Matthew Bourque
05/17/2016
COSC 716

In this project, I implemented two additional design patterns: The *Builder* pattern in order to build 'Daily' and 'Weekly' specials based on various aspects of the inventory, as well as the *Observer* pattern to send a warning to the system when inventory items are low in quantity (and to subsequently 'order' more inventory). Further details on these implementations are below and are also shown in the class diagram at the end of this document.

Builder pattern for specials:

The purpose of using this design pattern is to allow the restaurant system to construct specials based on the various attributes of the inventory items that make up the restaurant's menu. I specifically implemented a 'daily' special, which is comprised of inventory items that have the most available quantity (as to promote the sale of items in abundance), as well as a 'weekly' special, which is comprised of inventory items that have the earliest expiration date (as to promote the sale of items that are going to expire soon).

Several classes are constructed to support this implementation:

- ***MenuItem***, which contains a name (e.g. "Roasted Chicken Dinner"), price, and three *InventoryItems* (see below) which comprise the entree (a main dish, a vegetable, and a side).
- ***Menu***, which contains a list of all of the available *MenuItems*.
- ***InventoryItem***, which contains a name (e.g. "Roasted Chicken"), price, quantity,expiration date, and type (i.e. "main dish", "vegetable", and "side").
- ***Inventory***, which contains a list of all available *InventoryItems*
- ***Special***, which contains the type (i.e "daily" or "weekly"), name (e.g. "Roasted Chicken with Salad and White Rice"), price, and the three *InventoryItems* that comprise the special (i.e. a main dish, a vegetable, and a side).
- ***Specials***, which contains a list of the daily and weekly specials.

With these supporting classes, the Builder design pattern is implemented, which makes use of the following classes:

- A ***SpecialBuilder*** class, which is an interface containing methods to build the specific components of a *Special* (i.e. buildName(), buildPrice(), buildMain(), buildVegetable(), and buildSide()).
- Concrete builder classes ***DailySpecialBuilder*** and ***WeeklySpecialBuilder***, which implement the *SpecialBuilder* interface. The *DailySpecialBuilder* implements the methods to select the appropriate *InventoryItem* based on quantity, while the

WeeklySpecialBuilder implements the methods to select the appropriate *InventoryItem* based on expiration date.

- A ***Director*** class, which takes in a *SpecialBuilder* in it's constructor, and executes the building of the *Special* via a build() method.

Using this design pattern is beneficial, as it allows future maintainers of the system to easily modify the specials as well as remove or add specials. To add more types of specials (for example, a 'monthly' special, or a 'holiday' special), all that would be needed is to add another *SpecialBuilder* object (which would contain the logic to build a *Special* object based on inventory, time of year, etc), as well as the necessary logic in the *CMDDisplaySpecial Command* object to allow the *Director* to build the special. To change an existing special, the only object that needs to be modified are the concrete *SpecialBuilder* objects.

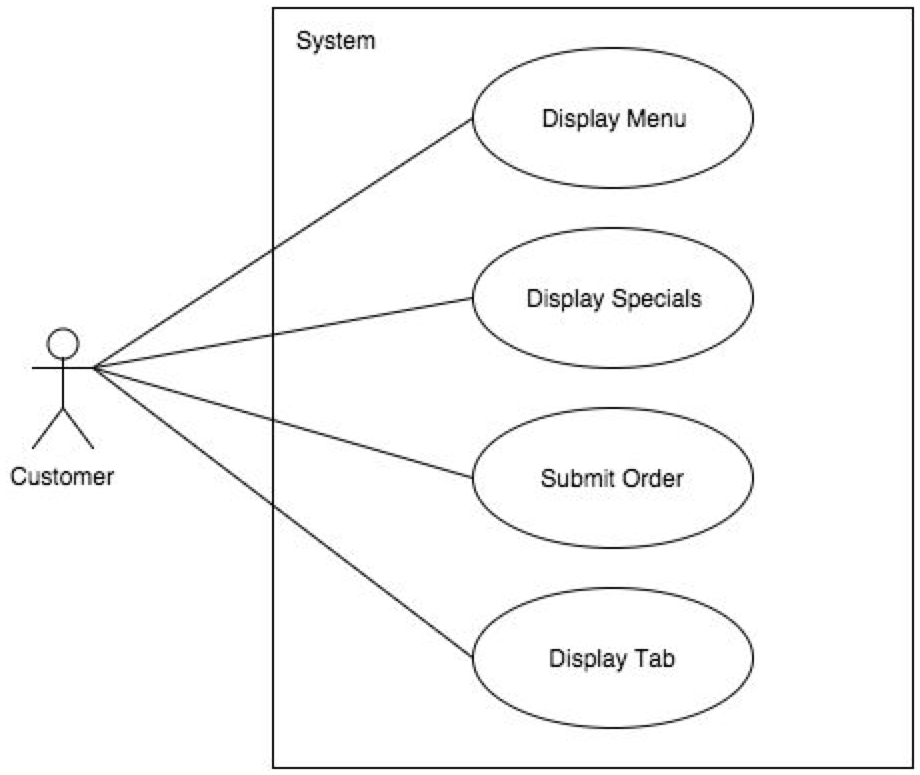
Observer pattern for low inventory:

