NATIONAL INSTITUTE OF TECHNOLOGY
KARNATAKA, SURATHAKAL

DEPARTMENT OF COMPUTER ENGINEERING

Object Oriented Programming Project on

SMART MENU CARD SYSTEM

Submitted by: G. Aishwarya – 14CO213

B. Sandhya Rani – 14CO205

Submitted to: Ms. Sharvari Madam

SMART MENU CARD SYSTEM

Digitalizing Food Ordering System

ABSTRACT

Through this project we attempt to engineer a solution to a real world problem; elimination of manual undertaking of food orders in restaurants. The existing system of placing orders in restaurants in the pen and paper style is outdated, and is flawed. It is time consuming, and at times, unreliable. Our project aims to reach new heights in terms of customer satisfaction and overall experience of customers in restaurants.

The project is divided into two main parts, the Customer Side, and the Chef Side. On the customer side, the customer can browse through the variety of dishes offered by the restaurant via the Graphical User Interface provided by our Java Application and customer will be able to place his final order, once he has made a choice. The second module of the project is a Kitchen View or the Chef's View. As soon as the customer places his final order, he will be notified about his total bill amount, and his order will reach the computer in the kitchen. The Chef will be able to distinguish between the orders placed by various tables, and can review them, and will be able to clear off the pending order list.

The implementation of this project has been implemented entirely in Java, in NetBeans Java Development Environment. This project can be easily implemented in any restaurant with the required hardware support. This project provides a basic, yet efficient solution to eliminate the hassle of manual food order undertaking.

INDEX

1. INTRODUCTION

This project was conceived with a motive to eliminate the existent system of waiters manually taking down the food orders of customers in restaurants, food joints, and hotels. The existing system of placing orders in restaurants in the pen and paper style is outdated, and flawed. Hence, we aim to digitize food ordering in restaurants. This project is aimed at providing customers flexibility, convenience, as well as reliability.

The intended system must work as follows: The customer can browse through the variety of dishes offered by the restaurant via the Graphical User Interface provided by our Java Application which displays all the food items prepared by the restaurant. The Graphical User Interface will include the prices as well as a brief description about each food item. The customer will be able to place his final order, once he has made a choice. As soon as the customer places his final order, he will be notified about his total bill amount, and his order will reach the computer in the kitchen. The Chef will be able to distinguish between the orders placed by various tables, and can review them. After finishing the preparation of a certain customer's order, the Chef will be able to clear off the pending order list.

This project can be easily implemented in any restaurant with the required hardware support. We need a dedicated machine, or a mobile device, or an embedded system with the ability to run basic Java applications at every customer site, i.e. every table in the restaurant. We also require a main server that will be running at all times when the restaurant is working, in order to undertake food orders from the customers. This main server will be a dedicated Personal Computer, or a laptop that will be stationed in the kitchen. This server will receive all the orders placed by the customers remotely. We plan to implement this connectionless service by establishing sockets. One main point to note is that, the server, and all the devices at the customer sites must be connected within a network. The network can be a local area network, comprising of just the hardware devices within the restaurant. Note that the system will be fully functional even in the absence of Internet. The system requires a network, and not the Internet.

2. COMPONENTS OF THE SYSTEM

The project is essentially divided into two main parts, the Customer Side, and the Chef Side. In a more technical jargon, it can be viewed as the Client Side, and the Server Side. We develop our project with these two views in mind.

2.1. CUSTOMER VIEW

In order to design the customer side of the project, we attempt to gather the exact requirements from the customer's point of view.

The customer must be able browse through the plethora of delicacies offered by the restaurant. To provide for the convenience of the customer, these dishes must not be displayed as one long list, but must be divided into different categories of food stuffs, such as Salads, Starters, Main Course, Desserts, and the like. Another method of categorizing food would be based on the cuisine of the food. Any one of the above methods can be adopted in order to facilitate a better user experience. We follow the first approach of categorizing food.

The system needs to ensure navigability. This implies that once a customer selects a certain category of food, he/she must at all times be given the option of switching to another category without any hassle, by simply clicking a button.

