Pimpri Chinchwad Education Trust's

Pimpri Chinchwad College of Engineering

Department of Computer Engineering

Mini Project 1 Report

On

Invoice Generation System: Testing using Junit

Group Members

BECOC350 : Nikumbh Abhay Suresh
BECOC328 : Ingale Mehul Nilesh

Guide
Mr. Shrikant Kokate

Index

Introduction

The project is to create an application that should provide service to the user, collect user's orders, and generate invoices of each transaction, each billing cycle depends on the billing type, collect payments and adjust balances.
The Invoice Generation system has the capacity to illustrate the basic billing system and the main functionalities that surround the billing system from a business perspective and generate an Invoice.
Also, development of a system emulator that is capable of functioning more quickly, accurately and updates the records and enables customers to view bill information.
This Invoice system can be deployed in a real world situation. For example, it could be implemented for a medium scaled departmental store or even to a small vendor who needs to keep track of his/ her transactions

Functional Requirements

1. Only validated logins will be allowed to proceed to the next page.
2. Payment for the meals is be recorded.
3. The food availability is displayed.
4. Food revenue for a specified period of time (days) is displayed.
5. Addition of items to the meal is allowed.
6. An invoice is generated at the end of the order.

Nonfunctional Requirements

Non - Functional requirements define the needs in terms of performance, logical requirements, design constraints, standards compliance, reliability, availability, security, maintainability, and portability.

1.1.1 Performance Requirements

Performance requirements define acceptable response times for system functionality.

- The load time for user interface screens shall take no longer than a few seconds.
- The log in information shall be verified in a short time.

1.1.2 Design Constraints

This Invoice Generation System shall be a stand-alone system running in a Windows environment. The system shall be developed using Java .

1.1.3 Standards Compliance

There shall be consistency in variable names within the system. The graphical user interface shall have a consistent look and feel.

1.1.4 Reliability

Specify the factors required to establish the required reliability of the software system at time of delivery.

1.1.5 Availability

The system shall be available during normal operating hours.

1.1.6 Security

The Customer Service Representatives and Managers will be able to log in to the Invoice Generation System.

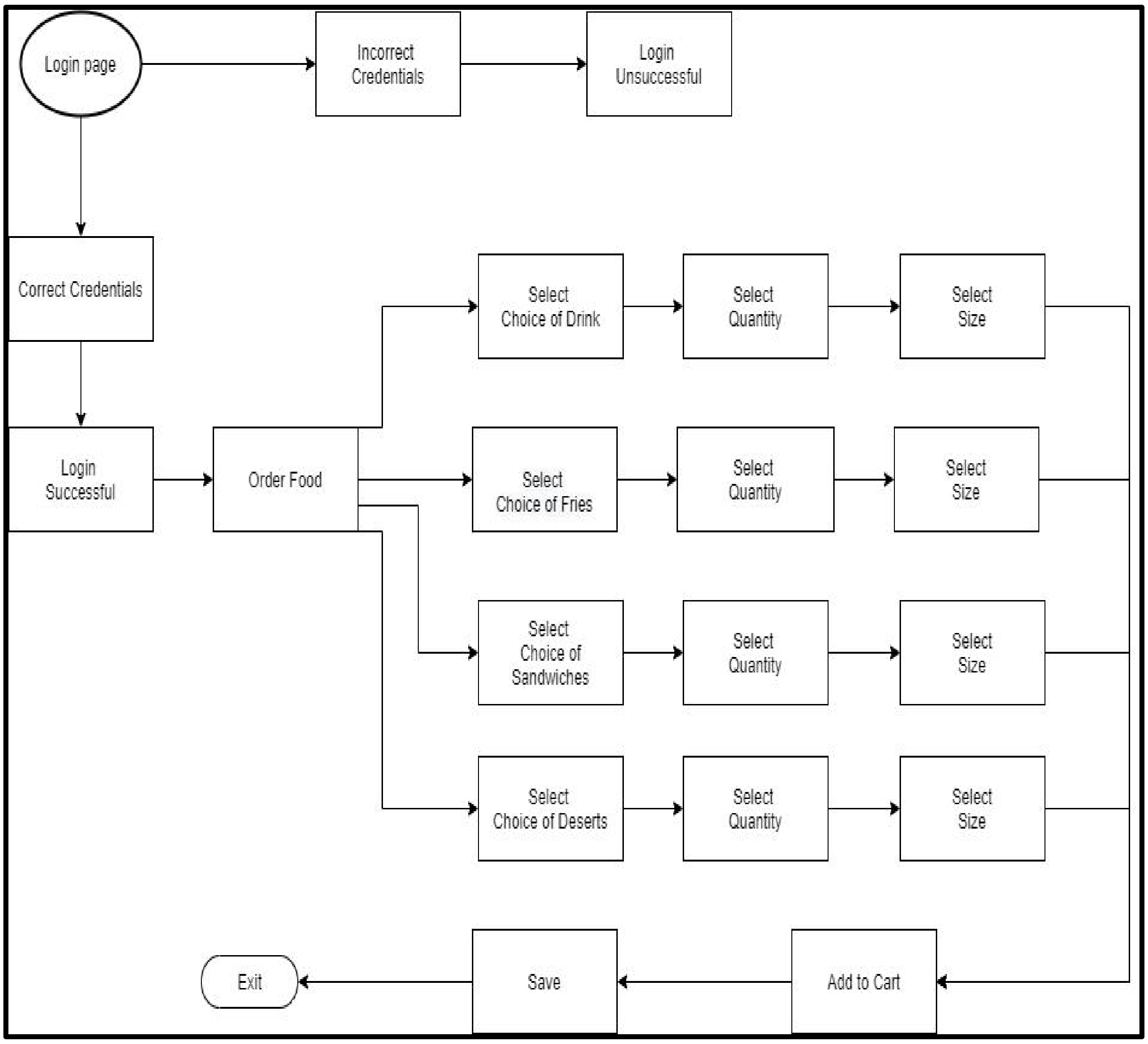
1.1.7 Maintainability

The Invoice Generation System is being developed in Java. Java is an object oriented programming language and shall be easy to maintain.

