Bakery
CS3500: Object Oriented Design
Assignment #11
Prof. Schmidt

Team #2
Jesus Cheng - chengpan.j@husky.neu.edu
Daniel Osborne - osborne.d@husky.neu.edu

Table of Contents

Requirements Document

Use Case Scenarios:

In order to minimize user interaction with the system, all user actions will be delegated through the cashier.

Scenario 1: Purchase Processing

1. **CUSTOMER** enters checkout with items they intend to purchase
2. **CLERK** inputs customer info into system
    a. **SYSTEM** checks if customer with same payment information has purchased before
    b. **SYSTEM** creates a new customer profile if one doesn't exist
3. **CLERK** Enters each item and number of items the customer is purchasing
4. **SYSTEM** calculates order total, displays
5. **CUSTOMER** Chooses to pay now or later
6. **CUSTOMER** chooses how much discount points to apply to order
7. **CLERK** inputs customer's pickup date into system
8. **SYSTEM** registers/stores the purchased products to the customer profile.
9. **SYSTEM** adds & displays rewards points earned from purchase.

Scenario 2: Owner Checking Inventory

1. **OWNER** requests inventory report
2. **SYSTEM** displays list of all items in inventory with quantities

Scenario 3: Owner Queries Specific Customer

1. **OWNER** enters a piece of customer identification info.
2. **SYSTEM** searches for matching customer
3. **SYSTEM** prints customer purchases (if a match is found)
4. **SYSTEM** prints reward point balance
5. **SYSTEM** prints contact information
6. **SYSTEM** prints all other relevant information store on customer

Scenario 4: Owner Checking Orders

1. **OWNER** requests order report
2. **SYSTEM** displays all orders

Scenario 5: Owner Updates Customer Infor

1. **OWNER** inputs user current information
2. **OWNER** inputs new user information
3. **SYSTEM** updates user

Functional / Nonfunctional Requirements:

Functional Requirements of the program:

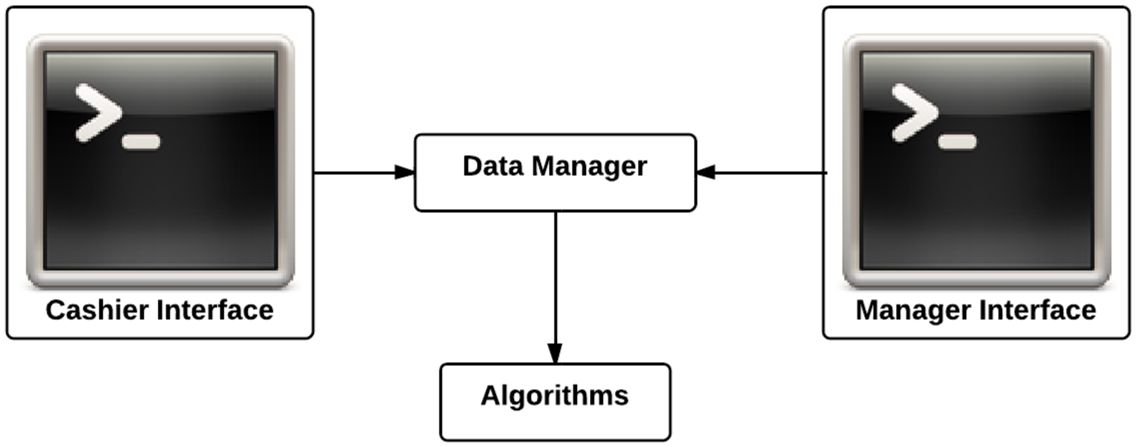
1. It should calculate the order totals.
2. It should store the customer information for contacting and billing the customer.
3. It should track purchases of customers
4. It should track loyalty card point balances.
5. It should give track and give out rewards to loyalty card members based off of purchases.
6. It should allow the owner to see all available items,
7. It should allow the owner to see existing customers
8. It should allow the owner to see existing orders.
9. It should track all items ordered by a given customer.
10. It should allow the owner to modify customer data
11. It should allow the owner to modify order data
12. It should allow the owner to modify inventory data
13. It should provide separate “Cashier” and “Owner” interfaces

Nonfunctional Requirements of the program:

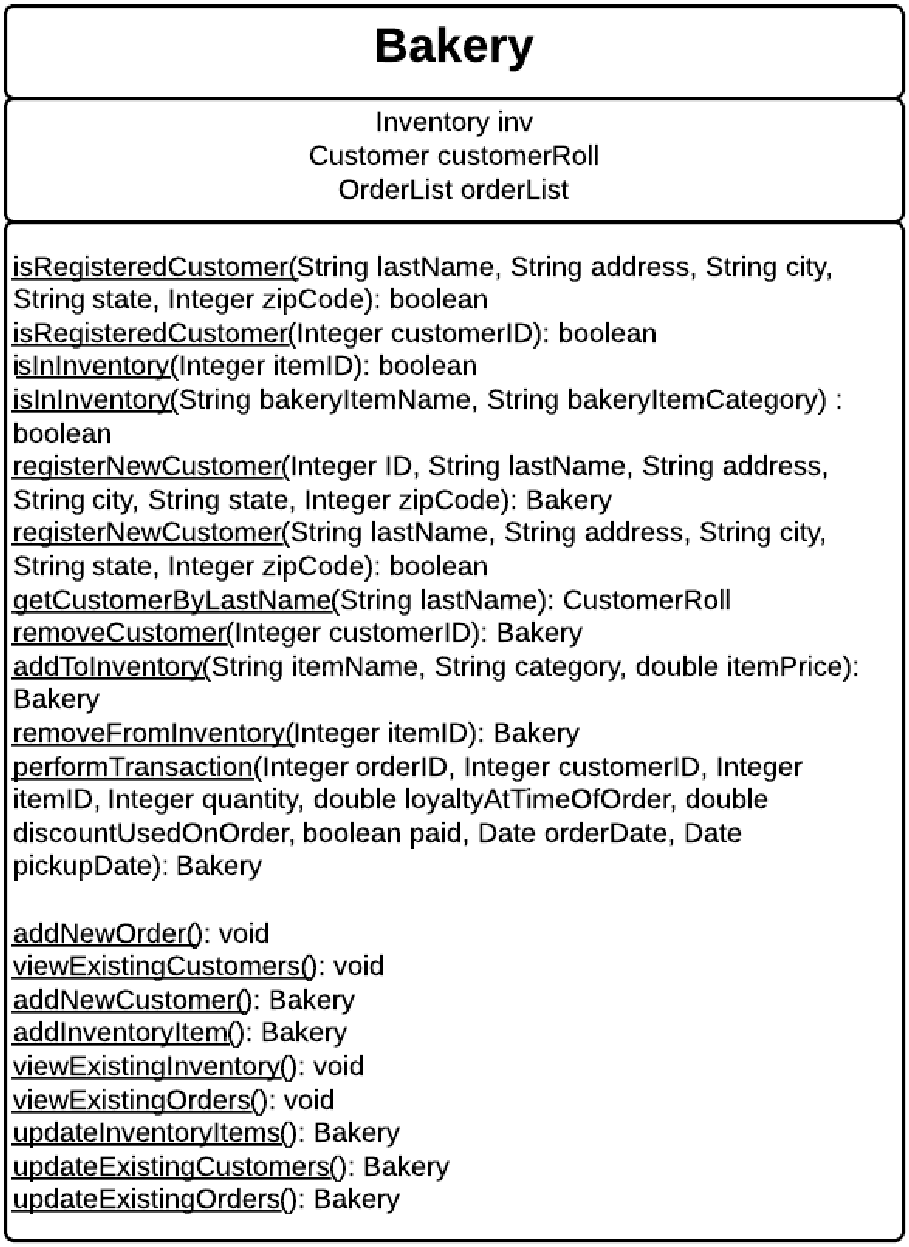
1. It should help the owner to better understand his sales.
2. It should be built in Java.
3. It should run on CCIS system.
4. It should efficiently respond to queries within a reasonable amount of time.
5. It should implement secure class visibility.
6. It should be testable.

