Mid Module Assignment: System Design

<u>Class Diagram</u>

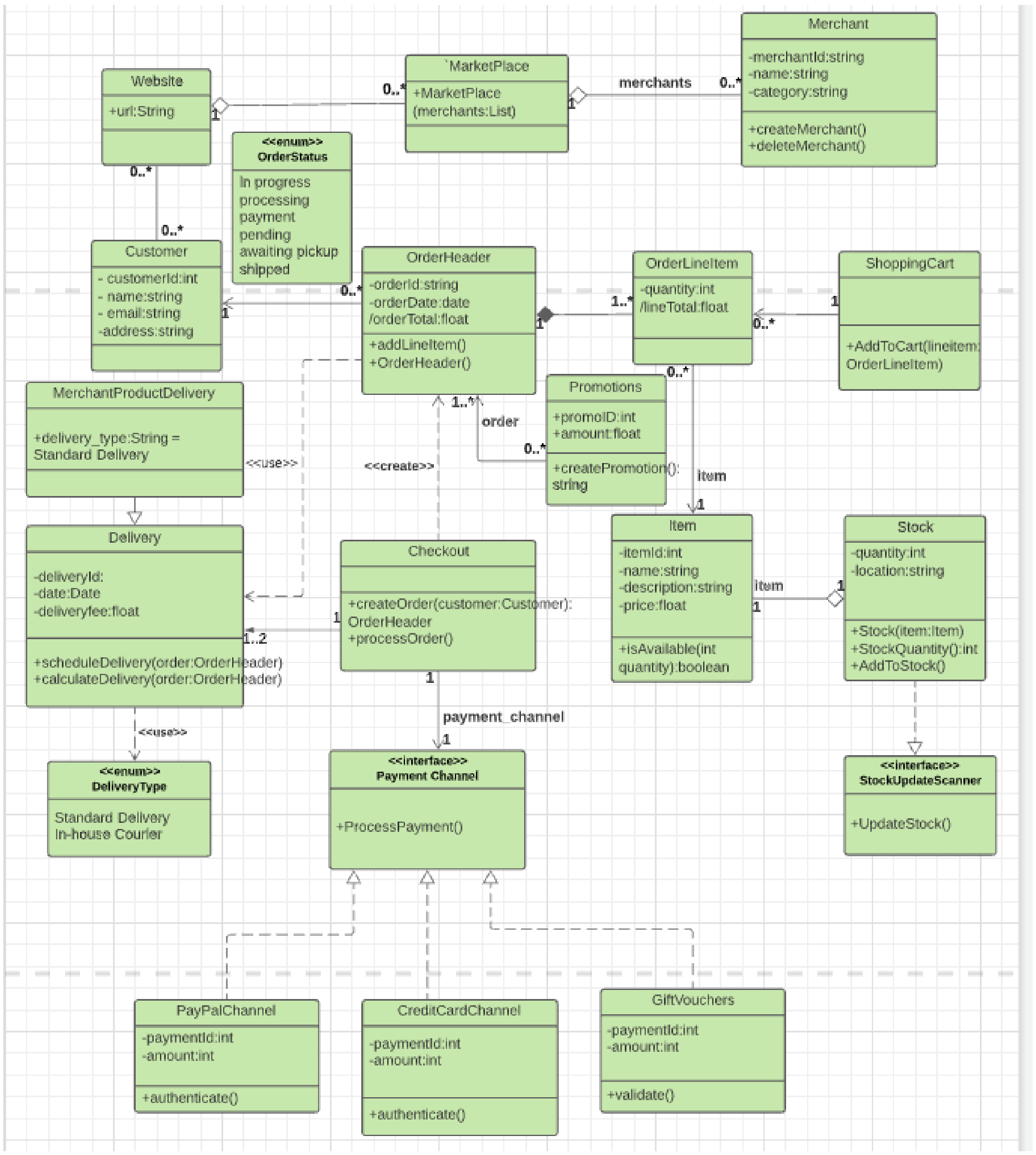

Fig 1: Class Diagram of an Online Store

Class relationships

Firstly, in the class diagram above, Order has been split into two parts; the **OrderHeader** class which shows a reference to the customer who placed the order, the order id, order date and other details and the **OrderLineItem** class which refers to each item on the order and the quantity.

The relationship between **OrderHeader** and **OrderLineItem** has been illustrated as composition as logically, an order cannot exist without buying at least an item. The **OrderLineItem** class has a variable item of class **Item** and the communication is unidirectional.

Customer and **OrderHeader** classes also have a unidirectional association relationship however, an order cannot exist without a customer so the **OrderHeader** class has been implemented with a constructor which accepts an object of type **Customer**. A composition relationship is not ideal in this case as a customer does not compose of order in the real sense.

The **OrderHeader** class also creates order line items, since those line items can then explicitly belong only to the **OrderHeader** instance that created them.

