Bakery System Proposal

Requirements and Design

Team 11

Authors:
William Silva silvaw@zimbra.ccs.neu.edu
Nam Luu Nhat luunhat.n@husky.neu.edu

June 18, 2014

Contents

1 Changelog

While most of our original requirements and designs were adequate, after receiving the actual assignment and attempting to implement it, we were forced to make a few changes.

- The most notable change was definitely in the design and layout of our classes. Previously, we just had the Bakery class that contained the list of available bakery items and list of customers. However, we discovered that this did not fit well with our module dependency diagram as the Bakery simply consisted of a database. We also needed a class for user interaction. This ended up being the Main class which held a static instance of a Bakery to communicate with. Through Main and it's various menu systems, the user is able to interact with the database properly.

- In addition, some of our requirements were found to be incorrect after receiving the more precise assignment 11 online. The most significant flaw, which was actually discovered during presentations, was that our Order class could only hold a single BakeryItem. In reality, customers would be ordering multiple types of items so that field was changed to an ArrayList¡ItemQuantityPair¿ with an ItemQuantityPair holding a bakery item being ordered and the quantity of the item being ordered.

- Lastly, our requirements had some minor mistakes in the use cases as the user needed the ability to update and view different fields than what was previously specified.

2 Requirements

2.1 Constraints:

- The system is implemented in Java 1.6.
- The system is well-designed using object-oriented approaches.
- The system runs appropriately on NEU CCIS Linux machines.
- The user shall interact with the system via console.

2.2 Functional Requirements:

- The system shall be capable of initializing with no data. That is, initialize without any customer or bakery item information.
- The system shall be capable of initializing data from text files:
  - The list of available bakery items can be obtained from a text file in the format:

BakeryItemID	BakeryItemName	Category	Price
[ID 1]	[ITEM NAME 1]	[CATEGORY 1]	[PRICE 1]
...	...	...	...
[ID n]	[ITEM NAME n]	[CATEGORY n]	[PRICE n]

    * `BakeryItemID` - Unique integer.
    * `BakeryItemName` - Name of the bakery item (spaces allowed).
    * `Category` - The type of the bakery item.
    * `Price` - The price in dollars of the item.

    Every new line in the table represents a new line in the text file and the different fields are separated by tabs.

  - The list of available bakery items can be obtained from a text file in a format similar to the format for bakery items above except the new fields, in order, are:
    * `CustomerID` - Unique integer representing the customer.
    * `LastName` - Last name of the customer.
    * `Address` - Address of the customer.

    * `City` - the city where the customer currently resides.
    * `State` - the state where the customer currently resides.
    * `ZipCode` - the zipcode of the customer's current residence.
    * `OrderID` - Unique integer representing the order.
    * `Paid?` -"Yes" or "No" stating if the order has been paid for.
    * `OrderDate` - the date that the item was ordered in the format of "month/date/year" with the year being the last two digits of the year (Example: 1/1/95).
    * `PickupDate` - the date that the item was picked up from the bakery in a format identical to that of OrderDate.
    * `BakeryItemID` - Unique integer.
    * `BakeryItemName` - Name of the bakery item (spaces allowed).
    * `Category` - The type of the bakery item.
    * `Quantity` - Integer representing how many units of the item were ordered.
    * `Price` - The price in dollars of the item.
    * `Total` - The total price of purchasing the quantity of the item (before discounts)
    * `DiscountUsedOnOrder` - Negative number representing the discount used and price decrease.
    * `TotalDue` - The total price of the order (after discounts).
    * `AvailableDiscount` - Positive number representing the total dollar discount that can be used on a future order.
    * `CurrentLoyalty` - the amount out of $100.00 that has been spent towards the next $10.00 discount.

- The system shall store customer contact and billing information, including an integer customer ID, last name, address, city, state, zipcode.

- The system shall be able to update all customer information.