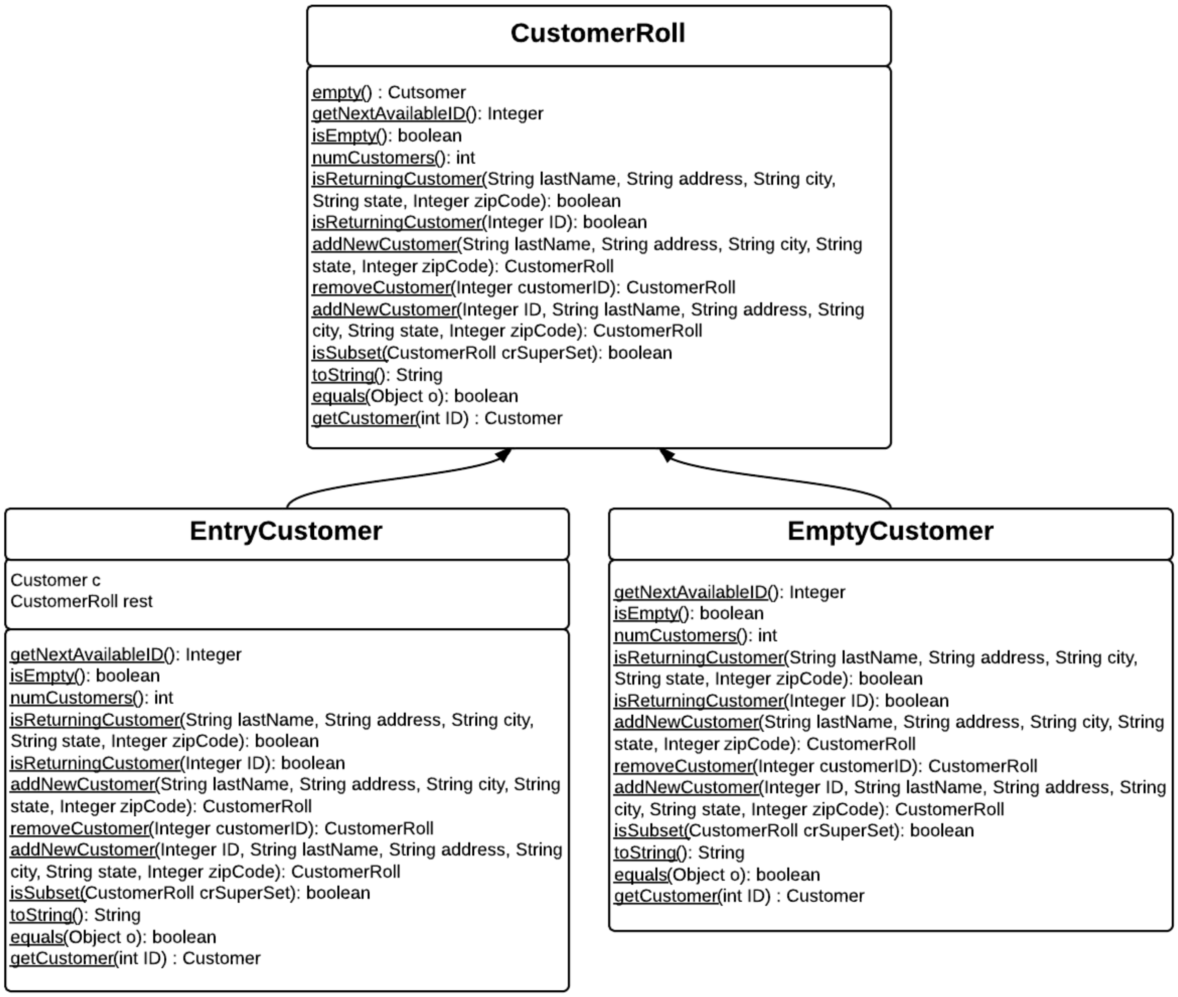
Design Document

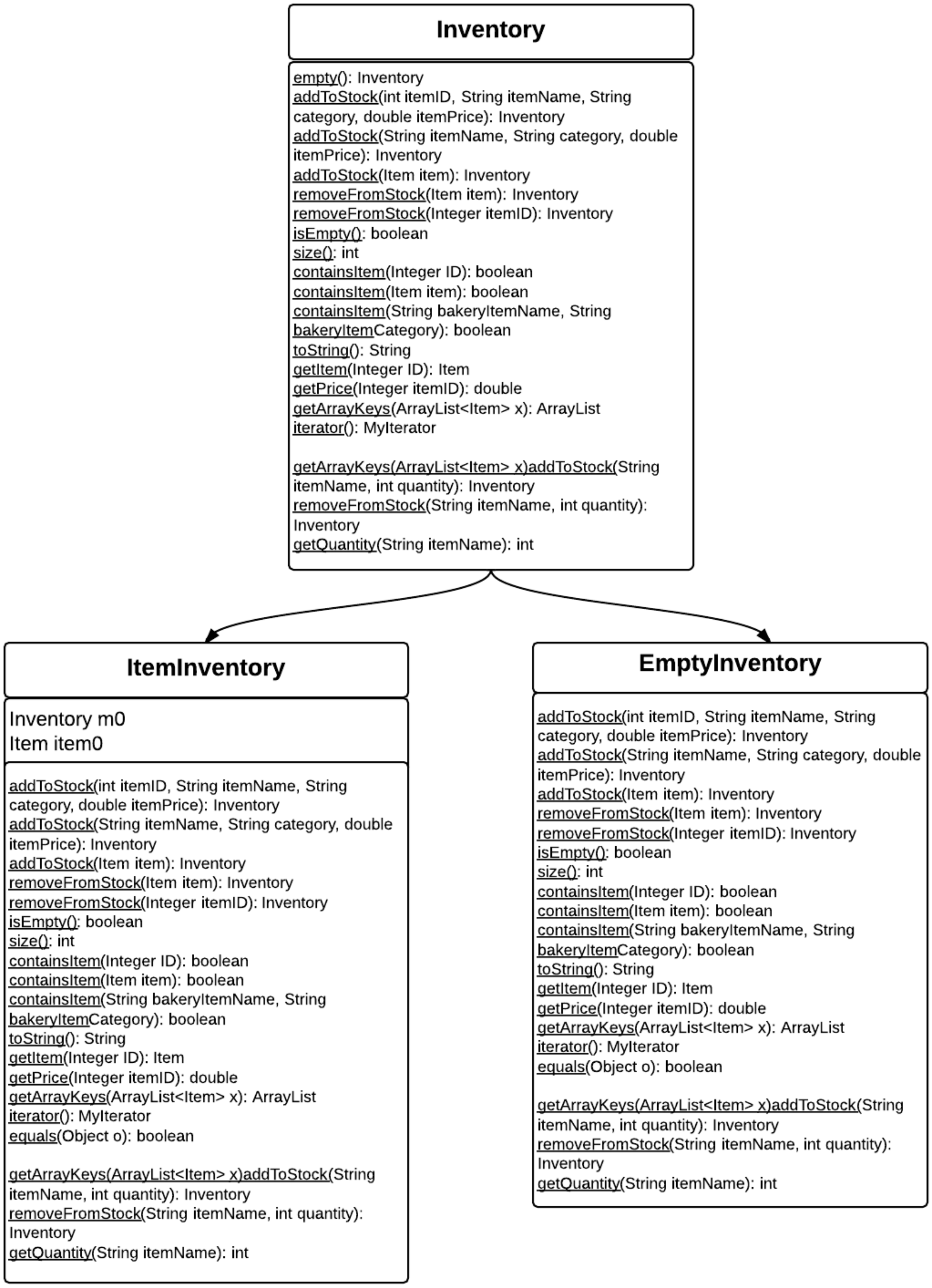
Module Dependency Diagram



UML Diagrams


Bakery
Inventory inv
Customer customerRoll
OrderList orderList
isRegisteredCustomer(String lastName, String address, String city, String state, Integer zipCode): boolean
isRegisteredCustomer(Integer customerID): boolean
isInInventory(Integer itemID): boolean
isInInventory(String bakeryItemName, String bakeryItemCategory) : boolean
registerNewCustomer(Integer ID, String lastName, String address, String city, String state, Integer zipCode): Bakery
registerNewCustomer(String lastName, String address, String city, String state, Integer zipCode): boolean
getCustomerByLastName(String lastName): CustomerRoll
removeCustomer(Integer customerID): Bakery
addToInventory(String itemName, String category, double itemPrice): Bakery
removeFromInventory(Integer itemID): Bakery
performTransaction(Integer orderID, Integer customerID, Integer itemID, Integer quantity, double loyaltyAtTimeOfOrder, double discountUsedOnOrder, boolean paid, Date orderDate, Date pickupDate): Bakery
addNewOrder(): void
viewExistingCustomers(): void
addNewCustomer(): Bakery
addInventoryItem(): Bakery
viewExistingInventory(): void
viewExistingOrders(): void
updateInventoryItems(): Bakery
updateExistingCustomers(): Bakery
updateExistingOrders(): Bakery


