Assignment 2 Report

COS80007 – Advanced Java

Abdul Moiz – 101727106

Alexander McWhae - 101801822

Mirza Akbar Beg - 101439582

Contents

Introduction

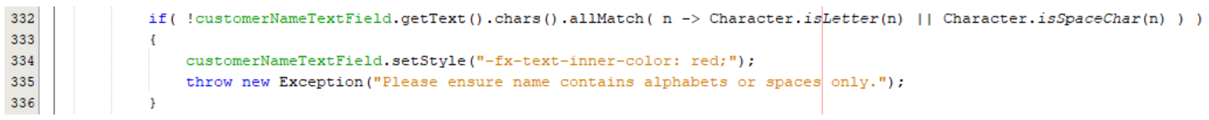
This software is a restaurant ordering and billing system and has been developed in Java. For implementing the Graphical User Interface, JavaFX has been used. This software enables the users in a restaurant to perform a list of activities such as placing an order, preparing it and billing the order at the end.

Two versions of this software have been implemented. The first version is the standalone prototype. The second one is based on the client-server communication and also involves database handling. In this version, three different instances of the application are run (customer, chef and biller) and these instances communicate with each other. This report contains an overview of the work done during the development of the software and also contains sequential screenshots of the application at the end.

Lambdas and Streams in Validations

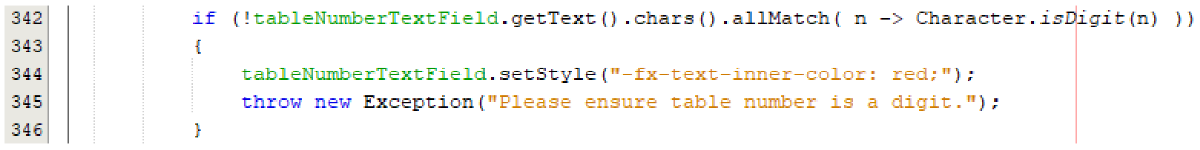
Lambdas & Stream validations have been implemented in the file CustomerFXMLController.java at line numbers shown below:

Line number 332:



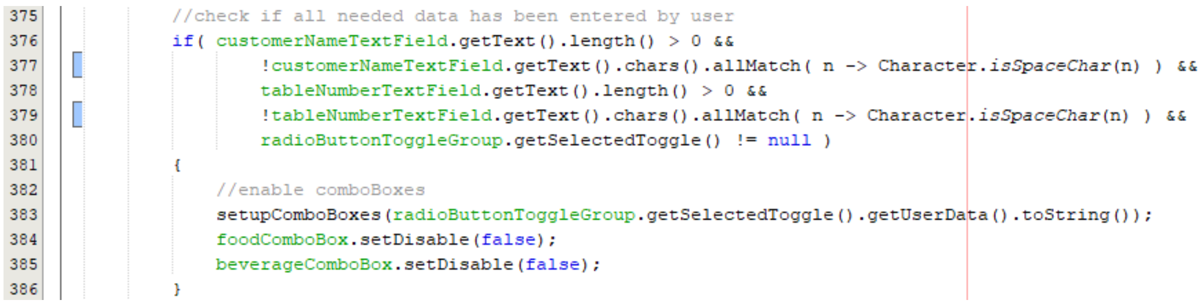
```
332             if( !customerNameTextField.getText().chars().allMatch( n -> Character.isLetter(n) || Character.isSpaceChar(n) ) )
333             {
334                 customerNameTextField.setStyle("-fx-text-inner-color: red;");
335                 throw new Exception("Please ensure name contains alphabets or spaces only.");
336             }
```

Line number 342:



```
342             if (!tableNumberTextField.getText().chars().allMatch( n -> Character.isDigit(n) ))
343             {
344                 tableNumberTextField.setStyle("-fx-text-inner-color: red;");
345                 throw new Exception("Please ensure table number is a digit.");
346             }
```

Line number 377 and 379:



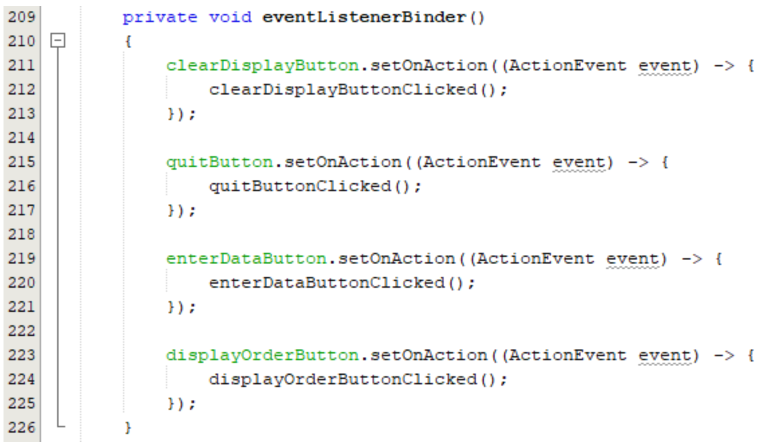
```
            //check if all needed data has been entered by user
            if( customerNameTextField.getText().length() > 0 &&
                    !customerNameTextField.getText().chars().allMatch( n -> Character.isSpaceChar(n) ) &&
                    tableNumberTextField.getText().length() > 0 &&
                    !tableNumberTextField.getText().chars().allMatch( n -> Character.isSpaceChar(n) ) &&
                    radioButtonToggleGroup.getSelectedToggle() != null )
            {
                //enable comboBoxes
                setupComboBoxes(radioButtonToggleGroup.getSelectedToggle().getUserData().toString());
                foodComboBox.setDisable(false);
                beverageComboBox.setDisable(false);
            }
```

Lambdas in Event Handling

Lambdas have been used in event handling as shown by screenshots below:

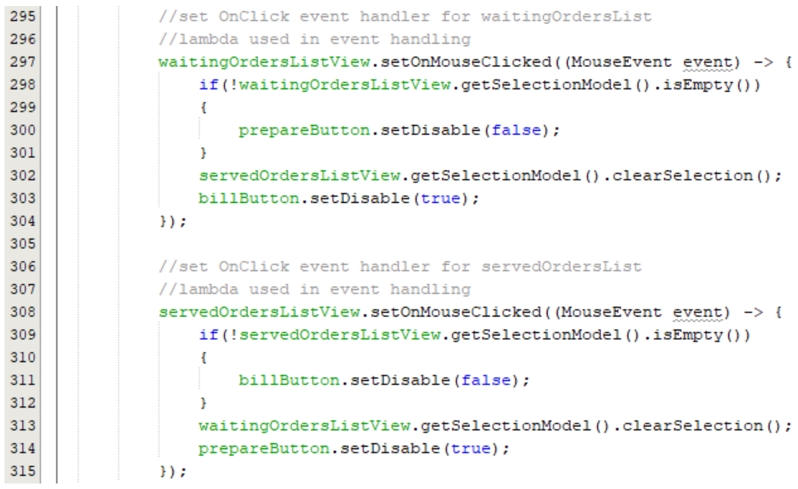
In file CustomerFXMLController.java:

For handling button actions:



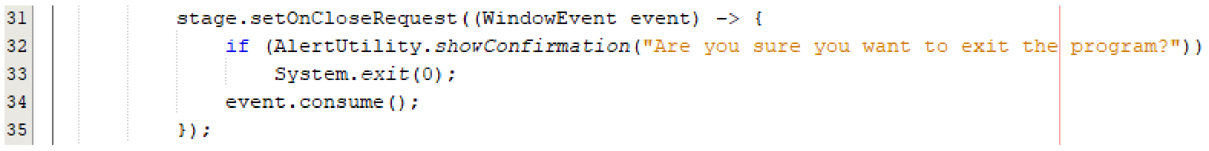
```
    private void eventListenerBinder()
    {
        clearDisplayButton.setOnAction((ActionEvent event) -> {
            clearDisplayButtonClicked();
        });

        quitButton.setOnAction((ActionEvent event) -> {
            quitButtonClicked();
        });

        enterDataButton.setOnAction((ActionEvent event) -> {
            enterDataButtonClicked();
        });

        displayOrderButton.setOnAction((ActionEvent event) -> {
            displayOrderButtonClicked();
        });
    }
```