1.1.8 Portability

This Invoice Generation System shall run in any Microsoft Windows environment that contains Java Runtime

Block Diagram



SOURCE CODE

Login Page :-

```java
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import java.awt.Font;
import javax.swing.SwingConstants;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JPasswordField;
import javax.swing.JCheckBox;

public class Login {

    private JFrame frame;
    private JTextField userTextField;
    private JPasswordField passwordField;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Login window = new Login();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public Login() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 817, 530);
        frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JLabel userLabel = new JLabel("Username");
        userLabel.setFont(new Font("Tahoma", Font.PLAIN, 25));
        userLabel.setBounds(126, 175, 136, 55);
        frame.getContentPane().add(userLabel);

        JLabel passwordLabel = new JLabel("Password");
        passwordLabel.setFont(new Font("Tahoma", Font.PLAIN, 25));
        passwordLabel.setBounds(126, 252, 136, 55);
        frame.getContentPane().add(passwordLabel);

        userTextField = new JTextField();
        userTextField.setBounds(347, 185, 252, 38);
        frame.getContentPane().add(userTextField);
        userTextField.setColumns(10);

        JButton loginButton = new JButton("Login");
        loginButton.setFont(new Font("Tahoma", Font.PLAIN, 25));

        loginButton.setBounds(247, 379, 203, 43);
        frame.getContentPane().add(loginButton);

        passwordField = new JPasswordField();
        passwordField.setBounds(347, 252, 252, 38);
        frame.getContentPane().add(passwordField);

        loginButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
            if (e.getSource() == loginButton) {
                    String userText;
                    String pwdText;
                    userText = userTextField.getText();
                    pwdText = passwordField.getText();
        if (userText.equalsIgnoreCase("mehul") &&
pwdText.equalsIgnoreCase("123")) {
                JOptionPane.showMessageDialog(null, "Login
Successful");
                        frame.dispose();
                        Main_Page MP = new Main_Page();
                        MP.setVisible(true);
                    }
else if (userText.equalsIgnoreCase("abhay") &&
pwdText.equalsIgnoreCase("123")) {
                    JOptionPane.showMessageDialog(null, "Login
Successful");
•                           frame.dispose();
•                           Main_Page MP = new Main_Page();
•                           MP.setVisible(true);
•                       }
•                       else
•  {
•               JOptionPane.showMessageDialog(null, "Invalid Username or
  Password");
•                       }
•                   }
•               }
•           });
•       }
•   }
```