CustomerRoll
empty() : Cutsomer
getNextAvailableID(): Integer
isEmpty(): boolean
numCustomers(): int
isReturningCustomer(String lastName, String address, String city, String state, Integer zipCode): boolean
isReturningCustomer(Integer ID): boolean
addNewCustomer(String lastName, String address, String city, String state, Integer zipCode): CustomerRoll
removeCustomer(Integer customerID): CustomerRoll
addNewCustomer(Integer ID, String lastName, String address, String city, String state, Integer zipCode): CustomerRoll
isSubset(CustomerRoll crSuperSet): boolean
toString(): String
equals(Object o): boolean
getCustomer(int ID) : Customer
EntryCustomer
Customer c
CustomerRoll rest
getNextAvailableID(): Integer
isEmpty(): boolean
numCustomers(): int
isReturningCustomer(String lastName, String address, String city, String state, Integer zipCode): boolean
isReturningCustomer(Integer ID): boolean
addNewCustomer(String lastName, String address, String city, String state, Integer zipCode): CustomerRoll
removeCustomer(Integer customerID): CustomerRoll
addNewCustomer(Integer ID, String lastName, String address, String city, String state, Integer zipCode): CustomerRoll
isSubset(CustomerRoll crSuperSet): boolean
toString(): String
equals(Object o): boolean
getCustomer(int ID) : Customer
EmptyCustomer
getNextAvailableID(): Integer
isEmpty(): boolean