For handling mouse OnClick events:



```
295         //set OnClick event handler for waitingOrdersList
296         //lambda used in event handling
297         waitingOrdersListView.setOnMouseClicked((MouseEvent event) -> {
298             if(!waitingOrdersListView.getSelectionModel().isEmpty())
299             {
300                 prepareButton.setDisable(false);
301             }
302             servedOrdersListView.getSelectionModel().clearSelection();
303             billButton.setDisable(true);
304         });
305 
306         //set OnClick event handler for servedOrdersList
307         //lambda used in event handling
308         servedOrdersListView.setOnMouseClicked((MouseEvent event) -> {
309             if(!servedOrdersListView.getSelectionModel().isEmpty())
310             {
311                 billButton.setDisable(false);
312             }
313             waitingOrdersListView.getSelectionModel().clearSelection();
314             prepareButton.setDisable(true);
315         });
```

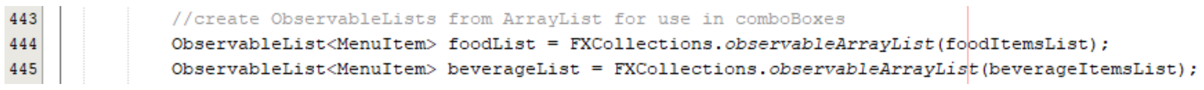
In file AdvancedJava_A2.java:



```
31          stage.setOnCloseRequest((WindowEvent event) -> {
32              if (AlertUtility.showConfirmation("Are you sure you want to exit the program?"))
33                  System.exit(0);
34              event.consume();
35          });
```

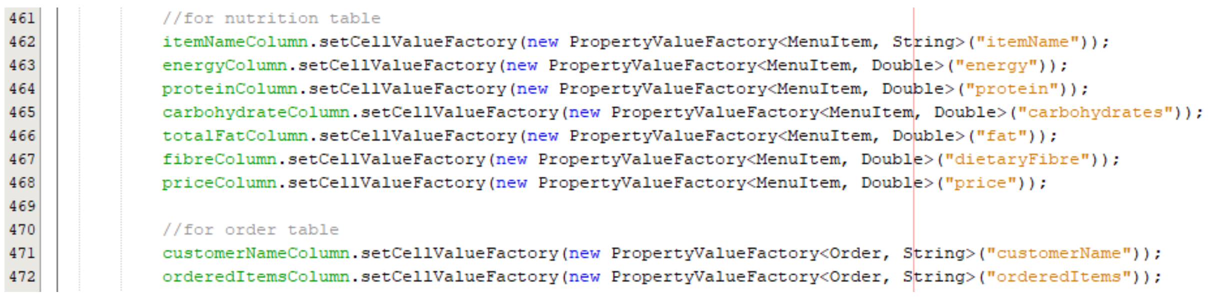
Collections/Generic Methods

Collections have been used in file CustomerFXMLController.java at line numbers shown by screenshots below:



```
443         //create ObservableLists from ArrayList for use in comboBoxes
444         ObservableList<MenuItem> foodList = FXCollections.observableArrayList(foodItemsList);
445         ObservableList<MenuItem> beverageList = FXCollections.observableArrayList(beverageItemsList);
```

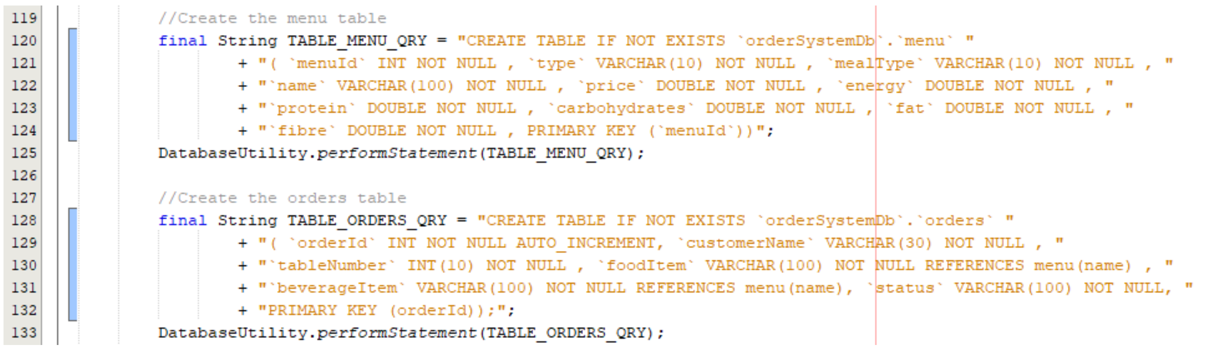
Generic Methods have also been used as shown below:



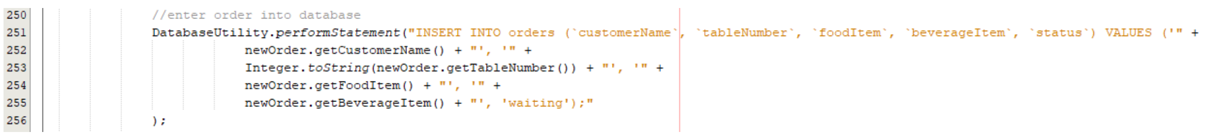
```
461         //for nutrition table
462         itemNameColumn.setCellValueFactory(new PropertyValueFactory<MenuItem, String>("itemName"));
463         energyColumn.setCellValueFactory(new PropertyValueFactory<MenuItem, Double>("energy"));
464         proteinColumn.setCellValueFactory(new PropertyValueFactory<MenuItem, Double>("protein"));
465         carbohydrateColumn.setCellValueFactory(new PropertyValueFactory<MenuItem, Double>("carbohydrates"));
466         totalFatColumn.setCellValueFactory(new PropertyValueFactory<MenuItem, Double>("fat"));
467         fibreColumn.setCellValueFactory(new PropertyValueFactory<MenuItem, Double>("dietaryFibre"));
468         priceColumn.setCellValueFactory(new PropertyValueFactory<MenuItem, Double>("price"));
469 
470         //for order table
471         customerNameColumn.setCellValueFactory(new PropertyValueFactory<Order, String>("customerName"));
472         orderedItemsColumn.setCellValueFactory(new PropertyValueFactory<Order, String>("orderedItems"));
```