Another main concern in the customer side is the display of food items in each category. The food items must be displayed in a neat manner with the price of every item enlisted. The customer must be given the option to select an item, in case he wants to order it. The customer must also have an option to change the quantity of an item as per his/her wishes.

Since the entire point of this system is to minimize human interaction during the placement of a food order, it is vital to explicitly provide a description about each item

prepared by the restaurant in order to give the customer an insight into the dish. This will aid the customer make better choices, and eliminate any ambiguity.

The system must provide the customer with a dynamically changing table wherein he/she can view the orders that he/she has placed, along with the quantity, and the total price for a certain item. This option will help the customer keep track of his order.

The system must also implement an option to “Update Order” as he/she navigates through the various categories of food. A button for “Place Final Order” must be created. Upon the placement of the final order, the system must generate the total bill amount, and notify the customer regarding the same. And, the most important part is that, the order placed by the customer must successfully reach the kitchen, where it can be processed.

Care must be taken that the developed Graphical User Interface (GUI) is aesthetically appealing, functional, as well as convenient at the same time to enhance the customer's experience, since the reputation of the restaurant will be at stake.

2.2. KITCHEN VIEW

The second module of the project is a Kitchen View or the Chef’s View. The main requirements of this module are as follows.

The GUI at the kitchen need not be aesthetically appealing. Functionality is of higher priority here. The GUI at the kitchen side must be divided, so as to accommodate orders from all the tables in the restaurant simultaneously.

The system must have a dedicated space in the GUI for each table where the particular table's food order will be populated.

It must be ensured that this table is dynamically changing. That is, if the same customer places two separate orders, the second order that he placed must not replace the contents

of the first order.

Every table in the GUI must be provided with a “Done” button, so that the Chef can clear off orders that have been prepared in the kitchen.

2.3. THE LINK

This is the most important part of the project. Despite identifying the exact requirements at the Customer Side, and the Kitchen Side, they would be rendered useless without a connection between them. The system will establish this link through Sockets.

In our system, the kitchen will act as the server, and the Customers will act as clients. They will connect via the socket that we establish between them. We will achieve this by socket programming. However, in socket programming, a server can connect to, and serve only one client at a time. This is a major drawback. As a solution to this problem, we adopt “Multithreading”.

Every time a customer places his final order, a socket is client socket is created, and a thread of the server is created. In this method, we have a dedicated server thread connected to each of our client sockets, eliminating the need to wait for the server to become idle.

Hence, Multithreading in collaboration with Socket programming serves as the basis of the connection between the kitchen and all the customers in the restaurant.

3. SPECIAL FEATURES

Our system inculcates certain features that sets it apart from other software of its kind.

- A dynamically changing view each time the customer selects a certain dish. In this manner, the customer can keep track of exactly what he has order, and make further choices accordingly.

- A description panel providing additional details about every food item that is offered by the restaurant. This is especially useful when the names of dishes have confusingly complicated names, a common problem that most customers encounter.

- A wireless connection between the customers, and the kitchen-side.

- A multi-threaded Server System in order to completely eliminate the waiting time on the Customer side.

4. IMPLEMENTATION

We preferred NetBeans for our project. In our project, we basically have two main interfaces one is on Customer view and the other is on Kitchen view. These two contain many GUI applications for look and feel as well as Socket Programming for the functionality of the code. Here are few code snippets of the codes on two sides.

4.1.1. CUSTOMER VIEW

The GUI of customer side is mainly divided into 5 panels namely categoryPanel, menuPanel, jScrollPane9, jPanel1and userPanel.
Firstly categoryPanel has a Grid layout and consists of several buttons that lead the customer to various interfaces that are explained later. The buttons include Salads, Appetizers, Main Course, Indian Breads, Rice Items, Quick Bites and Desserts.
The action that has to be performed by each button has its own action listener. Below is the sample code for Salads button.

```
categoryPanel.setLayout(new java.awt.GridLayout(0, 1, 5, 5));
    jButton1.setText("SALADS");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });
    categoryPanel.add(jButton1);
```

Next comes the most important panel i.e. menuPanel. It uses a Card layout which comprises of seven cards. Each card is nothing but a panel. On clicking the buttons added in categoryPanel, customer is directed to any of these cards. The cards are namely, SaladPanel, breadPanel, quickPanel, ricePanel, dessertPanel, AppPanel and MainCoursePanel.