numCustomers(): int
isReturningCustomer(String lastName, String address, String city, String state, Integer zipCode): boolean
isReturningCustomer(Integer ID): boolean
addNewCustomer(String lastName, String address, String city, String state, Integer zipCode): CustomerRoll
removeCustomer(Integer customerID): CustomerRoll
addNewCustomer(Integer ID, String lastName, String address, String city, String state, Integer zipCode): CustomerRoll
isSubset(CustomerRoll crSuperSet): boolean
toString(): String
equals(Object o): boolean
getCustomer(int ID) : Customer



Inventory

empty(): Inventory
addToStock(int itemID, String itemName, String category, double itemPrice): Inventory
addToStock(String itemName, String category, double itemPrice): Inventory
addToStock(Item item): Inventory
removeFromStock(Item item): Inventory
removeFromStock(Integer itemID): Inventory
isEmpty(): boolean
size(): int
containsItem(Integer ID): boolean
containsItem(Item item): boolean
containsItem(String bakeryItemName, String bakeryItemCategory): boolean
toString(): String
getItem(Integer ID): Item
getPrice(Integer itemID): double
getArrayKeys(ArrayList<Item> x): ArrayList
iterator(): MyIterator

getArrayKeys(ArrayList<Item> x)addToStock(String itemName, int quantity): Inventory
removeFromStock(String itemName, int quantity): Inventory
getQuantity(String itemName): int