Database and Multi-threading

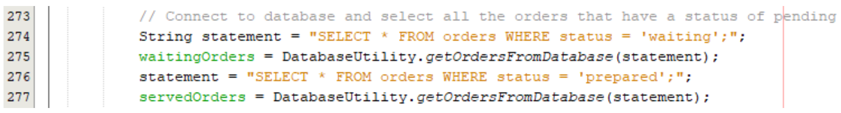
A database has been setup and is used to store information about orders. When a customer places an order, it is uploaded to the database as ‘pending’. The chef application then downloads this order from the database. Once the order is prepared by the chef it is uploaded back to the database as ‘served’. The biller application then downloads this order and upon billing, updates it in the database. Screenshots are shown below:



```
//Create the menu table
final String TABLE_MENU_QRY = "CREATE TABLE IF NOT EXISTS `orderSystemDb`.`menu` "
        + "( `menuId` INT NOT NULL , `type` VARCHAR(10) NOT NULL , `mealType` VARCHAR(10) NOT NULL , "
        + "`name` VARCHAR(100) NOT NULL , `price` DOUBLE NOT NULL , `energy` DOUBLE NOT NULL , "
        + "`protein` DOUBLE NOT NULL , `carbohydrates` DOUBLE NOT NULL , `fat` DOUBLE NOT NULL , "
        + "`fibre` DOUBLE NOT NULL , PRIMARY KEY (`menuId`))";
DatabaseUtility.performStatement(TABLE_MENU_QRY);

//Create the orders table
final String TABLE_ORDERS_QRY = "CREATE TABLE IF NOT EXISTS `orderSystemDb`.`orders` "
        + "( `orderId` INT NOT NULL AUTO_INCREMENT, `customerName` VARCHAR(30) NOT NULL , "
        + "`tableNumber` INT(10) NOT NULL , `foodItem` VARCHAR(100) NOT NULL REFERENCES menu(name) , "
        + "`beverageItem` VARCHAR(100) NOT NULL REFERENCES menu(name), `status` VARCHAR(100) NOT NULL, "
        + "PRIMARY KEY (orderId));";
DatabaseUtility.performStatement(TABLE_ORDERS_QRY);
```

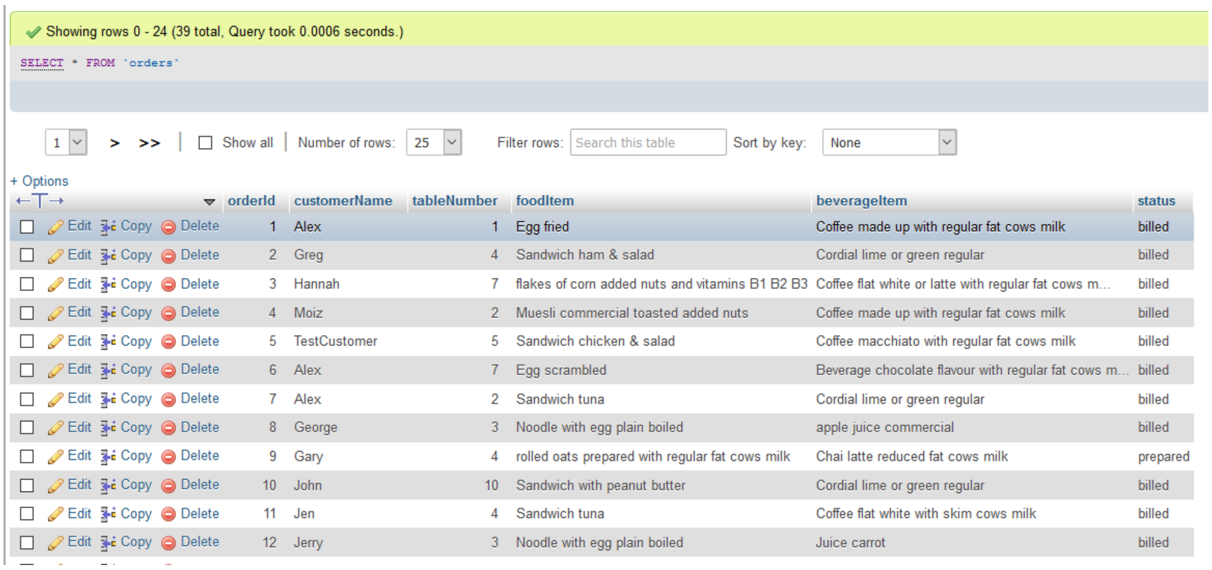


```
//enter order into database
DatabaseUtility.performStatement("INSERT INTO orders (`customerName`, `tableNumber`, `foodItem`, `beverageItem`, `status`) VALUES ('" +
            newOrder.getCustomerName() + "', '" +
            Integer.toString(newOrder.getTableNumber()) + "', '" +
            newOrder.getFoodItem() + "', '" +
            newOrder.getBeverageItem() + "', 'waiting');"
);
```



```
// Connect to database and select all the orders that have a status of pending
String statement = "SELECT * FROM orders WHERE status = 'waiting';";
waitingOrders = DatabaseUtility.getOrdersFromDatabase(statement);
statement = "SELECT * FROM orders WHERE status = 'prepared';";
servedOrders = DatabaseUtility.getOrdersFromDatabase(statement);
```

Screenshot showing orders in the database schema:



✓ Showing rows 0 - 24 (39 total, Query took 0.0006 seconds.)

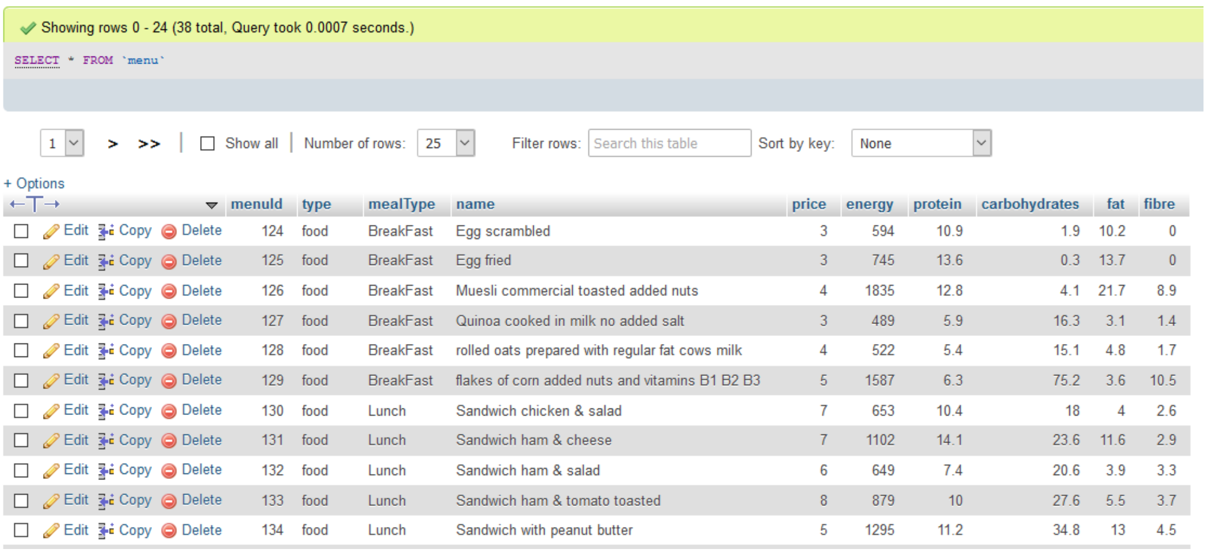
SELECT * FROM `orders`

1 > >> | Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

+ Options

	orderId	customerName	tableNumber	foodItem	beverageItem	status
Edit Copy Delete	1	Alex	1	Egg fried	Coffee made up with regular fat cows milk	billed
Edit Copy Delete	2	Greg	4	Sandwich ham & salad	Cordial lime or green regular	billed
Edit Copy Delete	3	Hannah	7	flakes of corn added nuts and vitamins B1 B2 B3	Coffee flat white or latte with regular fat cows m...	billed
Edit Copy Delete	4	Moiz	2	Muesli commercial toasted added nuts	Coffee made up with regular fat cows milk	billed
Edit Copy Delete	5	TestCustomer	5	Sandwich chicken & salad	Coffee macchiato with regular fat cows milk	billed
Edit Copy Delete	6	Alex	7	Egg scrambled	Beverage chocolate flavour with regular fat cows m...	billed
Edit Copy Delete	7	Alex	2	Sandwich tuna	Cordial lime or green regular	billed
Edit Copy Delete	8	George	3	Noodle with egg plain boiled	apple juice commercial	billed
Edit Copy Delete	9	Gary	4	rolled oats prepared with regular fat cows milk	Chai latte reduced fat cows milk	prepared
Edit Copy Delete	10	John	10	Sandwich with peanut butter	Cordial lime or green regular	billed
Edit Copy Delete	11	Jen	4	Sandwich tuna	Coffee flat white with skim cows milk	billed
Edit Copy Delete	12	Jerry	3	Noodle with egg plain boiled	Juice carrot	billed

