**Product Design**

|  |  |
| --- | --- |
| **Team** | s361-02b, Team B  Jedd Haberstro  Obaseki Idemudia  Gregg Miller  Johanna Calderon  Jenny Zhen  Richard Sperrazza |

# Architectural Model

# 

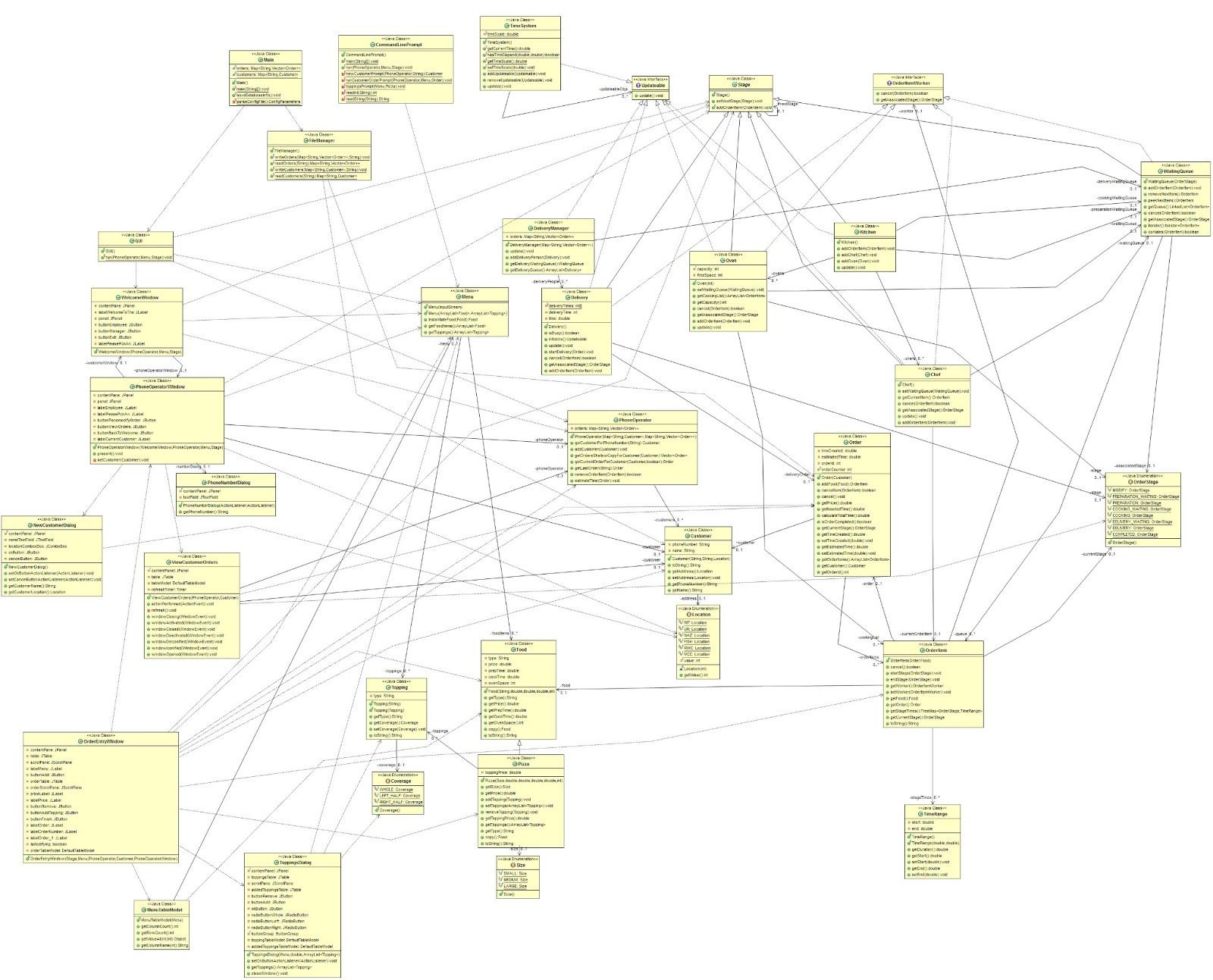
**The GUI and CLI represent the front-end the user interacts with. The GUI and CLI interact with the phone operator logic to create or modify new orders. Once the GUI or CLI places an order, it is registered with the report system and forwarded to the kitchen. The kitchen then appropriately processes the order and then forwards the order to the delivery system to have the order delivered.**