ItemInventory

Inventory m0
Item item0

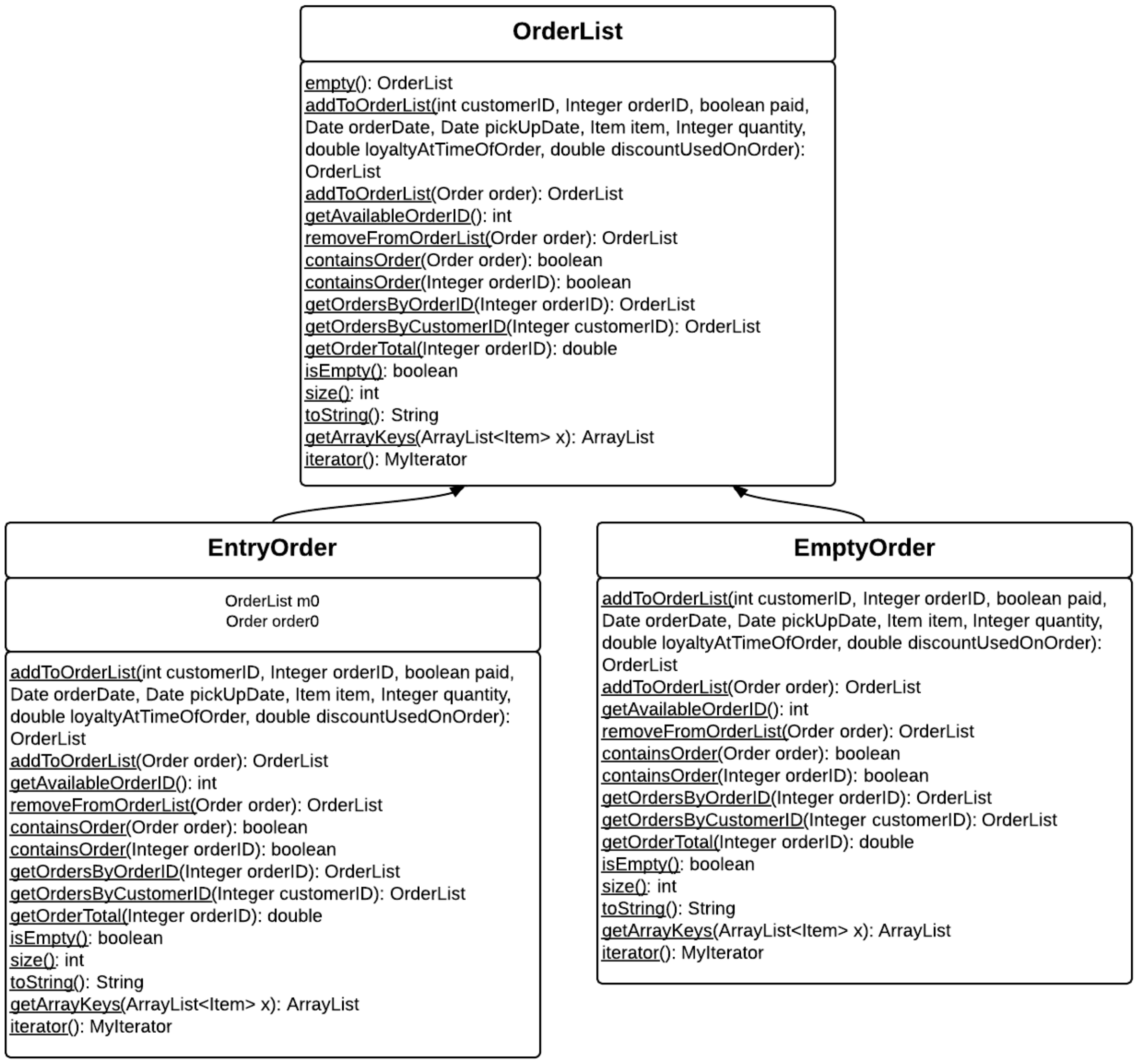
addToStock(int itemID, String itemName, String category, double itemPrice): Inventory
addToStock(String itemName, String category, double itemPrice): Inventory
addToStock(Item item): Inventory
removeFromStock(Item item): Inventory
removeFromStock(Integer itemID): Inventory
isEmpty(): boolean
size(): int
containsItem(Integer ID): boolean
containsItem(Item item): boolean
containsItem(String bakeryItemName, String bakeryItemCategory): boolean
toString(): String
getItem(Integer ID): Item
getPrice(Integer itemID): double
getArrayKeys(ArrayList<Item> x): ArrayList
iterator(): MyIterator
equals(Object o): boolean

getArrayKeys(ArrayList<Item> x)addToStock(String itemName, int quantity): Inventory
removeFromStock(String itemName, int quantity): Inventory
getQuantity(String itemName): int