The menu data provided in the CSV files have also been uploaded to the database as shown by the screenshot below:



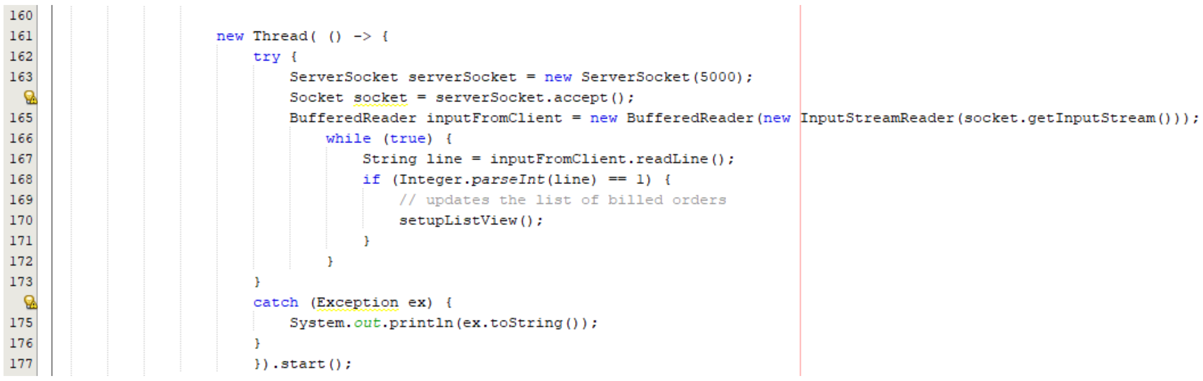
✓ Showing rows 0 - 24 (38 total, Query took 0.0007 seconds.)

```
SELECT * FROM `menu`
```

	menuId	type	mealType	name	price	energy	protein	carbohydrates	fat	fibre
Edit Copy Delete	124	food	BreakFast	Egg scrambled	3	594	10.9	1.9	10.2	0
Edit Copy Delete	125	food	BreakFast	Egg fried	3	745	13.6	0.3	13.7	0
Edit Copy Delete	126	food	BreakFast	Muesli commercial toasted added nuts	4	1835	12.8	4.1	21.7	8.9
Edit Copy Delete	127	food	BreakFast	Quinoa cooked in milk no added salt	3	489	5.9	16.3	3.1	1.4
Edit Copy Delete	128	food	BreakFast	rolled oats prepared with regular fat cows milk	4	522	5.4	15.1	4.8	1.7
Edit Copy Delete	129	food	BreakFast	flakes of corn added nuts and vitamins B1 B2 B3	5	1587	6.3	75.2	3.6	10.5
Edit Copy Delete	130	food	Lunch	Sandwich chicken & salad	7	653	10.4	18	4	2.6
Edit Copy Delete	131	food	Lunch	Sandwich ham & cheese	7	1102	14.1	23.6	11.6	2.9
Edit Copy Delete	132	food	Lunch	Sandwich ham & salad	6	649	7.4	20.6	3.9	3.3
Edit Copy Delete	133	food	Lunch	Sandwich ham & tomato toasted	8	879	10	27.6	5.5	3.7
Edit Copy Delete	134	food	Lunch	Sandwich with peanut butter	5	1295	11.2	34.8	13	4.5

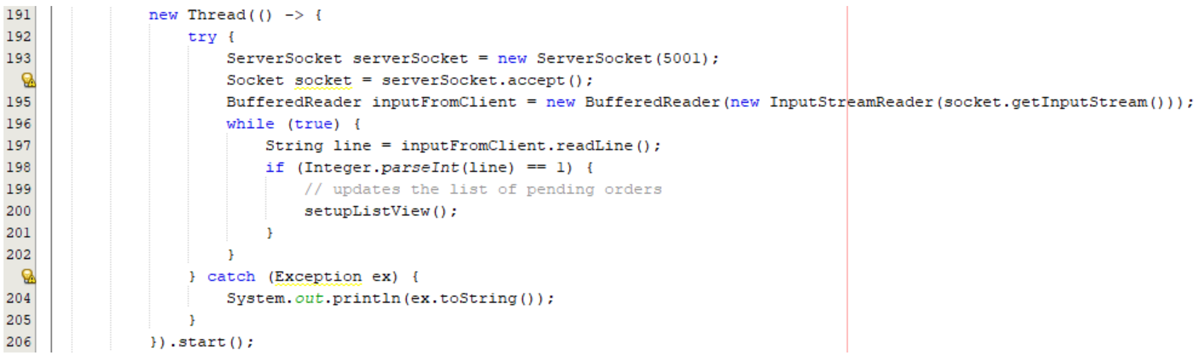
Multithreading:

When the chef application connects to the customer application it starts a new thread:



```
new Thread( () -> {
    try {
        ServerSocket serverSocket = new ServerSocket(5000);
        Socket socket = serverSocket.accept();
        BufferedReader inputFromClient = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            while (true) {
                String line = inputFromClient.readLine();
                if (Integer.parseInt(line) == 1) {
                    // updates the list of billed orders
                    setupListView();
                }
            }
    }
    catch (Exception ex) {
        System.out.println(ex.toString());
    }
    }).start();
```

Similarly, the biller application starts a new thread when connecting to the Chef:



```
new Thread(() -> {
    try {
        ServerSocket serverSocket = new ServerSocket(5001);
        Socket socket = serverSocket.accept();
        BufferedReader inputFromClient = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        while (true) {
            String line = inputFromClient.readLine();
            if (Integer.parseInt(line) == 1) {
                // updates the list of pending orders
                setupListView();
            }
        }
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }
}).start();
```

MVC

The design of the application is based on the MVC architecture. The data is stored in the model, the UI is the view, and the business logic is contained in the controller. For this application files belonging to the M, V, and C categories are listed below:

Standalone Version:

Model:

MenuItem.java – class for containing menu items

Order.java – class for containing orders

View:

CustomerFXML.fml – The main UI view for the application

FXMLDocument.fxml – The view preceding the main view, contains a 'Begin' button.

Controller:

CustomerFXMLController.java – The main business logic class of the program

FXMLDocumentController.java – Contains logic for displaying the first screen

AdvancedJava_A2.java – Contains logic for launching the application.

Client-Server Version:

Model:

MenuItem.java – class for containing menu items

Order.java – class for containing orders

View:

CustomerClientServerFXML.fml – The UI view for Customer mode of the application

ChefClientServerFXML.fml – The UI view for Chef mode of the application

BillerClientServerFXML.fml – The UI view for Biller mode of the application.

FXMLDocument.fxml – The view preceding the main view, contains a 'Begin' button.