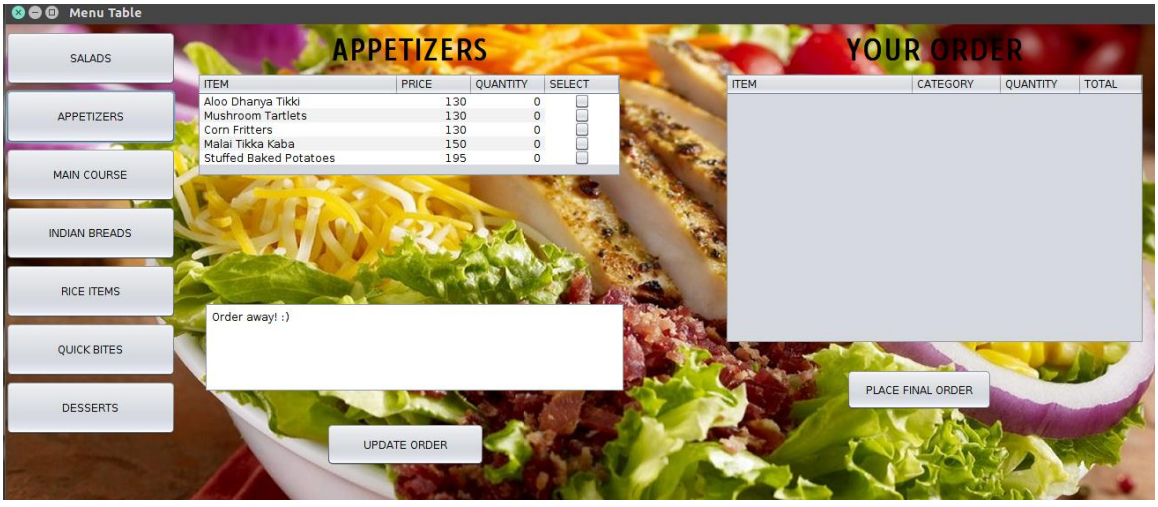
Each of these panels contains a jtable that has a list of names of related items, corresponding prices, a column for quantity to be ordered and a checkbox to select. The criterion is that when the customer wants to select a quantity, he/she must first check the box and then give a valid quantity (1 to as many as one wants).
The panel also consists of Label which is the name of the Panel in which the customer currently is. It can be Salads, Appetizers, and Desserts etc.
Below is the code that shows how to add a card and move to that card on button click.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        CardLayout showPanel = (CardLayout) menuPanel.getLayout();
        showPanel.show(menuPanel, "saladCard");
        currentCard = "saladCard";
    }
```

Below is the code that shows how menuPanel and one of the cards, here SaladCard are created along with functionalities.

```
menuPanel.setOpaque(false);
        menuPanel.setPreferredSize(new java.awt.Dimension(640, 399));
        menuPanel.setLayout(new java.awt.CardLayout());
        SaladPanel.setOpaque(false);
        saladTable.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {"Garden Green Salad",  new Integer(120),  new Integer(0),  new
Boolean(false)},
                {"Fruit Cabiana",  new Integer(150),  new Integer(0),  new Boolean(false)},
                {"Hawaiian Salad",  new Integer(150),  new Integer(0),  new Boolean(false)},
                {"Curried Baby  Potato Salad",  new Integer(150),  new Integer(0),  new
Boolean(false)},
                {"Greek Salad",  new Integer(195),  new Integer(0),  new Boolean(false)} },
            new String [] {
```