EmptyInventory

addToStock(int itemID, String itemName, String category, double itemPrice): Inventory
addToStock(String itemName, String category, double itemPrice): Inventory
addToStock(Item item): Inventory
removeFromStock(Item item): Inventory
removeFromStock(Integer itemID): Inventory
isEmpty(): boolean
size(): int
containsItem(Integer ID): boolean
containsItem(Item item): boolean
containsItem(String bakeryItemName, String bakeryItemCategory): boolean
toString(): String
getItem(Integer ID): Item
getPrice(Integer itemID): double
getArrayKeys(ArrayList<Item> x): ArrayList
iterator(): MyIterator
equals(Object o): boolean

getArrayKeys(ArrayList<Item> x)addToStock(String itemName, int quantity): Inventory
removeFromStock(String itemName, int quantity): Inventory
getQuantity(String itemName): int


OrderList
empty(): OrderList
addToOrderList(int customerID, Integer orderID, boolean paid, Date orderDate, Date pickUpDate, Item item, Integer quantity, double loyaltyAtTimeOfOrder, double discountUsedOnOrder): OrderList
addToOrderList(Order order): OrderList
getAvailableOrderID(): int
removeFromOrderList(Order order): OrderList
containsOrder(Order order): boolean
containsOrder(Integer orderID): boolean
getOrdersByOrderID(Integer orderID): OrderList
getOrdersByCustomerID(Integer customerID): OrderList
getOrderTotal(Integer orderID): double
isEmpty(): boolean
size(): int
toString(): String
getArrayKeys(ArrayList<Item> x): ArrayList
iterator(): MyIterator
EntryOrder
OrderList m0
Order order0
addToOrderList(int customerID, Integer orderID, boolean paid, Date orderDate, Date pickUpDate, Item item, Integer quantity, double loyaltyAtTimeOfOrder, double discountUsedOnOrder): OrderList
addToOrderList(Order order): OrderList
getAvailableOrderID(): int
removeFromOrderList(Order order): OrderList
containsOrder(Order order): boolean
containsOrder(Integer orderID): boolean
getOrdersByOrderID(Integer orderID): OrderList
getOrdersByCustomerID(Integer customerID): OrderList
getOrderTotal(Integer orderID): double
isEmpty(): boolean
size(): int
toString(): String
getArrayKeys(ArrayList<Item> x): ArrayList
iterator(): MyIterator
EmptyOrder
addToOrderList(int customerID, Integer orderID, boolean paid, Date orderDate, Date pickUpDate, Item item, Integer quantity, double loyaltyAtTimeOfOrder, double discountUsedOnOrder): OrderList
addToOrderList(Order order): OrderList
getAvailableOrderID(): int
removeFromOrderList(Order order): OrderList
containsOrder(Order order): boolean
containsOrder(Integer orderID): boolean
getOrdersByOrderID(Integer orderID): OrderList
getOrdersByCustomerID(Integer customerID): OrderList
getOrderTotal(Integer orderID): double
isEmpty(): boolean
size(): int
toString(): String
getArrayKeys(ArrayList<Item> x): ArrayList
iterator(): MyIterator

Changes in Design

Originally, we stored Orders in an ArrayList within each customer. We changed this design by creating a separate list structure for orders, so all orders could be searched without having to collect them every time. This also enabled more sensical function names in the order class.

Unfortunately, this created an immense amount of similar code where the same functions were passed along to several layers of objects. For example, bakery, orderList, orderEntry, and Order all have constructors to create new orders. This created a lot of unnecessary repetition. In the end, it probably would have been a better approach to just use ArrayLists and utilize a much simpler design project than the structure we chose.

Furthermore, we realized in the later requirements that all customers were rewards members instead of having two types of customers (members and non-members).
We had originally planned to calculate order balances on the fly, but realized that each order must keep track of the customer’s balances at the time of order, so we stored them as parameters within each order.

We also added many more methods to OrderList, Inventory, CustomerRoll and Bakery to make our program to work and follow the requirements discussed in class.

Group Member Contributions

Dan Osborne and Jesus Cheng worked collaboratively on the project online using Google Documents and GitHub. Both worked together on every file using GitHub. All work was done during in-person meetings and all contributions were done cooperatively.