Controller:

CustomerFXMLController.java – The main business logic class of the program

FXMLDocumentController.java – Contains logic for displaying the first screen

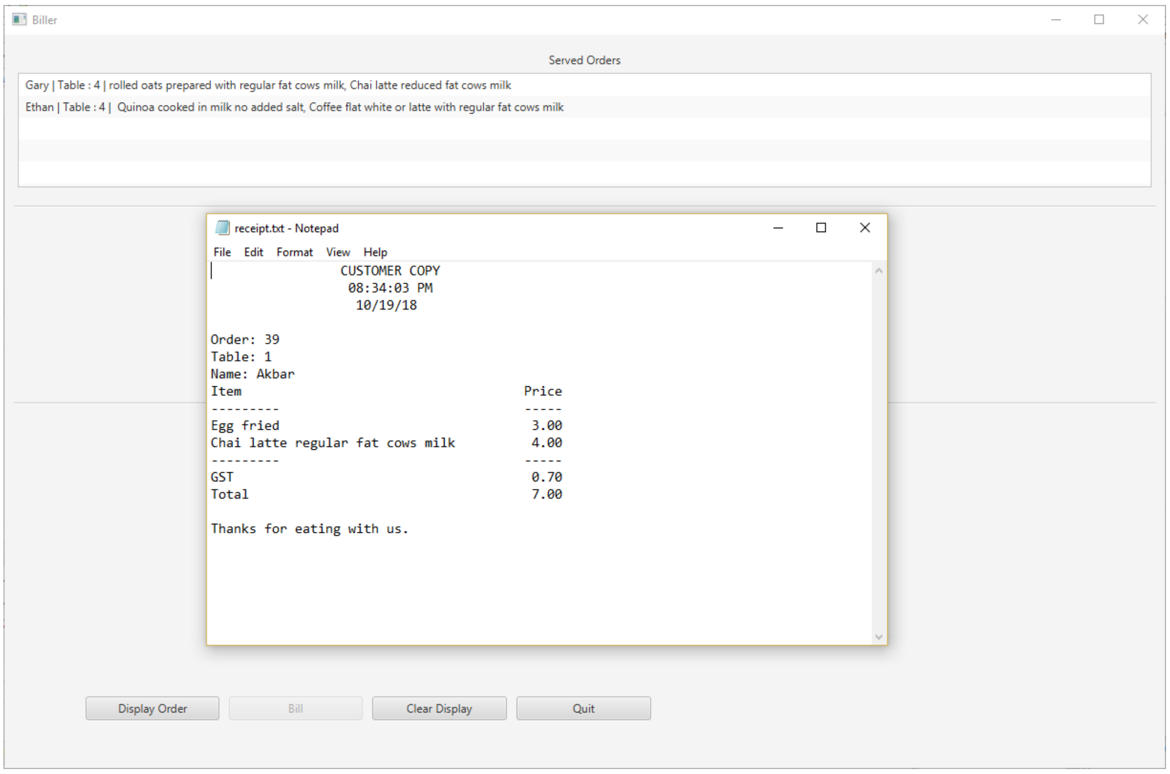
AdvancedJava_A2.java – Contains logic for launching the application.

Pre-Integration

The folder is simply called ‘preintegration’. It contains programs developed to test client-server networking.

Improvements (Additional Features)

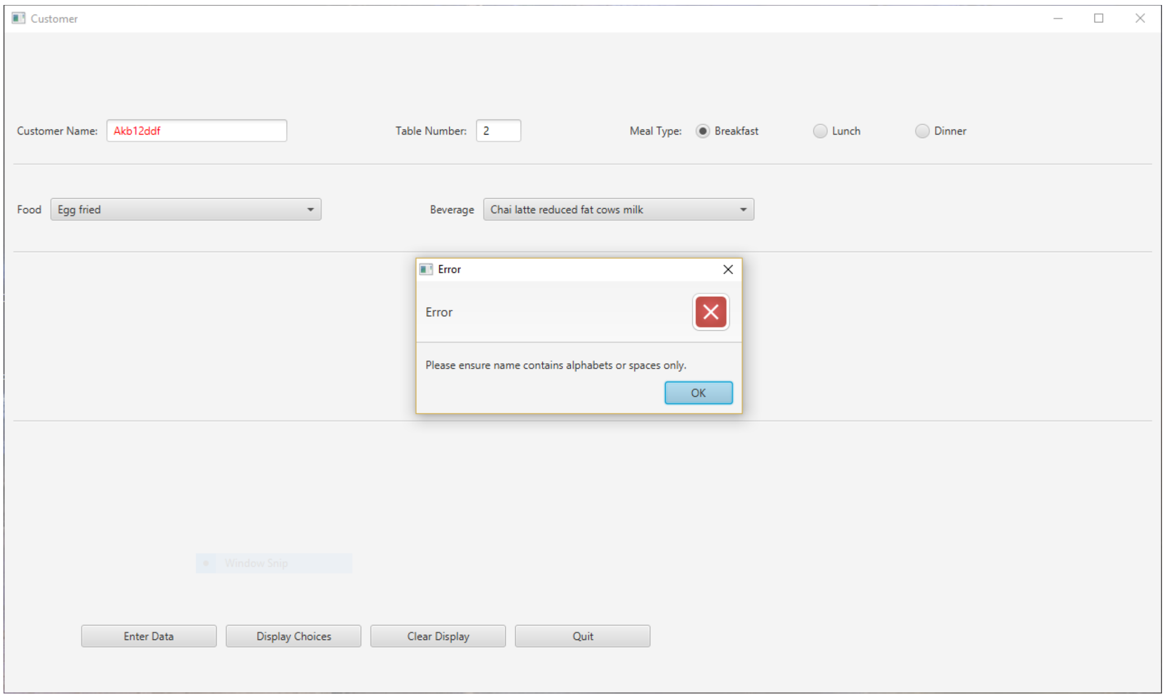
1. When an order is billed, the program opens up a text editor and displays the bill.



Gary | Table : 4 | rolled oats prepared with regular fat cows milk, Chai latte reduced fat cows milk
Ethan | Table : 4 | Quinoa cooked in milk no added salt, Coffee flat white or latte with regular fat cows milk

```
                 CUSTOMER COPY
                  08:34:03 PM
                   10/19/18

Order: 39
Table: 1
Name: Akbar
Item                                    Price
---------                               -----
Egg fried                                3.00
Chai latte regular fat cows milk         4.00
---------                               -----
GST                                      0.70
Total                                    7.00

Thanks for eating with us.
```

2. When entering data for placing an order, fields with invalid data turn red to improve usability. The customer name has turned red because it contains numbers.



Customer Name: Akb12ddf
Table Number: 2
Meal Type: [✓ Breakfast] Lunch Dinner
Food: Egg fried
Beverage: Chai latte reduced fat cows milk

Error

Please ensure name contains alphabets or spaces only.