Main Order Page

```
•   import java.awt.BorderLayout;
•   import java.awt.EventQueue;
•   import java.awt.event.ActionEvent;
•   import java.awt.event.ActionListener;
•   import java.io.BufferedWriter;
•   import java.io.FileWriter;
•   import java.io.IOException;
•   import java.text.DecimalFormat;
•   import java.util.ArrayList;
•
•   import javax.swing.ButtonGroup;
•   import javax.swing.JButton;
•   import javax.swing.JComboBox;
•   import javax.swing.JFrame;
•   import javax.swing.JLabel;
•   import javax.swing.JOptionPane;
•   import javax.swing.JPanel;
•   import javax.swing.JRadioButton;
•   import javax.swing.JSeparator;
•   import javax.swing.JTextField;
•   import javax.swing.SwingConstants;
•   import javax.swing.border.EmptyBorder;
•
•   public class CHECK extends JFrame implements ActionListener{
•
•
•       private static final long serialVersionUID = 1L;
•       private JFrame frame;
•       static JTextField drinkquan;
•       static JTextField friesquan;
•       static JTextField swquan;
•       static JTextField dessertquan;
•       static JTextField total;
•       static JTextField tax;
•       static JTextField donate;
•       static JTextField amount;
•       private JRadioButton rdbtnTea, rdbtnCoffee, rdbtnJuice,
   rdbtnColddrink, rdbtnSimple, rdbtnPeriperi, rdbtnMasalaCheese,
   rdbtnMexicanLoaded, rdbtnAlooTikki, rdbtnBombayMasala,
   rdbtnTangyPaneer, rdbtnVegloaded,rdbtnChococheeseCake,
   rdbtnRedvelvetCake, rdbtnBlueberryMuffin, rdbtnApplepie;
•       private JComboBox<String> drinkssize, dessertsize, swsize,
   friessize;
•       private JButton  btnAddToOrder;
•       static JButton btnMakeBill;
•       private JButton btnClear;
•       private JButton btnSave;
•       int amount_of;
•       ArrayList<Beverage> list_of_bvr = new ArrayList<Beverage>();
•       static ButtonGroup Drinksgroup;
•       static ButtonGroup Dessertgroup;
•       static ButtonGroup SWgroup;
•       static ButtonGroup Friesgroup;
•       String SaveReport;
•       private JPanel contentPane;
•
•       /**
•        * Launch the application.
•        */
•       public static void main(String[] args) {
•           EventQueue.invokeLater(new Runnable() {
•               public void run() {
•                   try {
•                       CHECK frame = new CHECK();
•                       frame.setVisible(true);
•                   } catch (Exception e) {
•                       e.printStackTrace();
•                   }
•               }
•           });
•       }
•
•       /**
•        * Create the frame.
•        */
•       public CHECK() {
•           setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
•           setBounds(100, 100, 710, 444);
•           contentPane = new JPanel();
•           contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
•           contentPane.setLayout(new BorderLayout(0, 0));
•           setContentPane(contentPane);
```

```
            initialize();
        }

        void initialize() {
            //Radio Buttons
            rdbtnTea = new JRadioButton("Tea");
            rdbtnTea.setBounds(7, 66, 72, 23);
            contentPane.add(rdbtnTea);

            rdbtnCoffee = new JRadioButton("Coffee");
            rdbtnCoffee.setBounds(7, 92, 72, 23);
            contentPane.add(rdbtnCoffee);

            rdbtnJuice = new JRadioButton("Juice");
            rdbtnJuice.setBounds(7, 118, 72, 23);
            contentPane.add(rdbtnJuice);

            rdbtnColddrink = new JRadioButton("Cold-Drink");
            rdbtnColddrink.setBounds(7, 144, 87, 23);
            contentPane.add(rdbtnColddrink);

            Drinksgroup = new ButtonGroup();
            Drinksgroup.add(rdbtnTea);
            Drinksgroup.add(rdbtnCoffee);
            Drinksgroup.add(rdbtnJuice);
            Drinksgroup.add(rdbtnColddrink);

            //////////////////////////////////////////////

            rdbtnSimple = new JRadioButton("Simple");
            rdbtnSimple.setBounds(96, 66, 109, 23);
            contentPane.add(rdbtnSimple);

            rdbtnPeriperi = new JRadioButton("Peri-Peri");
            rdbtnPeriperi.setBounds(96, 92, 109, 23);
            contentPane.add(rdbtnPeriperi);

            rdbtnMasalaCheese = new JRadioButton("Masala Cheese");
            rdbtnMasalaCheese.setBounds(96, 118, 109, 23);
            contentPane.add(rdbtnMasalaCheese);

            rdbtnMexicanLoaded = new JRadioButton("Mexican Loaded");
            rdbtnMexicanLoaded.setBounds(96, 144, 109, 23);
            contentPane.add(rdbtnMexicanLoaded);

            Friesgroup = new ButtonGroup();
            Friesgroup.add(rdbtnSimple);
            Friesgroup.add(rdbtnPeriperi);
            Friesgroup.add(rdbtnMasalaCheese);
            Friesgroup.add(rdbtnMexicanLoaded);

            //////////////////////////////////////////////////

            rdbtnAlooTikki = new JRadioButton("Aloo Tikki");
            rdbtnAlooTikki.setBounds(202, 66, 109, 23);
            contentPane.add(rdbtnAlooTikki);

            rdbtnBombayMasala = new JRadioButton("Bombay Masala");
            rdbtnBombayMasala.setBounds(202, 92, 109, 23);
            contentPane.add(rdbtnBombayMasala);

            rdbtnTangyPaneer = new JRadioButton("Tangy Paneer");
            rdbtnTangyPaneer.setBounds(202, 118, 109, 23);
            contentPane.add(rdbtnTangyPaneer);

            rdbtnVegloaded = new JRadioButton("Veg-Loaded");
            rdbtnVegloaded.setBounds(202, 144, 109, 23);
            contentPane.add(rdbtnVegloaded);

            SWgroup = new ButtonGroup();
            SWgroup.add(rdbtnAlooTikki);
            SWgroup.add(rdbtnBombayMasala);
            SWgroup.add(rdbtnTangyPaneer);
            SWgroup.add(rdbtnVegloaded);

            //////////////////////////////////////////////////////

            rdbtnChococheeseCake = new JRadioButton("Choco-Cheese Cake");
            rdbtnChococheeseCake.setBounds(308, 66, 139, 23);
            contentPane.add(rdbtnChococheeseCake);

            rdbtnRedvelvetCake = new JRadioButton("Red-Velvet Cake");
            rdbtnRedvelvetCake.setBounds(308, 92, 109, 23);
            contentPane.add(rdbtnRedvelvetCake);

            rdbtnBlueberryMuffin = new JRadioButton("Blue-Berry Muffin");
            rdbtnBlueberryMuffin.setBounds(308, 118, 109, 23);
            contentPane.add(rdbtnBlueberryMuffin);

            rdbtnApplepie = new JRadioButton("Apple-Pie");
            rdbtnApplepie.setBounds(308, 144, 109, 23);
            contentPane.add(rdbtnApplepie);

            Dessertgroup = new ButtonGroup();
            Dessertgroup.add(rdbtnChococheeseCake);
            Dessertgroup.add(rdbtnRedvelvetCake);
            Dessertgroup.add(rdbtnBlueberryMuffin);
            Dessertgroup.add(rdbtnApplepie);


            JLabel lblOrderSystem = new JLabel("Order System");
            lblOrderSystem.setBounds(170, 11, 72, 14);
            contentPane.add(lblOrderSystem);

            JLabel lblDrinks = new JLabel("Drinks");
            lblDrinks.setBounds(21, 41, 46, 14);
            contentPane.add(lblDrinks);

            JLabel lblFries = new JLabel("Fries");
            lblFries.setBounds(110, 41, 46, 14);
            contentPane.add(lblFries);

            JLabel lblSandwiches = new JLabel("Sandwiches");
            lblSandwiches.setBounds(216, 41, 72, 14);
            contentPane.add(lblSandwiches);

            JSeparator separator = new JSeparator();
            separator.setBounds(6, 61, 512, 2);
            contentPane.add(separator);

            JLabel lblDeserts = new JLabel("Deserts");
            lblDeserts.setBounds(320, 42, 46, 14);
            contentPane.add(lblDeserts);

            JSeparator separator_1 = new JSeparator();
            separator_1.setBounds(6, 298, 512, 2);
            contentPane.add(separator_1);

            btnAddToOrder = new JButton("ADD TO ORDER");
            btnAddToOrder.setBounds(96, 311, 109, 23);
            contentPane.add(btnAddToOrder);

