subclasses (mandatory relationship). Nevertheless, it should be allowed to combine two methods for one purchase, loyalty points with cash for example (=AND).
	Stock <---------- Shelf {Mandatory, AND} Warehouse	In this system, the supermarket has no third stock (a deposit for example), the relationship therefore is mandatory. However, an item can be present in both stocks at the same time (=AND).
	Employee <---------- supermarketEmployee {Optional, AND} warehouseEmployee	It is likely that the two represented subclasses are not the only ones (optionally, a manager class could be added but is not required). Nonetheless, it could be assumed that some employees are working between supermarket and warehouse and are a part of both subclasses (=AND).
Association	Stock <---------- Purchase 1,2 (Updates) 1	One purchase updates at least one stock, mostly both of them (when there is a internal transfer).
	Basket <---------- supermarketEmployee 0..* (Updates) 1	One supermarket employee (not warehouse staff) has the ability to add and remove items if needed (restricted age goods or wrong scan).
Recursive Association	Stock <---------- warehouseEmployee 1,2 (Manages) 1..*	One or more warehouse employees manage one or two stocks. Though, it is possible that some employees are more focused on the shelves whereas others are entirely responsible for the warehouse stock.
	Stock ----------> warehouseEmployee 0,1,2 (Sends alert) 1..*	Both stocks (shelf and warehouse) have the ability to send alerts and one or more employees can handle such a request.
Composition	Purchase ◆---- payMethod 1 (Registered with) 1..*	Every purchase is registered with one or more payment methods and the payMethod class only exists within the Purchase class. If there is no purchase, there is no reason to believe that the PayMethod class will ever be used.
Aggregation	Purchase ◇---- Basket 1 (Has) 1	Every purchase has exactly one basket, but the basket is only one part of a complete purchase. In other words, a basket instance can be created before the existence of a purchase instance and is therefore more independently compared to payMethod.
	Basket ◇---- Item 1 (Chooses) 1..*	One basket can choose 1 or more items but the Item class does not entirely depend on the existence of the basket class. Imagine that the supermarket would decide to completely close business and would stop the purchase of items in one location at a specific time. Then, there will probably be a stock of rest items that have to be transferred as a final step (when the basket class already is out of use) to another supermarket.
	Stock ◇---- Item 2 (Orders) 1..*	One or many items will be ordered by the supermarket and will be placed in both, warehouse and shelves. If there would be no stock system (neither shelf nor warehouse), then we can imagine that the supermarket with its items can still exist. Imagine a supermarket which doesn't work with a stock system but maintains an item list for other use (to create a virtual a basket for example).

Bibliography:

Ambler, S. (2003) *Elements of UML Style.* Cambridge: Cambridge University Press

Bruegge, B. (2014) *Object-oriented software engineering : using UML, patterns, and Java.* Harlow: Pearson

Connolly, T. & Beg, C. (2015) *Database Systems: A Practical Approach to Design, Implementation, and Management.* Global Edition. Edinburgh: Pearso

Linnaeus University (2016) Lecture 5-Modeling with UML. Available from: https://www.youtube.com/watch?v=ZJICJIwwS4w [Accessed 24 March 2022]