Screenshots

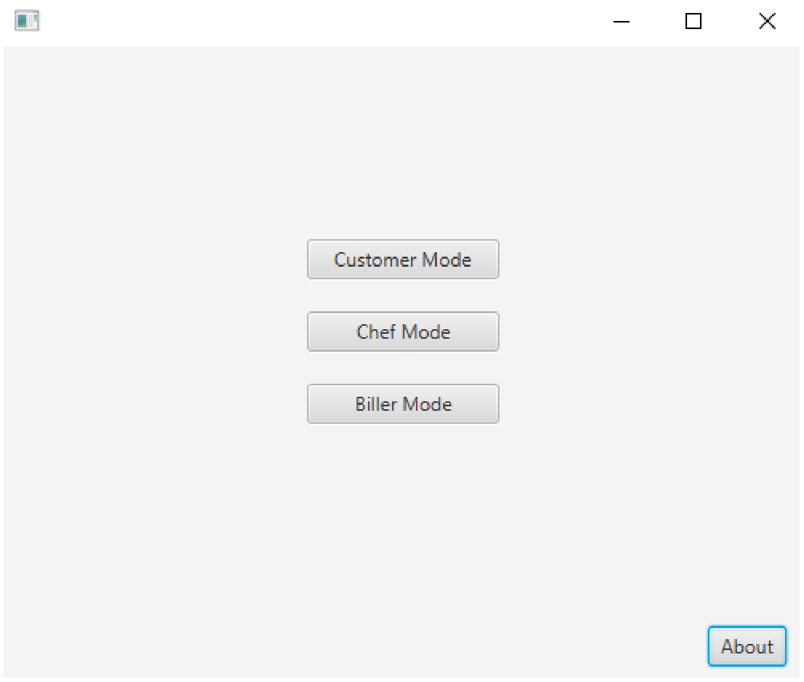
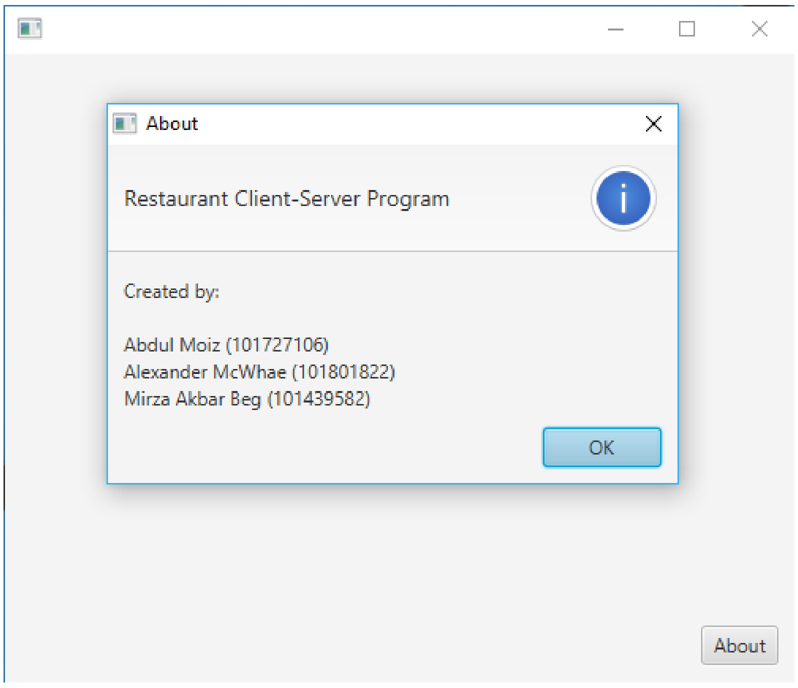


Figure 1 – Start-up Screen – Showing options for 3 different modes

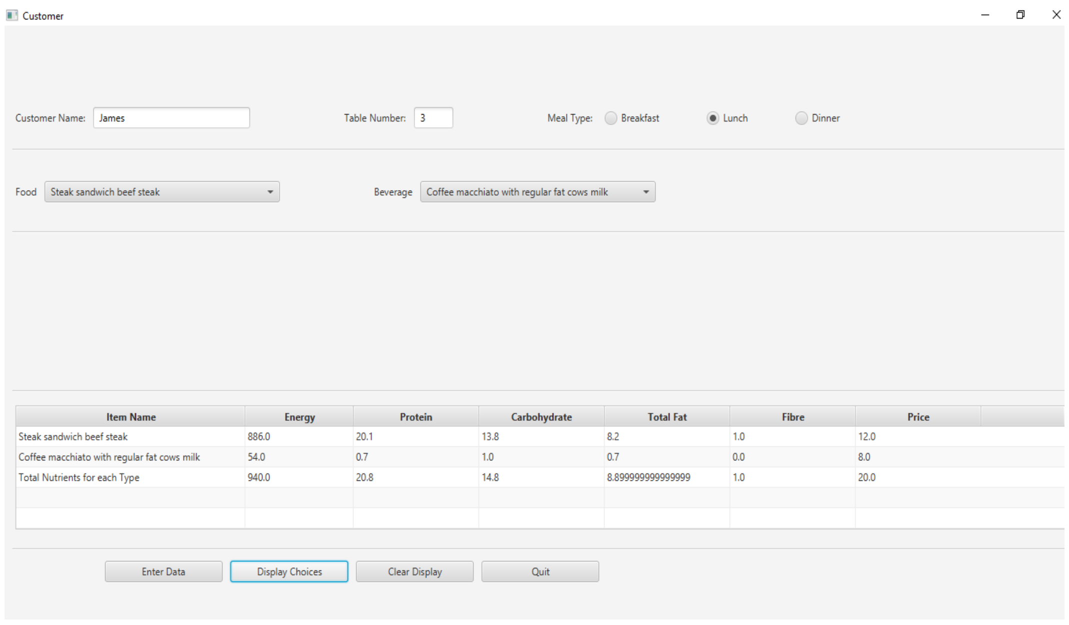


Restaurant Client-Server Program

Created by:

Abdul Moiz (101727106)
Alexander McWhae (101801822)
Mirza Akbar Beg (101439582)

Figure 2 – About Dialog from Start-up Screen

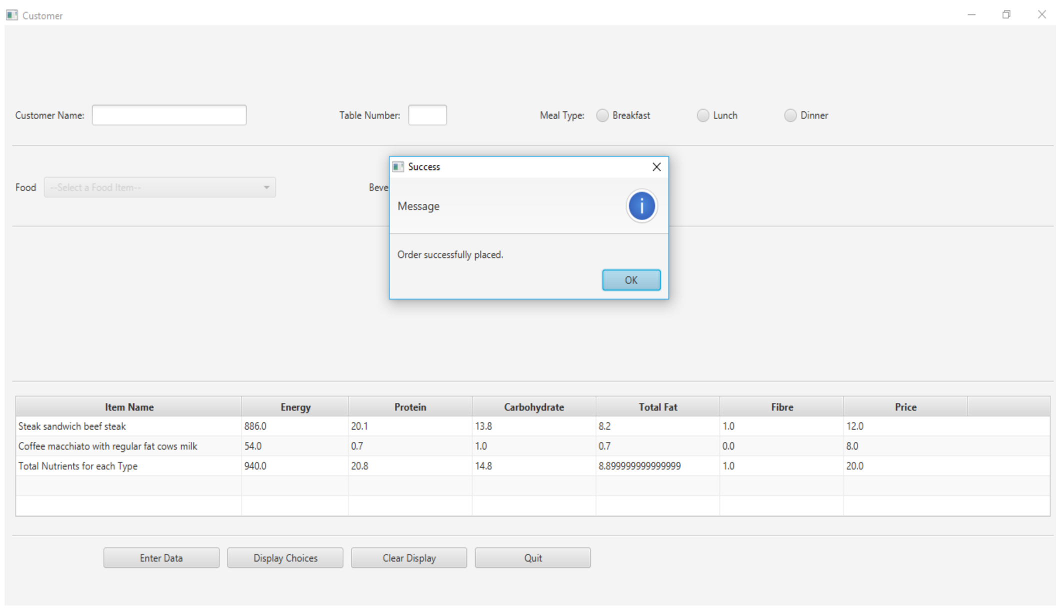


Customer Name: James   Table Number: 3   Meal Type: ( ) Breakfast [✓ Lunch] ( ) Dinner

Food: Steak sandwich beef steak   Beverage: Coffee macchiato with regular fat cows milk

Item Name	Energy	Protein	Carbohydrate	Total Fat	Fibre	Price
Steak sandwich beef steak	886.0	20.1	13.8	8.2	1.0	12.0
Coffee macchiato with regular fat cows milk	54.0	0.7	1.0	0.7	0.0	8.0
Total Nutrients for each Type	940.0	20.8	14.8	8.899999999999999	1.0	20.0

Figure 3 – Customer Mode: Customer Details and Order Selected – 'Display Choices' button displays the order in a table