```
            "ITEM", "PRICE", "QUANTITY", "SELECT"}
        ) {
            Class[] types = new Class [] {
                java.lang.String.class, java.lang.Integer.class, java.lang.Integer.class,
java.lang.Boolean.class
            };
            boolean[] canEdit = new boolean [] {
                false, true, true, true
            };
        saladTable.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                saladTableMouseClicked(evt);
            } });
        jScrollPane1.setViewportView(saladTable);
        if (saladTable.getColumnModel().getColumnCount() > 0) {
            saladTable.getColumnModel().getColumn(0).setResizable(false);
            saladTable.getColumnModel().getColumn(0).setPreferredWidth(160);
            ..... for other columns. }
        jLabel1.setFont(new java.awt.Font("Ubuntu Condensed", 1, 36)); // NOI18N
        jLabel1.setText("SALADS");
        menuPanel.add(SaladPanel, "saladCard");
```



The rest of the panels follow the same way.

Next comes the userPanel which has just one button with text **Update Order**. The functionality of this button is that on clicking the button, whichever items have been checked and given valid quantity as explained previously, get added to another table beside which will be explained later.

The unique feature of this and the previous panel is that if the customer wants to uncheck an item and select a new item or change the quantity of a selected item, he/she can just change them in the table and then click on the **Update Order** button and can see the updated items or values in the beside table.
The action listener of this button is as follows.

```
private void updateOrderButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String category = "";
    javax.swing.JTable table = null;
    switch (currentCard) {
       case "saladCard":
          table = saladTable;
          category = "SALAD";
          break;
      ……many more cases.
    }
    DefaultTableModel dtm = (DefaultTableModel) table.getModel();
    DefaultTableModel dtm2 = (DefaultTableModel) finalOrderTable.getModel();
    int nRows = dtm.getRowCount();
    int rowCount = dtm2.getRowCount();
    int j = rowCount, k;
    for (k = rowCount - 1; k >= 0; k--) {
       String compare = (String) dtm2.getValueAt(k, 1);
       if (compare.equals(category)) {
```

```
            dtm2.removeRow(k);
            j--;   }  }
        for (int i = 0; i < nRows; i++) {
           if (((boolean) dtm.getValueAt(i, 3)) && !(((Integer) dtm.getValueAt(i,
2)).equals(0))) {
            String s = (String) dtm.getValueAt(i, 0);
            int quantity = (int) dtm.getValueAt(i, 2);
            int total = (int) dtm.getValueAt(i, 1) * quantity;
            dtm2.addRow(new Object[1]);
            dtm2.setValueAt(s, j, 0);
            dtm2.setValueAt(quantity, j, 2);
            dtm2.setValueAt(category, j, 1);
            dtm2.setValueAt(total, j, 3);
            j++;
          }   }  }
```

Another panel which is basically a ScrollPane consists of a TextArea. Whenever the customer just clicks on a particular item (no need of selecting), then a small description of that item appears in that text area.
This functionality is added to every table separately. Here is the code snippet for quickBites table.

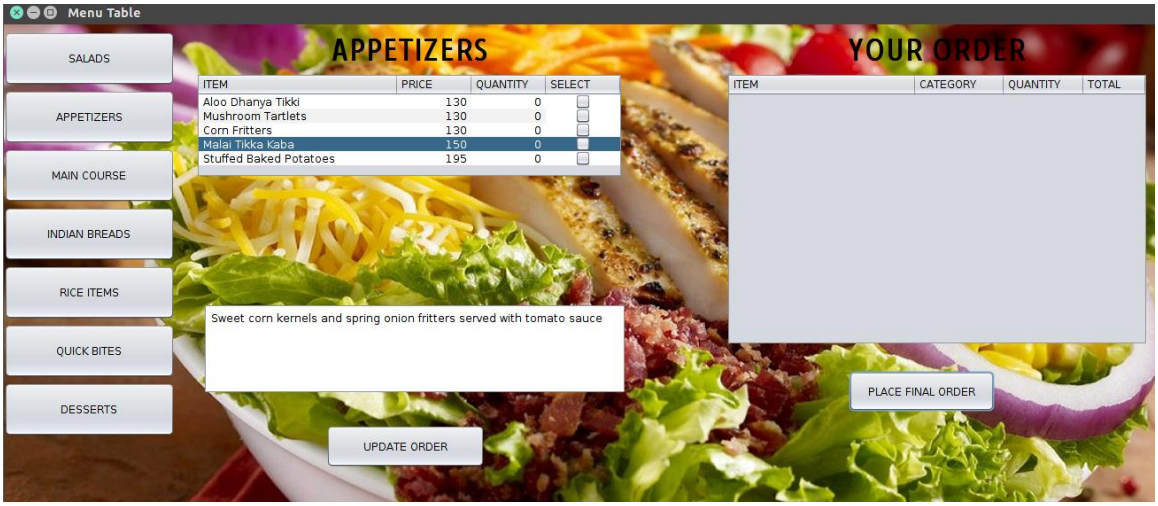
```
private void quickBitesTableMouseClicked(java.awt.event.MouseEvent evt) {
        int row;
        if ((row = quickBitesTable.getSelectedRow()) >= 0) {
           String item = (String) ((DefaultTableModel)
quickBitesTable.getModel()).getValueAt(row, 0);
           itemInfoTextArea.setText((String) itemInfo.get(item));
        } else {
           itemInfoTextArea.setText("Order away! :)");
        }    }
```

We used a HashMap to get the description of the corresponding item like below.

itemInfo = new HashMap<>();

itemInfo.put("Garden Green Salad",

"Fresh garden vegetables dressed with herb vinaigrette");



Next comes the jPanel1. This panel doesn't follow any layout. It has a single jtable and button to **Place Final Order**. The items which have been checked in the above discussed cards of the menuPanel and updated by button click are added to this table. This table gives list of items chosen by the customer, the quantity and price of whole quantity of the respective item. This also has the category of the item to which the item belongs.
On clicking the button **Place Final Order**, the order is meant to be confirmed and then a socket opens and sends the table number and contents of this table to the Kitchen (chef) through that socket. The Kitchen (nothing but the Server) will always be running at certain port and the customer gets connected to that port using a socket.The socket will be open till the order is served. The customer is free to make a second order or so after this.

DefaultTableModel dtm1 = (DefaultTableModel) finalOrderTable.getModel();

try {

Socket sock = new Socket("127.0.0.1", 9090);

DataOutputStream dos = new DataOutputStream(sock.getOutputStream());

