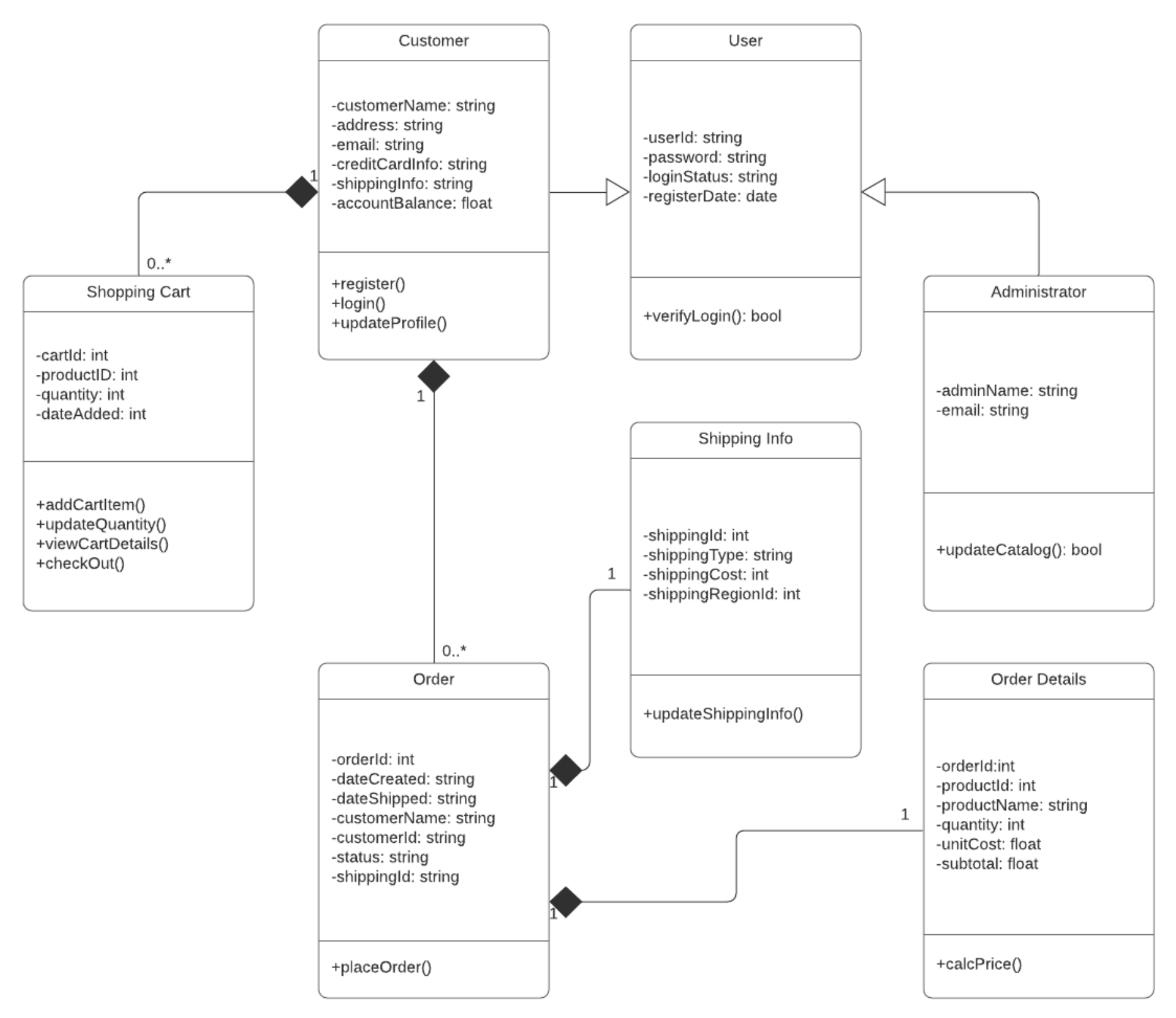
Daniel Loranger

CS-255, Week 4

Object Model Diagram



*Hamp Crafts would like customers to be able to create an account with their shipping, billing, and contact information. For customer orders, Hamp Crafts would like to accept credit and debit cards for transactions. Hamp Crafts plans on using an established credit card vendor service (e.g., Square, Shopify) to receive customer payments. Once a transaction is complete, the customer should receive a notification based on the information in their personal profile regarding order status and confirmation. On the administrative side of the online storefront, Hamp Crafts should receive an alert of the transaction. Customers should be able to check the status of their order any time online from their personal account profile under order history. The business owners also need an administrative back end for customer support and updates to customer information and the website.*

Analysis:

There are 7 classes portrayed in the UML Diagram provided, *Order*, *Order Details*, *Shopping Cart*, *Customer*, *User*, *Administrator*, and *Shipping Info*.

As portrayed, these classes are interdependent linked classes with some classes using polymorphism to add expanded functionality. An example is *User* is a base class where *Customer* and *Administrator* are *Users* that have expanded capabilities while retaining the base *User* functionality.

Also portrayed is the cardinality and composition (black diamonds) of each object which defines how certain objects should exist. For example, if an *Order* is deleted, then *Shipping Info* should also be deleted (composition link) else the *Shipping Info* would be orphaned information. *Order* is respectively dependent on a *Customer* existence; However, this dependency could be argued as a requirement for creation only, but should be maintained at *Customer* deletion as deleting *Order* information could lead to accounting issues. Cardinality of 1 order to 1 customer makes sense, and many orders to 1 *Customer* also makes sense, however, multiple *Shipping Info* to 1 *Order* could make sense for a single *Order* if the business intends to support back-orders while shipping in-stock products.

There are some behaviors that appear to missing such as upgrading/degrading a *User* to/from an Administrator, having an *Administrator* login routine (*Customer* class creates this, not the parent *User* class), having an *Administrator* be capable of being a *Customer* as well (a highly likely scenario). A class that might be considered missing is the *Payment* handler, no classes appear to be responsible for handling the payment system. Similarly, is an *Event Notifier*, that would email *Order* updates to the *Customer* and notify the processing center of a new event/order. This model needs further refinement with such considerations given additional review from an experienced webstore developer that will see missing details.

Using the process model method previously analyzed helps to show HOW the information will move thru the new system which can help to identify missing processes, where information will be needed shared as both sources and sinks, and can identify situations where special events need to happen that would not otherwise be captured. The Object model however is essential to capture WHAT the data is, where it will be handled, as well as will be implemented, public vs private, string vs int vs struct, etc. The Object model also portrays cardinality and composition where these details would not be visible on a process diagram.