            btnMakeBill = new JButton("MAKE BILL");
            btnMakeBill.setEnabled(false);
            btnMakeBill.setBounds(216, 311, 109, 23);
            contentPane.add(btnMakeBill);

            JLabel lblQuantity = new JLabel("Quantity");
            lblQuantity.setBounds(21, 187, 46, 14);
            contentPane.add(lblQuantity);

            String[] sizes = {"Small","Medium","Large"};
            drinkssize = new JComboBox<String>(sizes);
            drinkssize.setSelectedIndex(0);
            drinkssize.setBounds(10, 250, 75, 20);
            contentPane.add(drinkssize);


            drinkquan = new JTextField();
            drinkquan.setBounds(10, 205, 75, 20);
            contentPane.add(drinkquan);
```

```
•            drinkquan.setColumns(10);
•
•            JLabel lblSize = new JLabel("Size");
•            lblSize.setBounds(21, 232, 46, 14);
•            contentPane.add(lblSize);
•
•            JLabel label = new JLabel("Quantity");
•            label.setBounds(113, 187, 46, 14);
•            contentPane.add(label);
•
•            friesquan = new JTextField();
•            friesquan.setColumns(10);
•            friesquan.setBounds(102, 205, 75, 20);
•            contentPane.add(friesquan);
•
•            JLabel label_1 = new JLabel("Size");
•            label_1.setBounds(113, 232, 46, 14);
•            contentPane.add(label_1);
•
•            friessize = new JComboBox<String>(sizes);
•            friessize.setSelectedIndex(0);
•            friessize.setBounds(102, 250, 75, 20);
•            contentPane.add(friessize);
•
•            JLabel label_2 = new JLabel("Quantity");
•            label_2.setBounds(227, 187, 46, 14);
•            contentPane.add(label_2);
•
•            swquan = new JTextField();
•            swquan.setColumns(10);
•            swquan.setBounds(216, 205, 75, 20);
•            contentPane.add(swquan);
•
•            JLabel label_3 = new JLabel("Size");
•            label_3.setBounds(227, 232, 46, 14);
•            contentPane.add(label_3);
•
•            swsize = new JComboBox<String>(sizes);
•            swsize.setSelectedIndex(0);
•            swsize.setBounds(216, 250, 75, 20);
•            contentPane.add(swsize);
•
•
•            JLabel label_4 = new JLabel("Quantity");
•            label_4.setBounds(331, 187, 46, 14);
•            contentPane.add(label_4);
•
•            dessertquan = new JTextField();
•            dessertquan.setColumns(10);
•            dessertquan.setBounds(320, 205, 75, 20);
•            contentPane.add(dessertquan);
```

```
		JLabel label_5 = new JLabel("Size");
		label_5.setBounds(331, 232, 46, 14);
		contentPane.add(label_5);

		dessertsize = new JComboBox<String>(sizes);
		dessertsize.setSelectedIndex(0);
		dessertsize.setBounds(320, 250, 75, 20);
		contentPane.add(dessertsize);

		JSeparator separator_2 = new JSeparator();
		separator_2.setOrientation(SwingConstants.VERTICAL);
		separator_2.setBounds(518, 41, 2, 311);
		contentPane.add(separator_2);

		JLabel lblBill = new JLabel("Bill");
		lblBill.setBounds(575, 41, 46, 14);
		contentPane.add(lblBill);

		JLabel lblTotal = new JLabel("TOTAL");
		lblTotal.setBounds(543, 70, 46, 14);
		contentPane.add(lblTotal);

		JLabel lblTax = new JLabel("TAX(3%)");
		lblTax.setBounds(543, 96, 46, 14);
		contentPane.add(lblTax);

		JLabel lblDonate = new JLabel("DONATE");
		lblDonate.setBounds(543, 122, 46, 14);
		contentPane.add(lblDonate);

		total = new JTextField();
		total.setBounds(598, 67, 86, 20);
		contentPane.add(total);
		total.setColumns(10);

		tax = new JTextField();
		tax.setColumns(10);
		tax.setBounds(598, 93, 86, 20);
		contentPane.add(tax);

		donate = new JTextField();
		donate.setColumns(10);
		donate.setBounds(599, 119, 86, 20);
		contentPane.add(donate);

		JSeparator separator_3 = new JSeparator();
		separator_3.setBounds(522, 165, 168, 2);
		contentPane.add(separator_3);

		JLabel lblAmount = new JLabel("AMOUNT");
```

```
•           lblAmount.setBounds(543, 186, 46, 14);
•           contentPane.add(lblAmount);
•
•           amount = new JTextField();
•           amount.setColumns(10);
•           amount.setBounds(598, 181, 86, 20);
•           contentPane.add(amount);
•
•           btnClear = new JButton("CLEAR");
•           btnClear.setEnabled(false);
•           btnClear.setBounds(557, 228, 89, 23);
•           contentPane.add(btnClear);
•
•           btnSave = new JButton("SAVE");
•           btnSave.setEnabled(false);
•           btnSave.setBounds(557, 277, 89, 23);
•           contentPane.add(btnSave);
•
•           JLabel lblReport = new JLabel("");
•           lblReport.setBounds(7, 338, 382, 56);
•           contentPane.add(lblReport);
•
•           btnAddToOrder.addActionListener(this);
•           btnMakeBill.addActionListener(this);
•           btnClear.addActionListener(this);
•           btnSave.addActionListener(this);
•       }
•
•       public void actionPerformed(ActionEvent e) {
•           if(e.getSource().equals(btnAddToOrder)) {
•               //for drinks
•               if( rdbtnTea.isSelected() || rdbtnCoffee.isSelected() ||
    rdbtnJuice.isSelected() || rdbtnColddrink.isSelected()
    && !(drinkquan.getText().equals("0"))) {
•                   try {
•                       String sizf = (String)
    drinkssize.getSelectedItem();
•                       amount_of  =
    Integer.parseInt(drinkquan.getText().trim());
•                       Beverage bvg;
•                       if(rdbtnTea.isSelected())
•                       {
•                           bvg = new Tea(sizf,amount_of,this);
•                       }
•                       else if(rdbtnCoffee.isSelected())
•                       {
•                           bvg = new Coffee(sizf,amount_of,this);
•                       }
•                       else if(rdbtnColddrink.isSelected())
•                       {
•                           bvg = new Colddrink(sizf,amount_of,this);
```

```
                    }
                    else
                    {
                        bvg = new Juice(sizf,amount_of,this);
                    }
                    drinkquan.setText(null);
                    //JOptionPane.showMessageDialog(null, "check  1");
                    list_of_bvr.add(bvg);
                    //JOptionPane.showMessageDialog(null, "check 2");
                    JOptionPane.showMessageDialog(null,
bvg.toString());

                    //JOptionPane.showMessageDialog(null, "check 3");
                    btnMakeBill.setEnabled(true);

                    Drinksgroup.clearSelection();
                }
                catch(NumberFormatException e1) {//if written data in
TextField can't be converted to an integer[String,char,double etc...]
                    JOptionPane.showMessageDialog(this, "Enter an
integer as the amount");
                }
            }

            if( rdbtnSimple.isSelected() || rdbtnPeriperi.isSelected()
|| rdbtnMasalaCheese.isSelected() || rdbtnMexicanLoaded.isSelected()
&& !(friesquan.getText().equals("0"))) {
                try {
                    String sizf = (String)
friessize.getSelectedItem();
                    amount_of  =
Integer.parseInt(friesquan.getText().trim());
                    Beverage bvg;
                    if(rdbtnSimple.isSelected())
                    {
                        bvg = new Simple(sizf,amount_of,this);
                    }
                    else if(rdbtnPeriperi.isSelected())
                    {
                        bvg = new PeriPeri(sizf,amount_of,this);
                    }
                    else if(rdbtnMasalaCheese.isSelected())
                    {
                        bvg = new MasalaCheese(sizf,amount_of,this);
                    }
                    else
                    {
                        bvg = new MexicanLoaded(sizf,amount_of,this);
                    }
                    friesquan.setText(null);
                    //JOptionPane.showMessageDialog(null, "check  1");
```