```
            dos.writeInt(table_number);
            int rowCount = dtm1.getRowCount();
            dos.writeInt(rowCount);
            for(int i=0; i<rowCount; i++){
               String item = (String)dtm1.getValueAt(i, 0);
               int quantity = (int) dtm1.getValueAt(i, 2);
               dos.writeUTF(item);
               dos.writeInt(quantity);
            }        } catch (IOException ex)
{           Logger.getLogger(BaseFrame.class.getName()).log(Level.SEVERE, null, ex);
            }
           int total = 0;
           int rowCount = dtm1.getRowCount();
           for(int k=0;k<rowCount;k++)
             total+= (Integer)dtm1.getValueAt(k,3);
           JOptionPane.showMessageDialog(null,"Your order has been placed successfully!
\n Your total bill is: Rs."+total );
```

The above code also contains details about the total price and the order placed message in a dialogPane. On closing the dialogBox, all the checked boxes become unchecked and all the quantities are reset to zero (default).

4.2. KITCHEN VIEW

The GUI of Kitchen view contains four similar panels. Each panel represents customer table. It consists of a jTable listing the items and quantities sent by the customer through socket. When the customer actually gets connected to the same port on which server chef is running, then the chef accepts the connection and starts receiving data as follows.

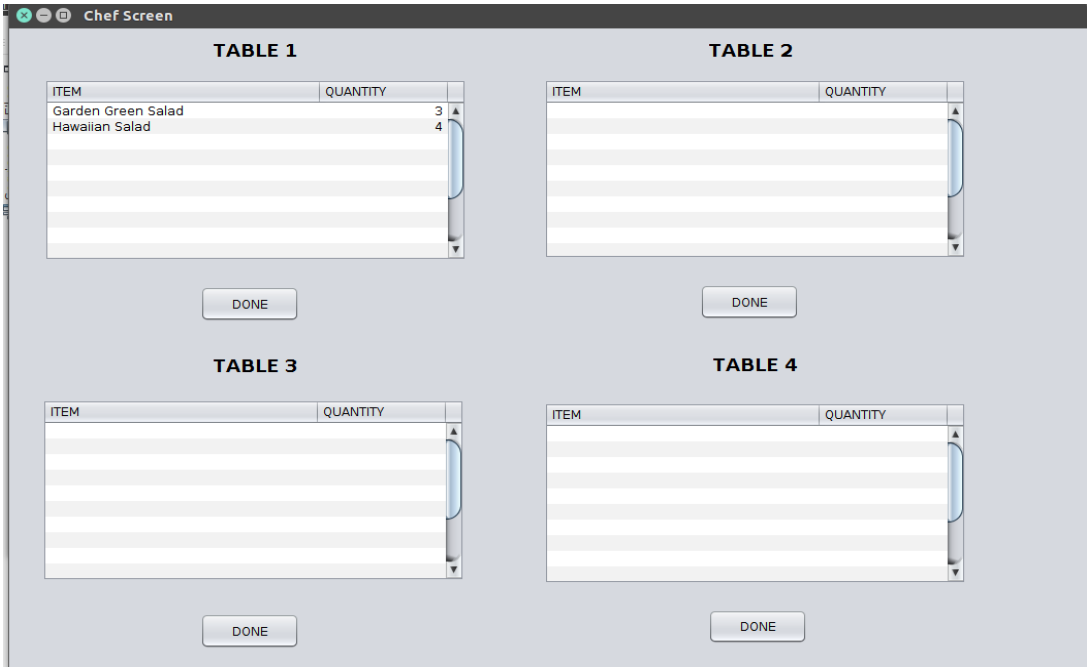
Accepting the Connection