- The system shall track loyalty card information for each customer, including how much discount a customer currently has along with the total they have toward the next discount. Specifically, a customer will receive a $10.00 discount for every $100 they spend, which may be across multiple orders.

- The system shall allow the user to add new users and display receipts that include customer information, information about what was ordered, and the order total.
- The system shall allow the user to update order information.
- The system shall allow the user to add new bakery items to the system by inputting its bakery item ID, name, food category, and price.
- The system shall allow the user to update the information of current bakery items.
- The system shall allow the user to view a customer's loyalty status and contact information.
- The system shall allow the user to view all orders (order ID, order date, pickup date, customer, what was order for each order, quantity of each item, price of each item, order total, discount, total due) filtered in various ways including:
  - By specific customer
  - By what date the order was placed on.
  - By what date the order was picked up or delivered.
  - By what product was ordered.
  - By if the order was paid or not.
- The system shall allow the user to view the list of all bakery items.
- The system shall allow the user to save the state of the system into two files: One containing information about the bakery items while the other contains information about the orders. The format for these text files will be the same format as what the bakery system may initialize as described above.

2.3 Nonfunctional Requirements:

All nonfunctional requirements are specified for a database with no more than 100 bakery items and 100 customers. Each customer has a single order.

- The bakery system shall display the list of available bakery items in 15 seconds or less.
- The bakery system shall display information about a bakery item in 15 seconds or less upon selecting it from the list of bakery items.
- The bakery system shall display customer information in 15 seconds or less.
- The bakery system shall be able to add an order to the order records in 15 seconds or less.

2.4 Use Cases:

2.4.1 Use Case 1 (constructing an empty bakery/initializing with data):

1. The user starts the system.
2. The system asks the user if they would like to start a new state or initialize the system state from text files.
   (a) If the user decides on a new state, the system will initialize with no bakery items, customers, or orders.
   (b) If the user decides to load the system state from text files, they will have to enter in the name of the text file with the available bakery items and the name of the text file with the list of all orders. The system will load using this data.

2.4.2 Use Case 2 (viewing information on, adding, updating customer information):

1. The system displays a main menu with all the options the user may select.
2. The user selects the option to perform operations on customers.
3. The system displays what operations may be performed on users.
   (a) The user selects the option to view/update customer information.

i. The system asks the user if they would like to view a list of all customers or search for a specific customer.
ii. The user picks an option to filter the customers.
iii. The system displays all customers who fit the criteria.
iv. The user picks a customer to display information about.
v. The system displays information on the customer including the customer ID, last name, address, city, state, and zip code.
    A. The user may select a prompt to update any of this information.
vi. The user may select an option to view the loyalty status of the customer.
    A. Current loyalty status is displayed including any available discounts and the number of dollars they have out of $100.00 until they get another discount.

(a) The user selects the option to add customer information:
    i. The system asks the user to input information about the customer including ID, last name, address, city, state, and zipcode.
    ii. The user enters this information in and the system adds the customer to the system. The added customer currently has no orders associated with it.

2.4.3 Use Case 3 (adding a new order):

1. The system displays the main menu.
2. The user selects the option to perform operations with bakery orders.
3. The system displays the actions available with bakery orders.
4. The user selects the option to add a bakery order.
5. The system displays a list of items that may be purchased.
6. The user selects an item.
7. The system asks for the quantity of that item.

8. The user inputs a number specifying how many of that item is being purchased.

9. The system asks the user if they would like to order any more items.

    (a) If yes, the user repeats the previous steps 5-8 to add another item and quantity to the order.

10. The system asks if the order is by an existing customer or a new customer:

    (a) If existing customer is selected, the system will ask how the user wants to search for the customer in the database. This includes last name.

        i. The user selects a way to search for the existing customer information and selects the correct customer from the results. The order will be in their name.

    (b) If the user selects new customer, then they will have to enter in all of the proper customer information including last name, address, city, state, and zip code. An ID will automatically be assigned to them.