The motivation behind using this design pattern is to allow the restaurant system to respond to changes in inventory. The specific implementation that I use is to make the *Inventory* class an *Observable* type (i.e. it implements the *Observable* interface, which has methods of register(), unregister(), and warn()), as well as make use of an *Observer* interface (which has a method of update()). The concrete observer that I specifically implement is a *LowInventoryObserver*.

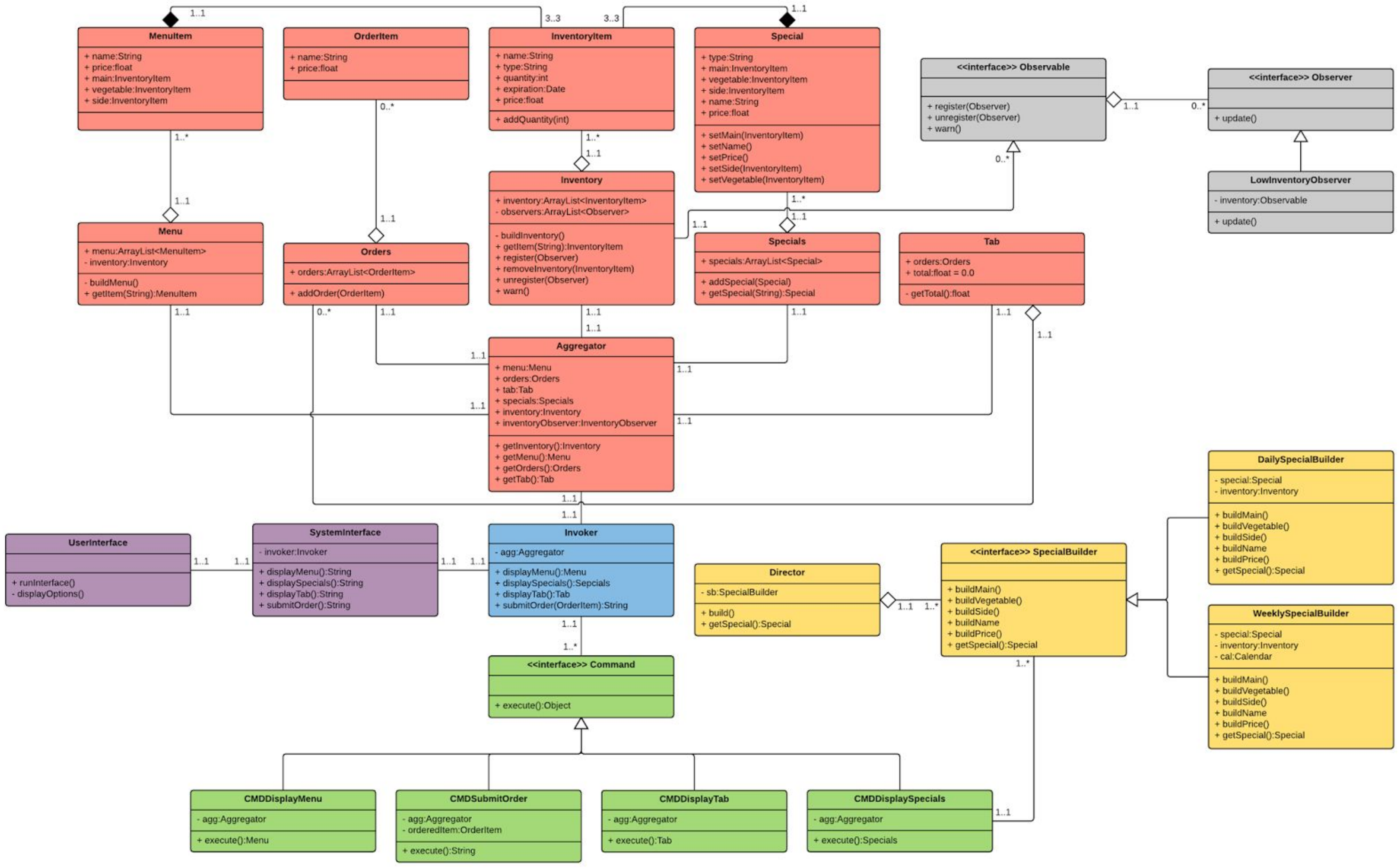
When an order is placed in the restaurant system, the removeInventory() method of the *Inventory* class is called, as to reduce the quantity of the InventoryItems that get used as part of the order. In this method, the warn() method is called, which in turn notifies each registered observer (which, in this specific implementation, is only the *LowInventoryObserver*) via the update() method. The update() method of the *LowInventoryObserver* object sends a 'warning' to the system by printing a warning message indicating that there is low inventory of a specific item. It also increases the quantity of said item (which is analogous to ordering more inventory).