```
while(true) {
        client = server.accept();
        System.out.println("Client at " + client.getInetAddress().getHostAddress() + "
connected.");
        dis = new DataInputStream(client.getInputStream());
        //read the table number
        tableNumber = dis.readInt();
        System.out.println("Table number: " + tableNumber);
        table = chefFrame.getTable(tableNumber);
        new ServerThread(threadCount++, client, table).start();
    }
```

Receiving Data

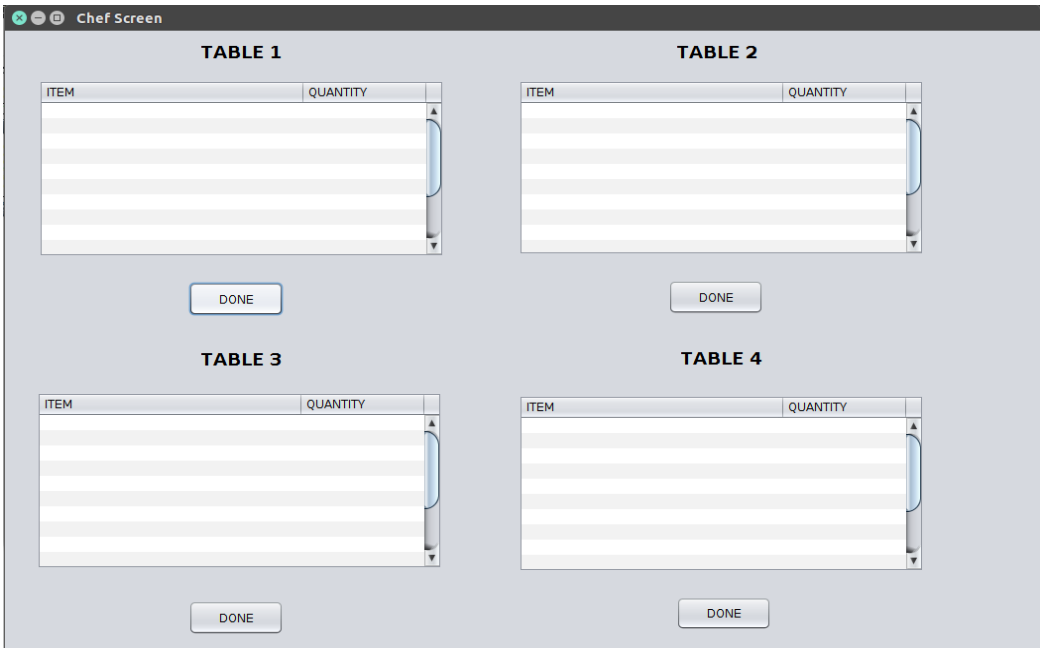
```
public void run() {
    try {
      DefaultTableModel Chefdtm = (DefaultTableModel) table.getModel();
      Integer noOfRows;
      noOfRows = dis.readInt();
      System.out.println("Number of rows at table: " + noOfRows);
      for(int i=0; i<noOfRows; i++) {
        String item = dis.readUTF();
```