Customer Name: Table Number: Meal Type: ( ) Breakfast ( ) Lunch ( ) Dinner

Food: --Select a Food Item--   Beve

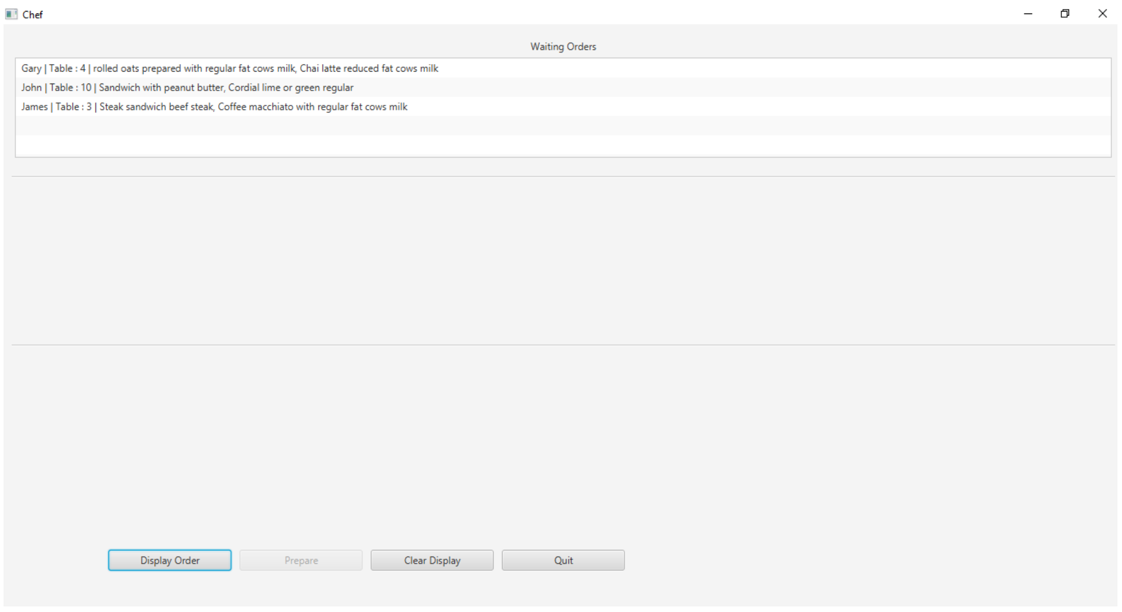
Success

Message

Order successfully placed.

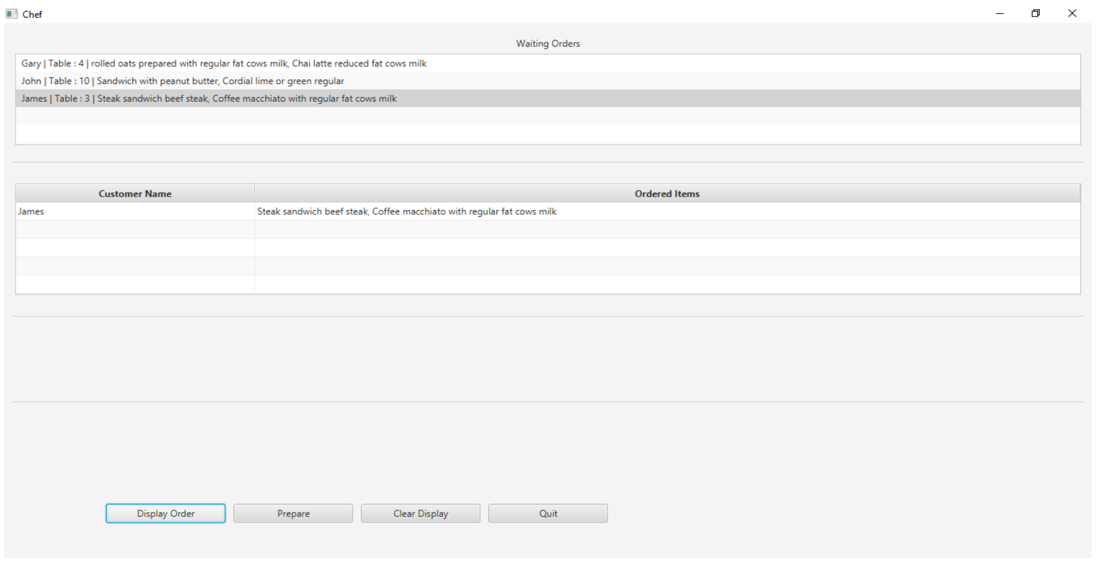
Item Name	Energy	Protein	Carbohydrate	Total Fat	Fibre	Price
Steak sandwich beef steak	886.0	20.1	13.8	8.2	1.0	12.0
Coffee macchiato with regular fat cows milk	54.0	0.7	1.0	0.7	0.0	8.0
Total Nutrients for each Type	940.0	20.8	14.8	8.899999999999999	1.0	20.0

Figure 4 – 'Enter Data' button places the order and shows confirmation dialog



Gary | Table : 4 | rolled oats prepared with regular fat cows milk, Chai latte reduced fat cows milk
John | Table : 10 | Sandwich with peanut butter, Cordial lime or green regular
James | Table : 3 | Steak sandwich beef steak, Coffee macchiato with regular fat cows milk

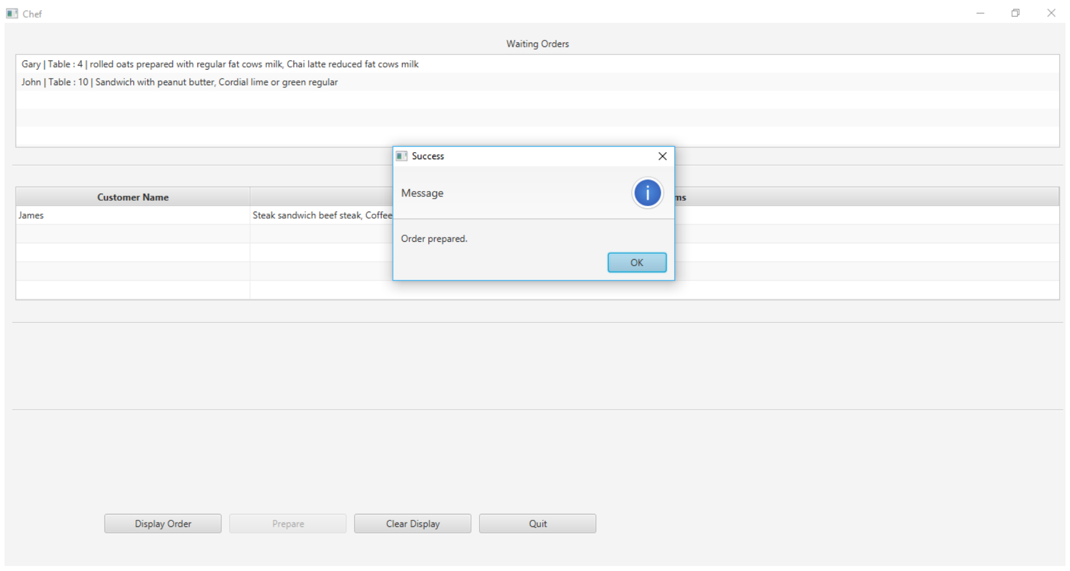
Figure 5 – The Chef application acts as the server and receives the order from Customer



Gary | Table : 4 | rolled oats prepared with regular fat cows milk, Chai latte reduced fat cows milk
John | Table : 10 | Sandwich with peanut butter, Cordial lime or green regular
James | Table : 3 | Steak sandwich beef steak, Coffee macchiato with regular fat cows milk

Customer Name	Ordered Items
James	Steak sandwich beef steak, Coffee macchiato with regular fat cows milk