11. The system adds the order to the database with the associated customer and displays an order receipt.

12. The system asks if the user would like to make another order.

    (a) If the user says yes, go to step 2.

    (b) If the user says no, the system returns to the main menu.

2.4.4 Use Case 4 (viewing and updating order information):

1. The system displays the main menu.

2. The user selects the option to perform operations on bakery orders.

3. The system displays the available actions for bakery orders.

4. The user selects the option to view orders.

5. The system asks the user if they would like to view all orders or filter by customer, date order was placed, date order was picked up, product ordered, or if they are unpaid.
6. The user picks one of the options, entering any necessary information to filter.
7. The system displays the appropriate orders in a list in order from most recent to least recent.
8. The user selects an order.
9. The system displays the order information (order ID, order date, pickup date, customer, what was order for each order, quantity of each item, price of each item, order total, discount, total due).
    - The user may chose to update any order information here.
10. The user selects the option to go back to the list of orders.
11. Previous steps are repeated as necessary.
12. The user selects the option to go back to the main menu.

2.4.5 Use Case 5 (viewing information on and updating available items):

1. The system displays a main menu with all actions the user may select.
2. The user selects the option to perform operations on bakery items.
3. The system asks the user what they would like to do with bakery items.
4. The user selects the option to view all of the available bakery items.
5. The system displays all of the available bakery items in alphabetical order.
    (a) The user may select an option to sort by category if they please.
6. The user selects an item to get more information about.

7. The system displays more information about the item, including the ID, name, category, and price.
8. The user may select the option to update information about the bakery item.
9. The user selects the option to go back to the list of available bakery items.
10. Steps 3-6 may be repeated as many times as the user likes.
11. The user selects the option to go back to the main menu.

2.4.6 Use Case 6 (saving the system):

1. The system displays the main menu.
2. The user selects the option to save the current state of the system.
3. The system prompts the user for both the file name of the text file to contain a list of all available bakery items and the text file to contain the list of all the orders.
4. The user enters in the file names.
5. The system saves the system to these files, creating new files if necessary or overwritting the previous ones.
6. The system returns to the main menu.

2.5 Scenarios:

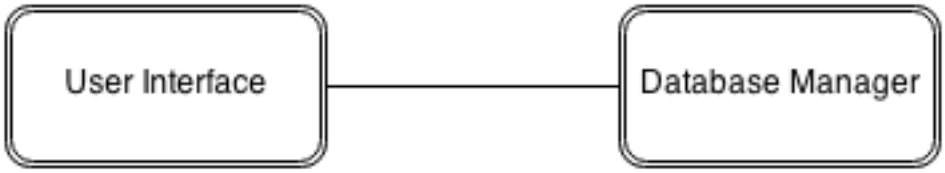
- The user attempts to initialize from a file that does not exist. The system reports that the file is not there and asks for another file name.
- The user says that the customer is ordering a quantity of items that is zero or fewer. The system responds that there was a mistake and asks the user for input again.
- As the user is placing an order, they search for an ID that does not exist in the system. The system alerts them to this and asks for another ID to search for.

