**Brandon Hobbs**

**CS-255**

**May 23, 2022**

**Module 4: OOD Discussion**

Hamp Crafts’ online system was decomposed into an object model that shows 7 classes: *Shopping Cart*, *Customer*, *Order*, *User*, *Administrator*, *Shipping Info*, and *Order Details*. Within the UML class diagram there are a few important observations to be made:

* *User* is a superclass with derived classes *Customer* and *Administrator*
* *Customer* is associated through composition with the *Shopping Cart* and *Order* classes; that is, without the *Customer* object *Order* and *Shopping Cart* cannot stand alone
* The *Order* class is associated through composition with the *Shipping Info* and *Order Details* classes

The *User* class being a superclass with derived classes *Customer* and *Administrator* suggests that there are to be specializations, or roles, within the application. That is, since not all users are intended to be an administrator, the *Administrator* class is created as a subclass to the *User* superclass with additional attributes and methods.

Moreover, the multiplicity/cardinality of the various classes is also denoted in the UML. For example, The *Customer* object, with a multiplicity of one and only one, is shown to be associated through composition with the of the *Shopping Cart* object, with a multiplicity of zero to many. This is correct, as a customer may “own” zero, one, or multiple shopping carts but a *Shopping Cart* object belonging to multiple customers would be very confusing.

The UML shows that *Shopping Cart* is associated with the *Customer* by the solid black diamond, a special aggregation type (composition) that means the *Shopping Cart* objects cannot exist without the *Customer*. This makes sense as if orphaned *Shopping Cart* Objects were allowed, as if it were simple aggregation, then there would be no context of who that *Shopping Cart* belonged to. Another way to think about this is to say if a user was deleted from the system should their shopping carts remain – more than likely not. The same thoughts may be directed towards *Order Details* and *Shipping Info* being associated with *Order* – without the order there is no reason to keep the shipping and order details.

The UML also shows the private attributes and public methods available within the classes. For example, the *Shipping Info* object has a public method, updateShippingInfo, that would allow the program to, presumably, update the shipping information of an associated *Order*. *Shopping Cart* also has a similar method called addCartItem so a user can add more items to that cart. Overall, the UML suggests that encapsulation is being used as the attributes’ states are being protected from manipulation except by very specific, public methods. Furthermore, abstraction is suggested with functions such as calcPrice in the *Order Details* class – there are no mentions of ***HOW*** the price is calculated as the user should not be concerned with that detail, just the resultant operation.

There is at least one missing class that might be imagined based upon Hamp Crafts’ stated wishes. For example, “the owners want to be sure that any payments will be transferred to Hamp Crafts’ business account” but there are no objects present to represent that business account nor are there any methods available in other classes to deposit payments.

This model focuses on the objects themselves whereas the previous, Module 3, model focuses on behaviors. Each has their specialty and communicates information differently. For example, the process model from Module 3 shows that the carrier exhibits the behavior of sending information to the shipper. This object model does not even have the concepts of Shipper and Carrier as they are outside the system. However, there could be, in theory, a *Shipper* object that contains pertinent details about that shipper, e.g., phone number, average lead time, and this might have been shown in the process model as a data store.

What is nice about the object model is that a reader can see the information (attributes) being captured within the system pertaining to a particular entity. That is, one can see in the UML that the date an item is added to the shopping cart and the quantity of that item are being tracked. This level of understanding could facilitate the creation of business Key Performance Indicators (KPIs) or even dashboards that drive real-time decision making, i.e., a user with an item in his cart for 5 days without purchasing might be converted to a buyer if offered a small discount on that item.