There is a two-way association relationship between the **Stock** and **Item** class. The Item class has an +isAvailable () method that checks if an item is in stock and reflects it on the website while the **Stock** class has a constructor that accepts an Item object enforcing the constraint that a stock object cannot be created without a reference item. If an aggregation relationship is used here, the system would be implemented as a Stock object aggregating several item objects, that is a stock object holding a list of items. This means each item of same product must have a unique identifier and could make the system a bit complicated.

The **Stock** class realizes an interface **<<StockUpdateScanner>>** to update stock of items.

The requirements states that there are three payment options therefore the **PaymentChannel** class has been modelled as an interface realized by all three different channel classes actually responsible for the method +processPayment()

The Checkout class is responsible for creating and validating an order and as such has association relationships with the **OrderHeader** and **PaymentChannel** classes. It is also responsible for specifying delivery details hence the association relationship with **Delivery** class. The **Delivery** class depends on the enum **DeliveryType** to calculate the delivery fee and has a **MerchantDelivery** child class that handles delivery for products bought from the merchants on marketplace.

The **OrderHeader** class also depends on the **Delivery** class as a change in the delivery class can affect order details.

Attributes

All attributes have been defined as private although the /lineTotal and /orderTotal are shown as derived attributes in the **OrderLineItem** and **OrderHeader** classes respectively.

In the **MerchantProductDelivery** class, the attribute delivery_type has a constant value of “standard delivery” as the system requirement states that merchant products can only be delivered using standard delivery option.

Enums

There are two enums depicted on the class diagram representing **DeliveryType** and **OrderStatus** with specific values as stated in the systems requirement.