2.6 User stories:

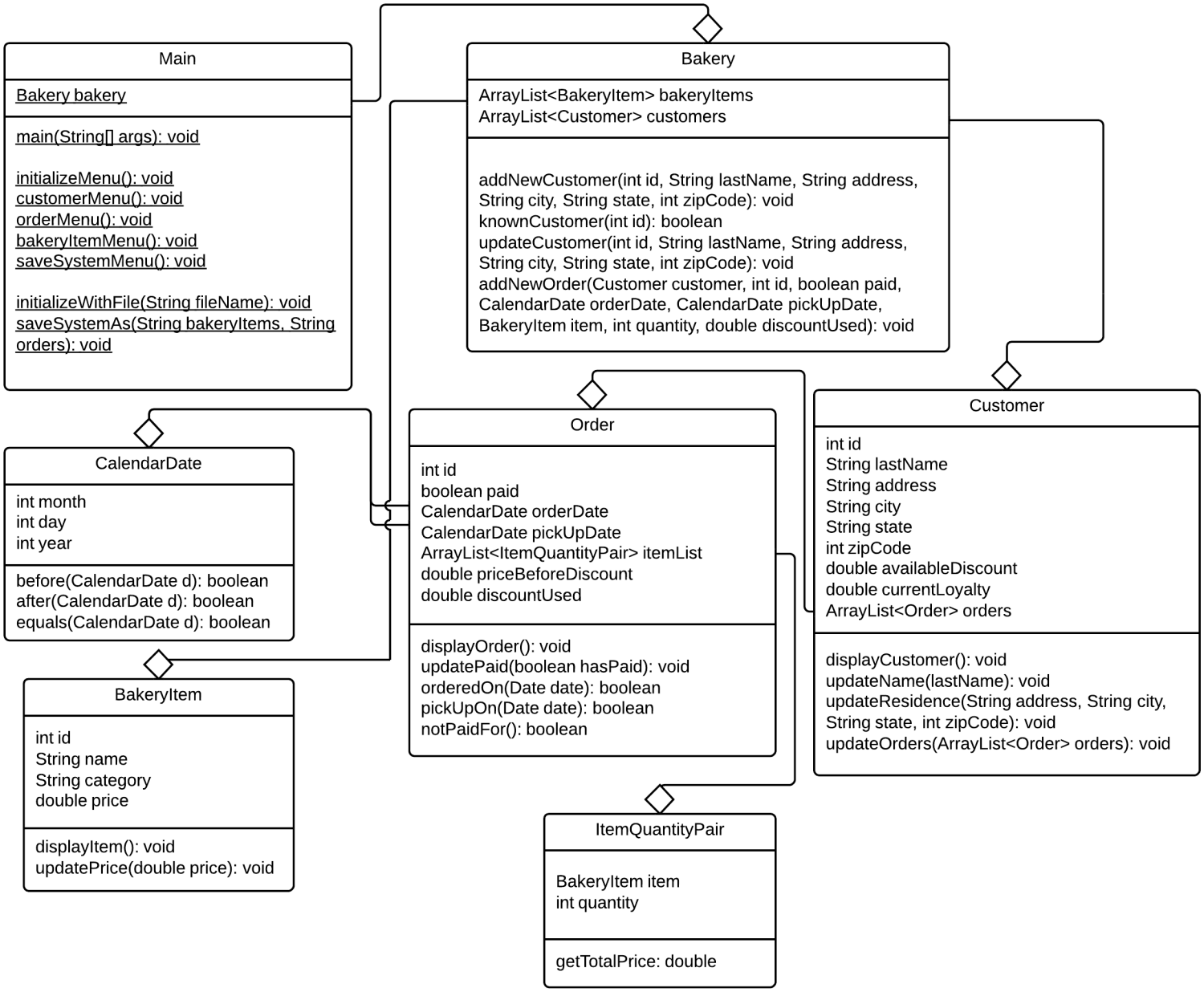
- As a user, I can see a list of all items being sold at the bakery.
- As a user, I can search for a specific customer by name and view all bakery items they have ordered in a list.
- As a user, I can place an order and see the receipt.
- As a user, I check the loyalty card status of a customer and see how close they are to receiving a discount, and how many discounts they currently have.

3 Design

3.1 Module Dependency Diagram



3.2 UML Diagram



4 Team Member Contributions

Each team member of our group contributed an approximately equal amount of work to the assignment. The documents were originally completed together when the pair meet up to do work. Minor updates following the posting of assignment 11 were made by William. In order to separate the work equally while still being efficient for the actual implementation, each team member worked on a separate module. William worked on the User Interface while Nam coded the Database Manager. All work was shared through Github with the partners communicating for the implementation process. After the implementation was finished, both partners worked on adding various test cases to ensure that the classes were as bug-free as possible.