Mid-Module Assignments

Object-oriented Information Systems November 2021

# I. List of Classes

## Loyalty

Since every customer will enrol in a loyalty program, where each customer will have one loyalty card. And considering that the loyalty can be accessed without the presence of the customer. A class is created that will represent the Loyalty object.

## Customer

As the main Actor in the scenario, the Customer class will represent the buyer’s actor. For every Customer will have one Loyalty object, linked by its loyalty\_id attribute. This object will be able to scan\_items(), pay\_order(), enter\_loyalty\_card(), and see their loyalty information through the get\_loyalty\_number().

## Product

Product class represents the product the store is selling. But to make it easy, no matter how many items there are in the inventory, the system will be having only one object for all of them. For example, there are 10 books and 5 bottles in the inventory. The system will have only 2 objects of the Product class that represents the books and bottles. Instead, the information regarding the number of stocks available will be stored as the class attribute.

Several attributes are also being set so that the Product class will be able to do important operations such as create\_stock\_alert(), which will alert the respective employee, stored inside the product\_manager attribute, that the stock is low and needs to be replenished.

## Order

Although an Order is not a living thing, an order can have its behaviours. Thus, constructing a class designated for orders will be helpful to make it centralize all the data and operations about it. Having its class also makes it easy for the overall system during a transaction. It will help the Customer object do a payment, it will update Product object attributes after a transaction is done, etc.

## Employee

The employee is another living actor in the scenario. Its job is to order products from suppliers, override transactions manually in the case of scanning error, or check the Customer’s age when a restricted item is in the bucket. Thus, creating an Employee class will help in placing, or encapsulating the information and operations correctly.

## Payment

The Payment class will be created when a Customer object is about to pay. The reason why it is having its class is that there will be multiple payment methods available for the Customer object, and they are all having different attributes and behaviours on each method. Having a parent class for each method will help in simplifying the construction of the class.

## Cash, Coupon, Card

Represents the payment method available. Inherited from the Payment class.

## Debit, Credit

Represents the sub-type of Card objects. Since Credit card and Debit card ways of processing payments are different – e.g. a signature won’t make a debit payment valid, each of them are having its class.

# II. Relationships

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Type** | **Class** | **Relationship Notes** |
| **Loyalty** | *Association* | Customer | A customer class will have a Loyalty object with a 1-to-1 relationship. A Customer won't be able to create many Loyalty objects. |
| **Customer** | *Association* | Order | A Customer can create an order by scan\_items, thus the relationship is Association as Order will closely related with Customer objects. Customers can create many Order objects, thus this association is in a 1-to-many relationship. |
| *Association* | Product | Since the Customer will be able to create an Order object by scanning the product, an association with the Product should exist. This relationship will help the Customer on the scan\_items() operation. Because the product\_id will be exposed to the Customer. |
| **Product** | *Aggregation* | Order | Since the order is made up of multiple Product objects, the relationship between the Product and Order class is an Aggregation relationship. |
| **Order** | *Association* | Payment | An order will be related to payment activity. But not all Order objects will create a Payment object. This relationship means that not all information of Payment will be stored in an Order. Through payment\_id, it can tell which Order object is related to which Payment object. |
| **Employee** | Association | Product | Product class will alert Employee object if the number of stock available is low, as well as the ability for Employee to order the product – with reorder\_stock\_from\_supplier() operation inside Employee object and create\_stock\_alert() operation inside Product object. |
| Association | Order | Employee object will be able to do customer's age checking as well as manually override an order. through check\_customer\_age() and override\_order() operation.  Thus, with this rationale, the Employee class will have an Association relationship with the Order class.  This also means that Employee doesn't need to have direct interaction with the Customer object. Everything about a transaction is "centralized" in the Order object. |
| **Payment** | *Generalization* | Cash | An inheritance from the Payment class that represents the sub-type of a Payment object. |
| *Generalization* | Coupon | An inheritance from the Payment class that represents the sub-type of a Payment object. |
| *Generalization* | Cards | An inheritance from the Payment class that represents the sub-type of a Payment object. |
| **Card** | *Generalization* | Debit | Represents the subtype of the Card class. |
| *Generalization* | Credit | Represents the subtype of the Card class. |