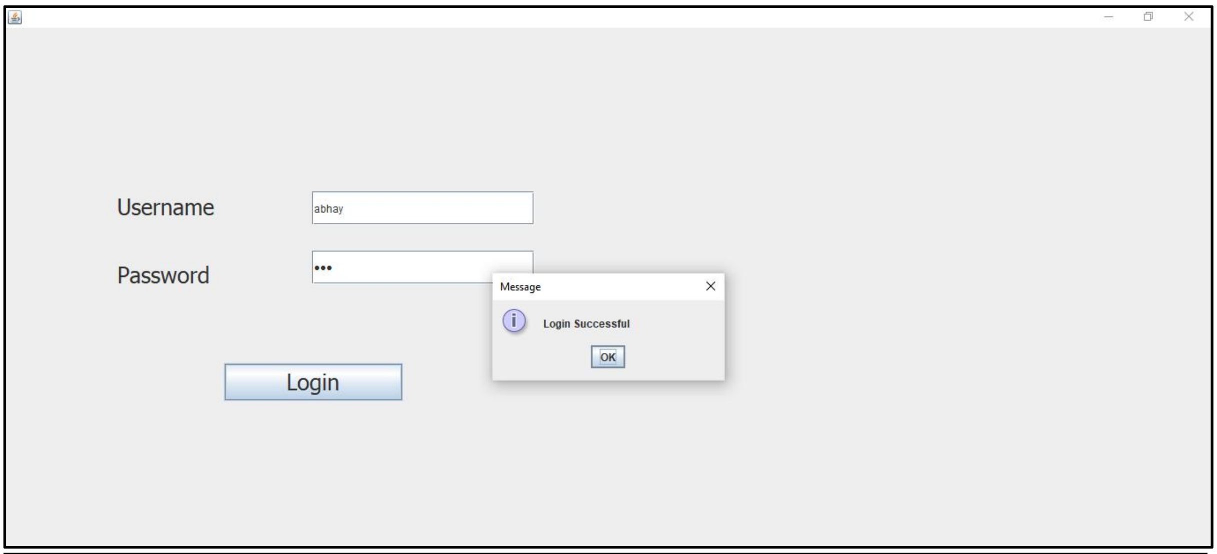
```
•                       list_of_bvr.add(bvg);
•                       //JOptionPane.showMessageDialog(null, "check 2");
•                       JOptionPane.showMessageDialog(null,
   bvg.toString());
•
•                       //JOptionPane.showMessageDialog(null, "check 3");
•                       btnMakeBill.setEnabled(true);
•
•                       Friesgroup.clearSelection();
•
•                   }
•                   catch(NumberFormatException e1) {//if written data in
   TextField can't be converted to an integer[String,char,double etc...]
•                       JOptionPane.showMessageDialog(this, "Enter an
   integer as the Quantity");
•                   }
•               }
•
•               if( rdbtnAlooTikki.isSelected() ||
   rdbtnBombayMasala.isSelected() || rdbtnTangyPaneer.isSelected() ||
   rdbtnVegloaded.isSelected() && !(swquan.getText().equals("0"))) {
•                   try {
•                       String sizf = (String) swsize.getSelectedItem();
•                       amount_of  =
   Integer.parseInt(swquan.getText().trim());
•                       Beverage bvg;
•                       if(rdbtnAlooTikki.isSelected())
•                       {
•                           bvg = new AlooTikki(sizf,amount_of,this);
•                       }
•                       else if(rdbtnBombayMasala.isSelected())
•                       {
•                           bvg = new BombayMasala(sizf,amount_of,this);
•                       }
•                       else if(rdbtnTangyPaneer.isSelected())
•                       {
•                           bvg = new TangyPaneer(sizf,amount_of,this);
•                       }
•                       else
•                       {
•                           bvg = new VegLoaded(sizf,amount_of,this);
•                       }
•                       swquan.setText(null);
•                       //JOptionPane.showMessageDialog(null, "check  1");
•                       list_of_bvr.add(bvg);
•                       //JOptionPane.showMessageDialog(null, "check 2");
•                       JOptionPane.showMessageDialog(null,
   bvg.toString());
•
•                       //JOptionPane.showMessageDialog(null, "check 3");
•                       btnMakeBill.setEnabled(true);
```

