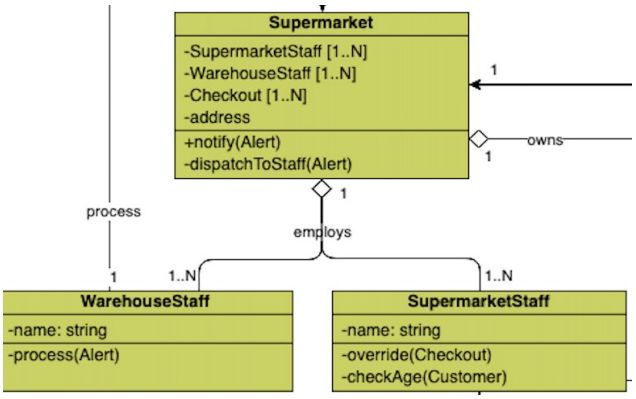
Self- service checkouts

System design

Supermarket Domain

Supermarket models the physical stores, and **WarehouseStaff** and **SupermarketStaff** the two types of employees. The staff is part of the supermarket but can exist independently therefore aggregation is the chosen relation.



Supermarket delegates WarehouseStaff to process Alerts for low level of stocks.

The staff could share a common base class, but it was omitted because no function requires it.

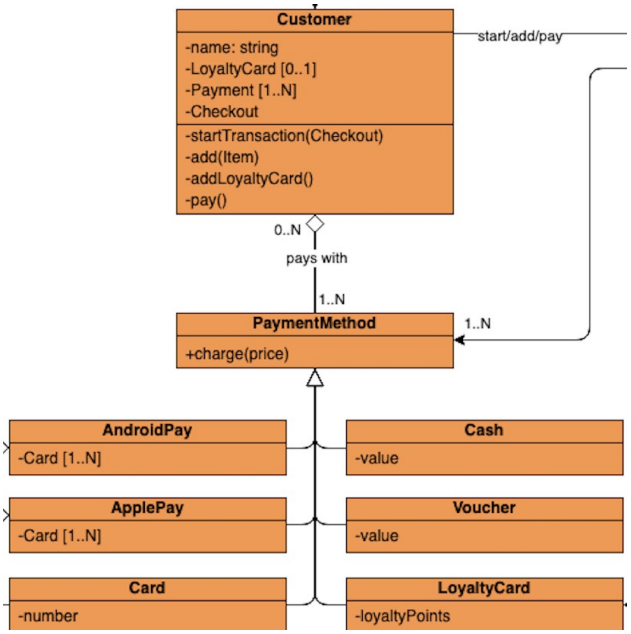
Similarly, multiple Supermarkets could be aggregated to model the SupermarketChain, but no function requires it.

Customer Domain

Customer is the main class driving the process, therefore all its methods are private.

Customer can have one transaction at a time. The method “startTransaction()” assigns one Checkout to be used in all operations until the end of the transaction. Customer is only associated with Checkout given the temporary nature of their relation.

Customer has **PaymentMethods** to be used via the method “pay()”. Given the stable many-to-many relation between Customers and PaymentMethods, aggregation is the appropriate relation.



PaymentMethods, as base-class of all payment methods, is useful to encapsulate the complexity of each single method behind the generic method “charge()”. Payment methods include the subclasses **Cash**, **Voucher**, **LoyaltyCard**, **Card**, **ApplePay**, and **AndroidPay**. ApplePay and AndroidPay aggregate Card acting as a wrapper.