# Components and Functions



|  |  |
| --- | --- |
| **Menu** | Component state   * Holds the available items for selection/purchase by the customer by their names, prices, preparation times, and cooking times.   Component behavior   * It also restricts what a phone operator can allow a customer to order * Parses menu file. |
| **Report System** | Component state   * Contains all details of completed orders (items, time taken, price paid) in the database   Component behavior   * Managed from GUI or CLI by a manager to get the reports |
| **Phone Operator Logic** | Component state   * Reference to order/customer database   Component behavior   * Sends orders to the kitchen * Acts an an intermediary between the user and the order database. * Estimates time for order completion * Can modify/cancel orders |
| **Kitchen System** | Component state   * List of cooks * List of ovens * List of orders that need to be prepared or cooked.   Component behavior   * Determines if a cook is free or busy * Sends prepared/cooked orders to delivery * Prepares food for required times * Determines space in oven(s) and places required items in oven(s) |
| **Delivery System** | Component state   * List of drivers * List of orders ready to be delivered   Component behavior   * Reports finished orders to the report system * Takes finished orders and delivers them, taking the appropriate amount of time |
| **Main** | Component state   * None   Component behavior   * Parses config file and creates the specified number of cook, ovens, and drivers. * Initializes and launches all other systems. |

# Class Diagram(s)



# Sequence Diagram(s)

# 

# Design Rationale

The first debate was on how time would be represented and structured, since it governs the management of nearly every other component of the code. The major ideas were between running each order as a thread and having a central loop to call for updates on components. While the former seemed more accurate with regards to time, the methods we would need to use to control each thread seemed to outweigh the benefits - plus, a few of us had little experience with threads, and the rest weren’t entirely confident about them. The update loop was decided on for this reason, and after it was designed, it was found to run accurately - it is still up in the air for whether or not it will work properly with every module (as we haven’t gotten that far yet), but hopes are high. If there’s a catastrophic failure, reworking it to use threads will be the back-up plan.

We do recognize however there is still the need for threading so that customer interactions can happen concurrently to the rest of the system. There will be two threads: a thread for the user interfacing components (the GUI or command line interface interaction) and a “main thread” that runs the central loop. As the user/customer interacts with the GUI or CLI, information will be dispatched from that thread to the main thread, so the rest of the application can process the customer’s request, perform deliveries, etc..

Other than that major part of the code, the decision was made to create classes for food items, cooks, ovens, delivery people, phone operators, and orders. There was little argument or even alternative ideas for this - it was unanimous. Food would be worked in to suit both regular items and pizzas (which can have toppings on either half), and the food objects can be placed into a list within an order. Phone operators would create orders by doing so, and send it to a queue for waiting cooks to pick up from, where the cook object would interact with the oven(s) and end up sending it to the delivery queue for a delivery person to pick it up. Separating these classes was mainly on an Object-Oriented design approach, as well as allowing for the coding effort to be split among group members easily.

After Release 1, we found that our design strategy and plan proved to be efficient. The missing Release 2 functionalities were all implemented in a straight-forward manner - their structure was already accounted for in previous planning sessions, so their actual implementation and code worked smoothly into and around the existing code. The GUI was also implemented close to our initial plans, only being modified and touched up to suit the errors found in testing sessions by Team A and the Freshman Seminar.

There was only one design mistake we missed and were unable to fix: our time stamp method was not well suited as to be converted into a date format, and that has caused a lot of time formatting issues in our Manager GUI.

**Changes**

10/8/2012 - Updated with new architecture diagram explanation and simplification, updated UML class diagram, and new sequence diagram for the chef preparation sequence.

11/03/2012 - Updated UML Diagram and the design rationale from R2

11/06/2012 - Updated design rationale to highlight Manager GUI oversights.