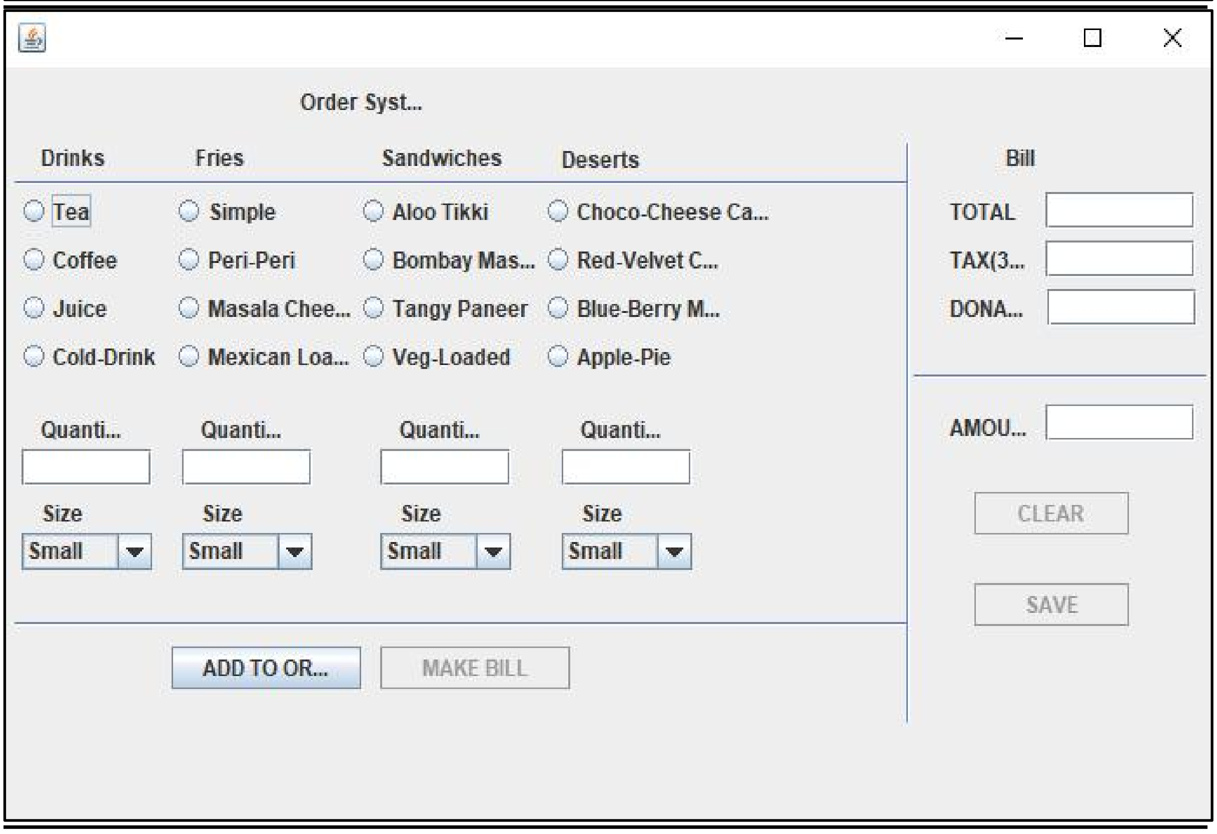
```
                    SWgroup.clearSelection();
                }
                catch(NumberFormatException e1) {//if written data in TextField can't be converted to an integer[String,char,double etc...]
                    JOptionPane.showMessageDialog(this, "Enter an integer as the quantity");
                }
            }

            if( rdbtnChococheeseCake.isSelected() || rdbtnRedvelvetCake.isSelected() || rdbtnBlueberryMuffin.isSelected() || rdbtnApplepie.isSelected() && !(dessertquan.getText().equals("0"))) {
                try {
                    String sizf = (String) dessertsize.getSelectedItem();
                    amount_of  = Integer.parseInt(dessertquan.getText().trim());
                    Beverage bvg;
                    if(rdbtnChococheeseCake.isSelected())
                    {
                        bvg = new CC_cake(sizf,amount_of,this);
                    }
                    else if(rdbtnRedvelvetCake.isSelected())
                    {
                        bvg = new RV_cake(sizf,amount_of,this);
                    }
                    else if(rdbtnBlueberryMuffin.isSelected())
                    {
                        bvg = new BB_muffin(sizf,amount_of,this);
                    }
                    else
                    {
                        bvg = new ApplePie(sizf,amount_of,this);
                    }
                    dessertquan.setText(null);
                    //JOptionPane.showMessageDialog(null, "check  1");
                    list_of_bvr.add(bvg);
                    //JOptionPane.showMessageDialog(null, "check 2");
                    JOptionPane.showMessageDialog(null, bvg.toString());

                    //JOptionPane.showMessageDialog(null, "check 3");
                    btnMakeBill.setEnabled(true);

                    Dessertgroup.clearSelection();

                }
                catch(NumberFormatException e1) {//if written data in TextField can't be converted to an integer[String,char,double etc...]
```