```
            System.out.println("Item in row " + i + ": " + item);
            Chefdtm.setValueAt(item, i, 0);
            int quantity = dis.readInt();
            System.out.println("Quantity in row " + i + ": " + quantity);
            Chefdtm.setValueAt(quantity, i, 1);
        }       } catch (IOException ex) {
        Logger.getLogger(ServerThread.class.getName()).log(Level.SEVERE, null, ex);
    }   }
```



The panel also contains a label showing the table number and a button to clear the contents of the table once the order is finished.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        DefaultTableModel table1 = (DefaultTableModel) jTable1.getModel();
        for (int i = 0; i < table1.getRowCount(); i++) {
            for (int j = 0; j < table1.getColumnCount(); j++) {
                table1.setValueAt("", i, j);
            }   }  }
```


Chef Screen
TABLE 1
ITEM
QUANTITY
DONE
TABLE 2
ITEM
QUANTITY
DONE
TABLE 3
ITEM
QUANTITY
DONE
TABLE 4
ITEM
QUANTITY
DONE

5. EXTENSIONS

This project currently only digitalizes the food ordering in a restaurant, but, we can extend this digitalization to several other aspects within a restaurant.

The process of billing can be digitalized. As soon as a customer places a final order, the generated bill amount can be sent to the main billing desk via a socket. If a customer places more than one orders, the application at the main billing desk can perform simple calculations to calculate the total bill amount and produce a bill. This can be done without any human interference.

We can also extend this application to collect feedback from the customers as its invaluable information for the restaurant. Questions pertaining to the taste of the food, ambiance, and overall experience can be asked, and the results can be used to improve the quality of the restaurant.

A dynamically changing Menu can also be implemented that will reflect seasonal foods, or special items for the day.

6. AREAS OF LEARNING

This project has helped us to explore two completely new areas, Socket Programming and Multi-Threading. Though it seemed slightly complex in the beginning, with enough research, and persistence we made steady progress in these topics. After this project, we find ourselves more comfortable with the entire concept of sockets, and connectionless services. We have begun to appreciate multi-threading and its uses.

We have also learned how to develop Graphical User Interfaces. In today's world GUIs are of utmost importance since all users look for ease of operation. We have learned how make use of Java Development Environments to build aesthetically striking GUIs which are both functional, and pleasing to the eye.

The project also strengthened all the object oriented concepts that we have been taught during the course of this semester. We used the concepts of inheritance, interfaces widely throughout the project which helped us to understand them better and appreciate them.

The entire experience of developing an entire Java Application for beginners like us was both challenging, and exciting at the same time. This project definitely benefited us, and the learning curve was commendable!

7. CONCLUSION

Through this project we attempted to engineer a solution to a real world problem; elimination of manual undertaking of food orders in restaurants. The development of this project tackles this basic problem, though it can be extended to inculcate several other options. In fact, this project is easily deployable provided the required hardware is present.

8. REFERENCES

1. http://docs.oracle.com/javase/tutorial/networking/sockets/
2. https://www.youtube.com/watch?v=3CC9PtzTCVE
3. http://www.java-forums.org/netbeans/49274-how-insert-data-into-jtable-netbeans.html
4. https://www.youtube.com/watch?v=Y-M8kEKn0Rs
5. http://stackoverflow.com/questions/15619682/uncheck-checkboxes-in-java
6. https://www.youtube.com/watch?v=uZFgiqM0udA
7. http://stackoverflow.com/questions/11609900/how-to-make-the-background-of-a-jtable-transparent
8. http://stackoverflow.com/questions/15332585/set-value-of-jtable-column-using-setvalueat
9. http://stackoverflow.com/questions/18268663/get-cell-values-of-jtable