Using the observer design pattern allows future maintainers of the system to easily expand and modify the response that the system undertakes when changes to the inventory occur. For example, one could implement an *ExpirationInventoryObserver*, which removes inventory when it expires or orders more inventory that is about to expire. Another addition could be a *InventoryPriceAdjusterObserver*, which changes the price of an *InventoryItem* if the item's quantity exceeds a certain threshold. In cases such as these, all the maintainer has to do is to create a concrete *Observer* object that implements the *Observer* interface, and register it with the *Inventory* object.

Use Case Diagram:



Class Diagram (next page)


MenuItem
+ name:String
+ price:float
+ main:InventoryItem
+ vegetable:InventoryItem
+ side:InventoryItem
OrderItem
+ name:String
+ price:float
InventoryItem
+ name:String
+ type:String
+ quantity:int
+ expiration:Date
+ price:float
+ addQuantity(int)
Special
+ type:String
+ main:InventoryItem
+ vegetable:InventoryItem
+ side:InventoryItem
+ name:String
+ price:float
+ setMain(InventoryItem)
+ setName()
+ setPrice()
+ setSide(InventoryItem)
+ setVegetable(InventoryItem)
<<interface>> Observable
+ register(Observer)
+ unregister(Observer)
+ warn()
<<interface>> Observer
+ update()
LowInventoryObserver
- inventory:Observable
+ update()
Menu
+ menu:ArrayList<MenuItem>
- inventory:Inventory
- buildMenu()
+ getItem(String):MenuItem
Orders
+ orders:ArrayList<OrderItem>
+ addOrder(OrderItem)
Inventory
+ inventory:ArrayList<InventoryItem>
- observers:ArrayList<Observer>
- buildInventory()
+ getItem(String):InventoryItem
+ register(Observer)
+ removeInventory(InventoryItem)
+ unregister(Observer)
+ warn()
Specials
+ specials:ArrayList<Special>
+ addSpecial(Special)
+ getSpecial(String):Special
Tab
+ orders:Orders
+ total:float = 0.0
- getTotal():float
Aggregator
+ menu:Menu
+ orders:Orders
+ tab:Tab
+ specials:Specials
+ inventory:Inventory
+ inventoryObserver:InventoryObserver
+ getInventory():Inventory
+ getMenu():Menu
+ getOrders():Orders
+ getTab():Tab
UserInterface
+ runInterface()
- displayOptions()
SystemInterface
- invoker:Invoker
+ displayMenu():String
+ displaySpecials():String
+ displayTab():String
+ submitOrder():String
Invoker
- agg:Aggregator
+ displayMenu():Menu
+ displaySpecials():Sepcials
+ displayTab():Tab
+ submitOrder(OrderItem):String
Director
- sb:SpecialBuilder
+ build()
+ getSpecial():Special
<<interface>> SpecialBuilder
+ buildMain()
+ buildVegetable()
+ buildSide()
+ buildName
+ buildPrice()
+ getSpecial():Special
DailySpecialBuilder
- special:Special
- inventory:Inventory
+ buildMain()
+ buildVegetable()
+ buildSide()
+ buildName
+ buildPrice()
+ getSpecial():Special
WeeklySpecialBuilder
- special:Special
- inventory:Inventory
- cal:Calendar
+ buildMain()
+ buildVegetable()
+ buildSide()
+ buildName
+ buildPrice()
+ getSpecial():Special
<<interface>> Command
+ execute():Object
CMDDisplayMenu
- agg:Aggregator
+ execute():Menu
CMDSubmitOrder
- agg:Aggregator
- orderedItem:OrderItem
+ execute():String
CMDDisplayTab
- agg:Aggregator
+ execute():Tab
CMDDisplaySpecials
- agg:Aggregator
+ execute():Specials