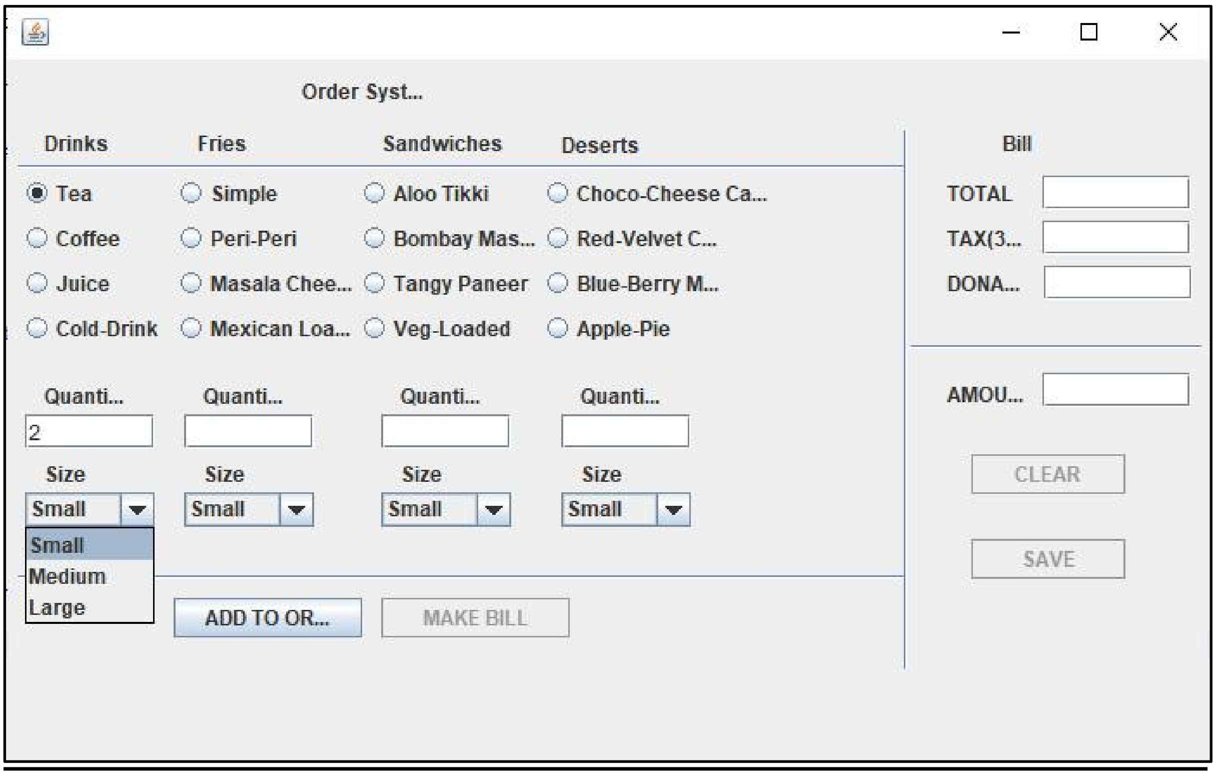
```
                    JOptionPane.showMessageDialog(this, "Enter an
integer as the quantity");
                }
            }

        }


        if(e.getSource().equals(btnMakeBill)) {
            String report = "";
            double pay=0.0;
            for(int i=0;i<list_of_bvr.size();i++) {
                Beverage bvgi = list_of_bvr.get(i);
                report += bvgi.toString();
                double totalprice_of_bvg = bvgi.getAmount() *
bvgi.getPrice();
                pay += totalprice_of_bvg;
                report = report + " - "+totalprice_of_bvg+" Rs. \n";
            }

            DecimalFormat df2 = new DecimalFormat("#.##");

            total.setText(df2.format(pay));
            tax.setText(df2.format(pay*0.3));
            amount.setText(df2.format(pay*1.3 + 10));
            donate.setText("10");
            double qq = pay*1.3 + 10;
            JOptionPane.showMessageDialog(this, report);
            JOptionPane.showMessageDialog(this,  "You should pay
"+qq+" Rs.");
            btnMakeBill.setEnabled(false);
            btnClear.setEnabled(true);
            btnSave.setEnabled(true);
            list_of_bvr.clear();
            SaveReport = report;
        }

        if(e.getSource().equals(btnClear)) {
            JOptionPane.showMessageDialog(null, "Cleared");
            total.setText(null);
            amount.setText(null);
            donate.setText(null);
            tax.setText(null);
            SaveReport = "";
        }

        if(e.getSource().equals(btnSave)) {
            JOptionPane.showMessageDialog(null, "Saved");
            try {
                FileWriter writer = new FileWriter("MyFile.txt",
true);
```

```
•                       BufferedWriter bufferedWriter = new
   BufferedWriter(writer);
•
•                       bufferedWriter.write(SaveReport);
•                       bufferedWriter.newLine();
•                       bufferedWriter.close();
•
•                   } catch (IOException e1) {
•                       // TODO Auto-generated catch block
•                       e1.printStackTrace();
•                   }
•               }
•
•           }
•   }
```