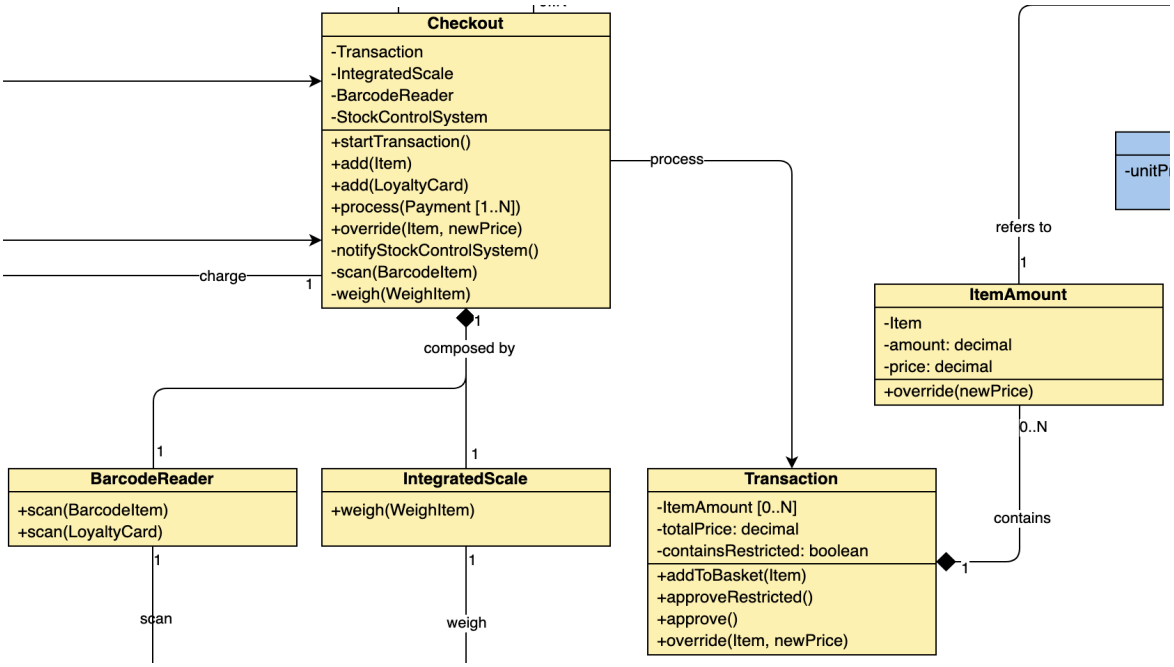
Checkout Domain

Checkout represents the self service machine that a customers can use.

Only one customer a time can use one Checkout. The state of Checkout is represented by **Transaction** that is created by the method “startTransaction()” and destroyed at the end of the payment processing. Given the transitory relation, the two classes are only associated. A Transaction collects **ItemAmounts** representing the amount of Items being purchased. Their relation is composition because the destruction of the Transaction destroys ItemAmounts as well. ItemAmount has a price that by default is equivalent to the referred Item’s price, but that can be overridden by SupermarketStaff.

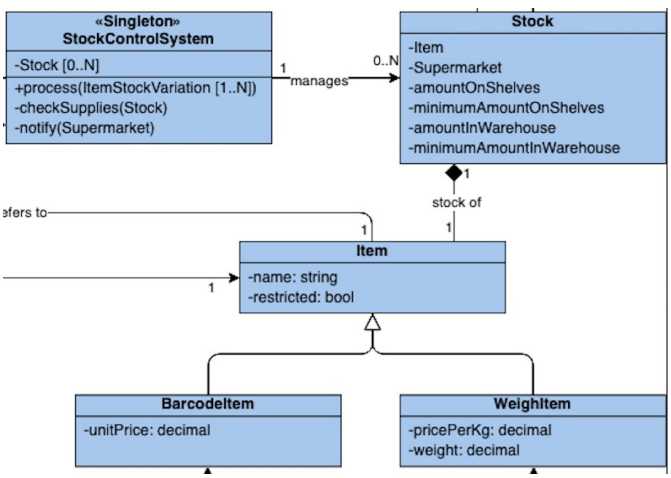
The “add()” method in Checkout is used for both Items and LoyaltyCard and delegates to the private methods “scan()” and “weigh()” that in turn delegate to **IntegratedScale** and **BarcodeReader** that are components of Checkout.

Part of the payment processing includes a call to the private method “notify()” responsible for notifications to the StockControlSystem.



All interactions to Transation, IntegratedScale, and BarcodeReader are mediated by Checkout.

StockControlSystem Domain



Item is the base class of the items for sale. The classes **WeighItems** and **BarcodeItems** are required to specialize the different processes that add them to the Transaction.

Stock models the relation between Item and Supermarket holding the current and desired minimum level of stocks. Its relation with Item is composition because erasing Item implies erasing its stocks. Conversely, its relation with Supermarket is a simple association because Stocks can be moved between stores.

StockControlSystem is the singleton responsible for keeping track of the **Stocks**. The relation is association because the system can exist initially with no Stocks.

Notifications

Notifications are classes modeling a message. **ItemStockVariation** models the message sent by Checkout to notify StockControlSystem about the purchase of one Item. **Alert** models the message that StockControlSystem may send to a Supermarket signaling that stock levels are low. Their existence depends on Stock and Item respectively, therefore their relation is one-to-one composition.