The different classes of “users” represented by this object model are Shopping cart, Customer, User, Shipping Info, Administrator, Order Details, and Order.

The functions of the online store front are:

|  |  |  |  |
| --- | --- | --- | --- |
| addCartItem() | updateQuantity() | viewCartDetails() | checkout() |
| register() | login() | updateProfile() | placeOrder() |
| verifyLogin(): bool | updateShippingInfo() | updateCatalog(): bool | calcPrice() |

The functions in the object model are identified using the “+” identifier and are public, there are several private functions (-) such as UserId and password. The different user classes are Customer and Administrator. Admin can update the catalog, while customers can create a shopping cart and create orders from the catalog. Both rely on an email and password to access. The classes use their respective variables and functions to allow specific actions to be taken that are based on their respective class. For instance, the administrator class does not need to have a shopping cart because their class isn’t a customer, however they still need access to the catalog to update the catalog from the backend. The customer class can add products from the catalog to their cart but cannot access and update the catalog itself. Aside from connecting to a third party for secure payment, this layout covers the needs of Hemp Crafts. The black diamonds in the diagram represent a composition type of aggregation from another class. For example, the Customer class is a composition aggregation of Shopping Cart and Order; if the customer is deleted, so are the shopping cart and order associated with the customer account. The implication of the composition is that the attaching classes are dependents to the composition class and not predecessors. The black diamond is appropriate in this situation because an order and shopping cart cannot be generated without an associated customer account to assign the two classes too.   
 After reviewing the Hamp Crafts process model and the object model, for usability I find the object model for more useful as it is more detailed on what functions and classes need to be generated as well as how the classes need to communicate with one another. The process model is great in terms of an initial flow map, however once the work needs to be started an object model