Figure 6 – 'Display Order' button displays the selected order to the Chef



Waiting Orders

Gary | Table : 4 | rolled oats prepared with regular fat cows milk, Chai latte reduced fat cows milk
John | Table : 10 | Sandwich with peanut butter, Cordial lime or green regular

Customer Name	ms
James	Steak sandwich beef steak, Coffee

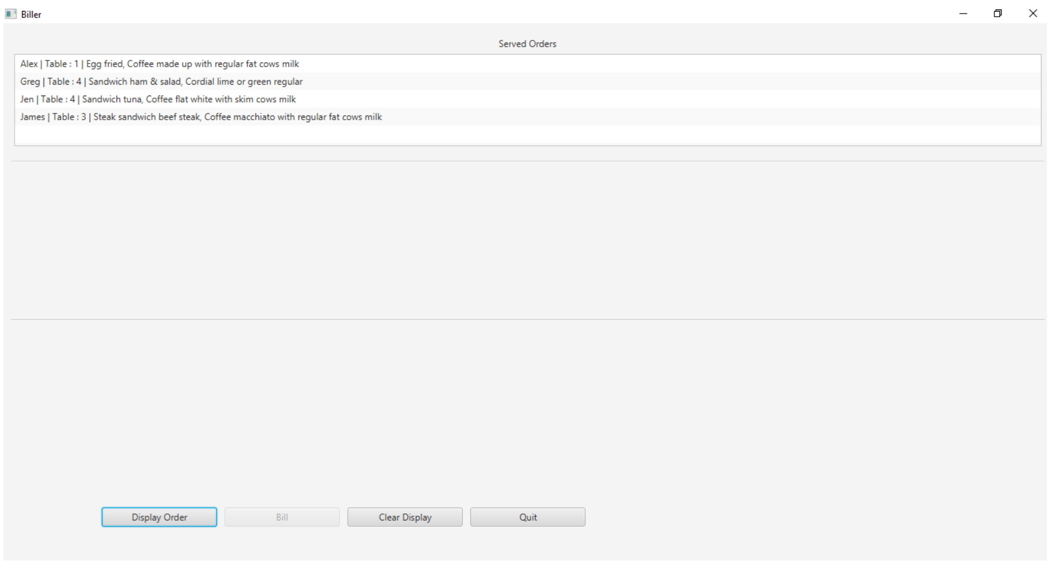
Success

Message

Order prepared.

OK

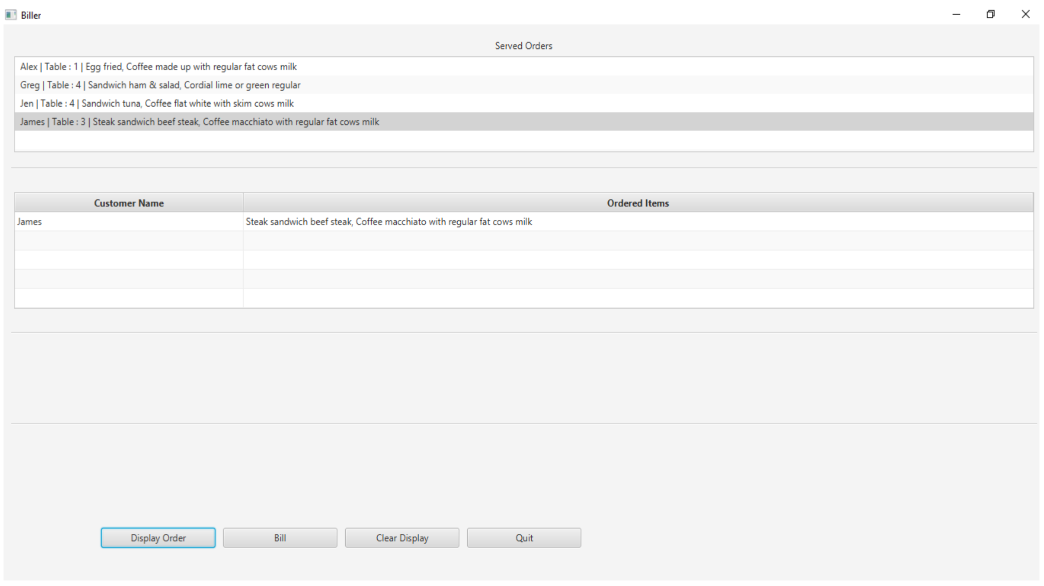
Figure 7 – 'Prepare' button prepares the selected order and it is removed from the Waiting Orders List



Served Orders

Alex | Table : 1 | Egg fried, Coffee made up with regular fat cows milk
Greg | Table : 4 | Sandwich ham & salad, Cordial lime or green regular
Jen | Table : 4 | Sandwich tuna, Coffee flat white with skim cows milk
James | Table : 3 | Steak sandwich beef steak, Coffee macchiato with regular fat cows milk

Figure 8 – The Biller application receives the order from the Chef

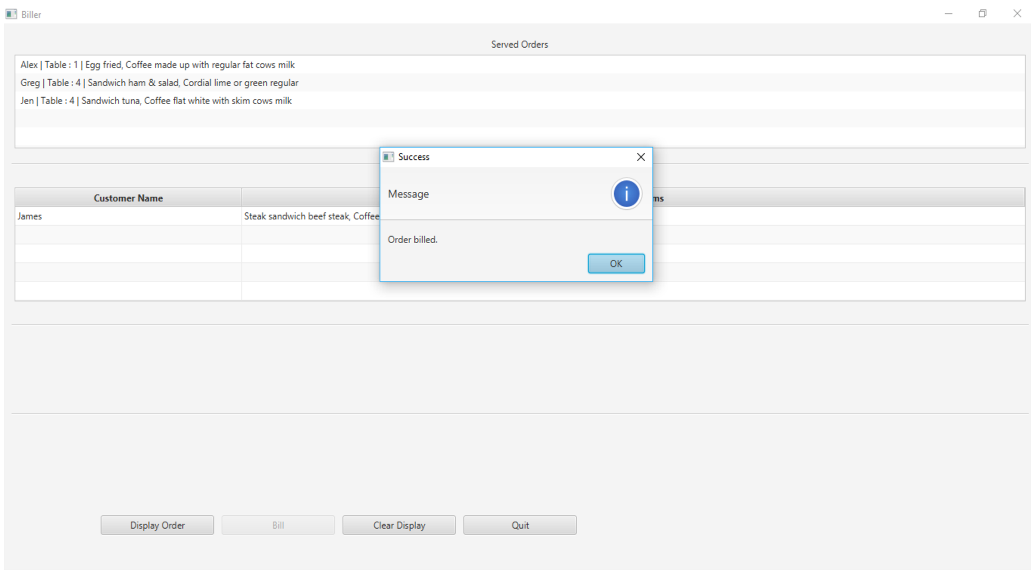


Served Orders

```
Alex | Table : 1 | Egg fried, Coffee made up with regular fat cows milk
Greg | Table : 4 | Sandwich ham & salad, Cordial lime or green regular
Jen | Table : 4 | Sandwich tuna, Coffee flat white with skim cows milk
James | Table : 3 | Steak sandwich beef steak, Coffee macchiato with regular fat cows milk
```

Customer Name	Ordered Items
James	Steak sandwich beef steak, Coffee macchiato with regular fat cows milk

Figure 9 – Clicking the 'Display Order' button displays the selected order for the Biller



Served Orders

```
Alex | Table : 1 | Egg fried, Coffee made up with regular fat cows milk
Greg | Table : 4 | Sandwich ham & salad, Cordial lime or green regular
Jen | Table : 4 | Sandwich tuna, Coffee flat white with skim cows milk
```

Message

Order billed.

Customer Name	
James	Steak sandwich beef steak, Coffee

Figure 10 – Clicking the 'Bill' button bills the order and displays confirmation message