Activity Diagram

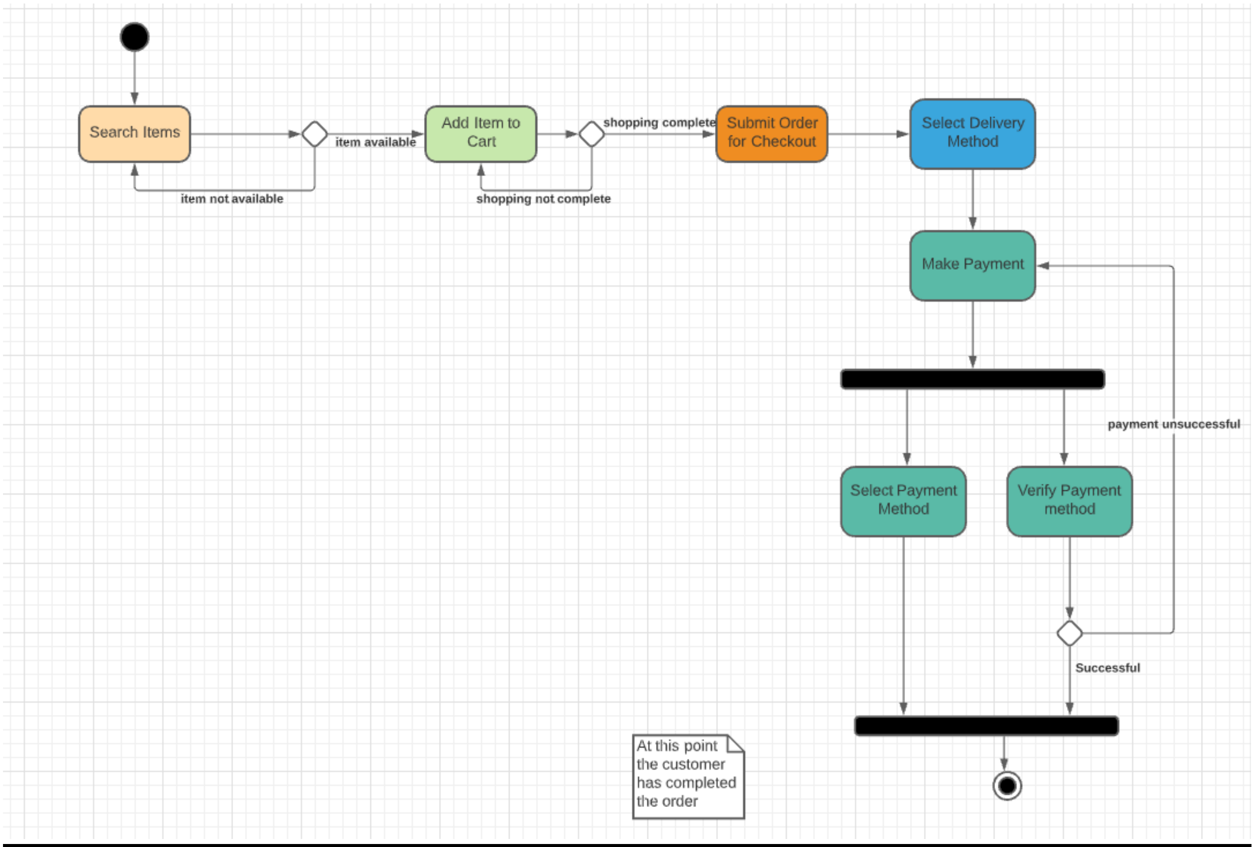

Fig 2: Activity Diagram

State Diagram

Order States
In progress
processing
Payment pending
Awaiting payment
Shipped

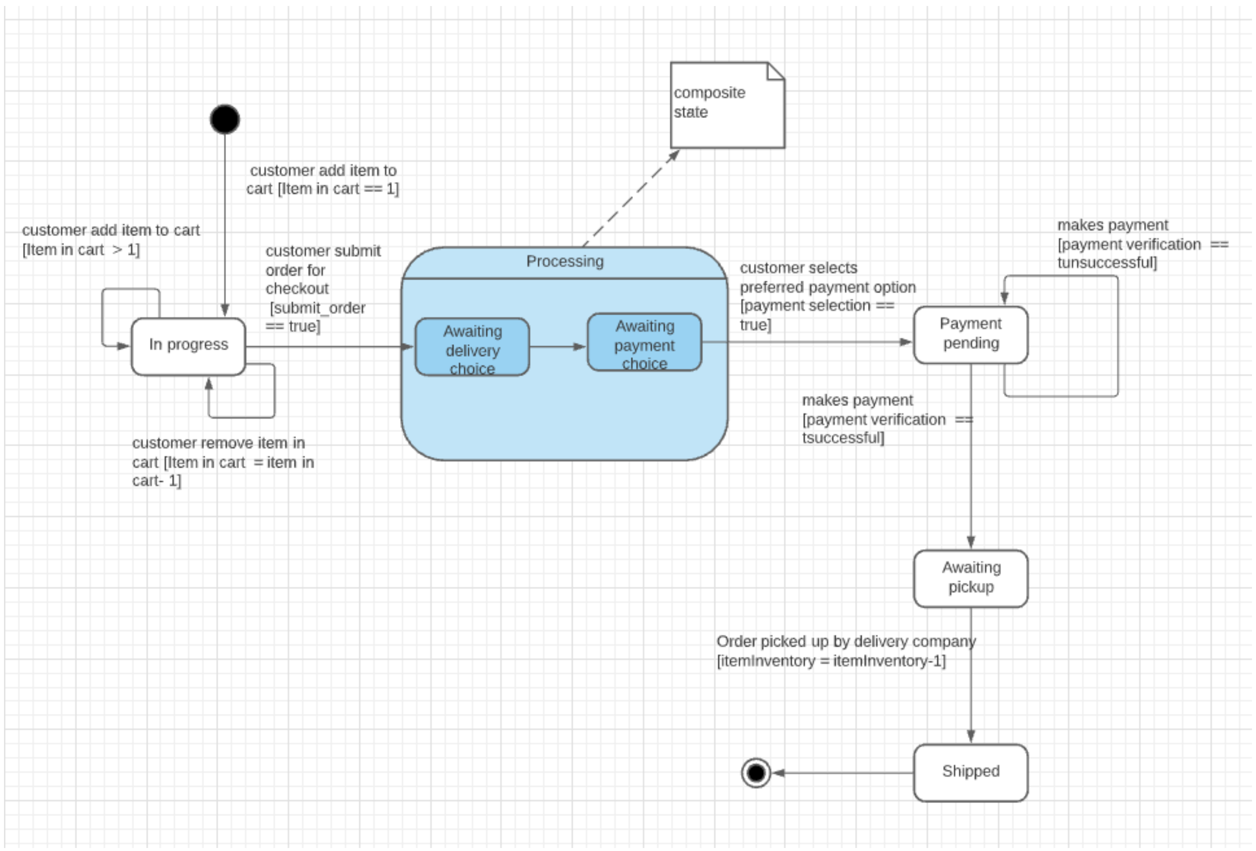
composite state
customer add item to cart [Item in cart == 1]
customer add item to cart [Item in cart > 1]
customer submit order for checkout [submit_order == true]
Processing
Awaiting delivery choice
Awaiting payment choice
customer selects preferred payment option [payment selection == true]
makes payment [payment verification == tunsuccessful]
In progress
Payment pending
makes payment [payment verification == tsuccessful]
customer remove item in cart [Item in cart = item in cart- 1]
Awaiting pickup
Order picked up by delivery company [itemInventory = itemInventory-1]
Shipped

Fig 3: State transition Diagram

References

Lucidchart (2021) UML Class Diagram tutorial. Available from: https://www.lucidchart.com/pages/uml-class-diagram [Accessed 18 June 2021].

Nel R. (2020) UML Class Diagram for programmers. Available from: https://www.udemy.com/course/uml-class-diagrams-for-programmers/ [Accessed 16 June 2021].

VisualParadigm (2021) State Machine Diagram tutorial Available from: https://online.visual-paradigm.com/diagrams/tutorials/state-machine-diagram-tutorial/ [Accessed 19 June 2021].