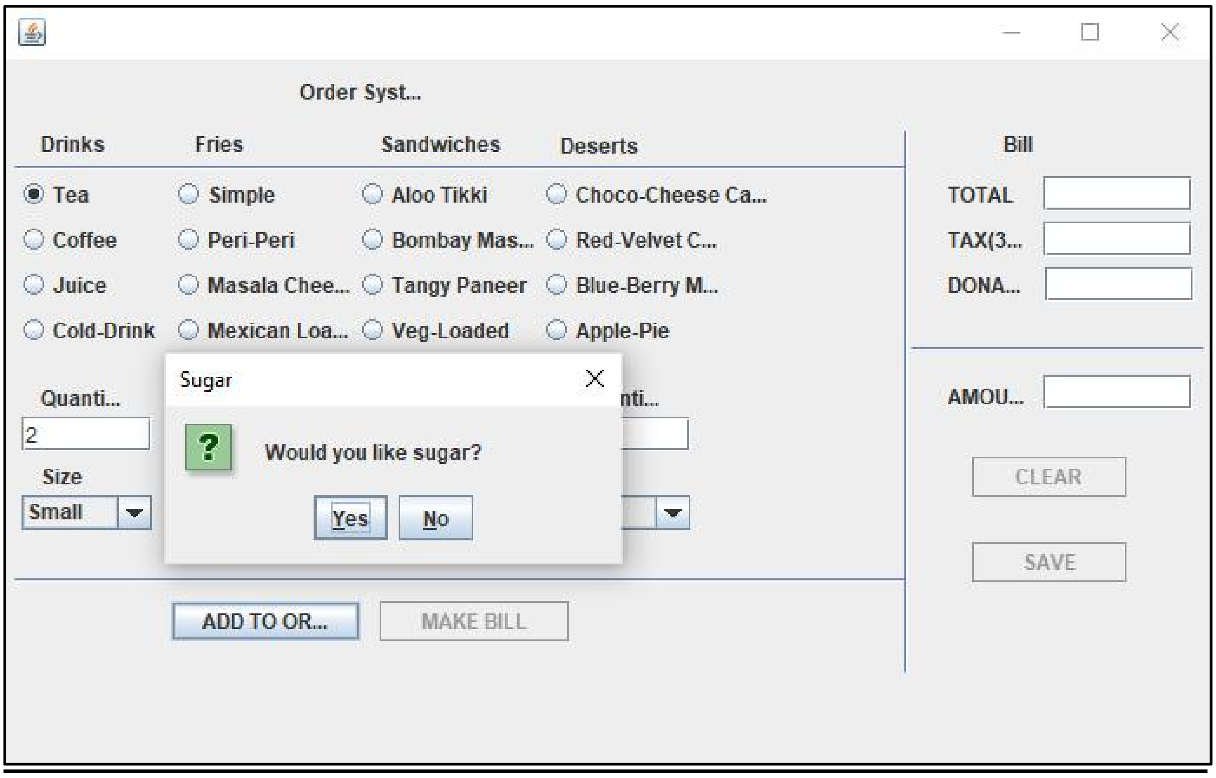
SCREENSHOTS







Drinks: Tea [✓]
Quanti...: 2
Size: Small

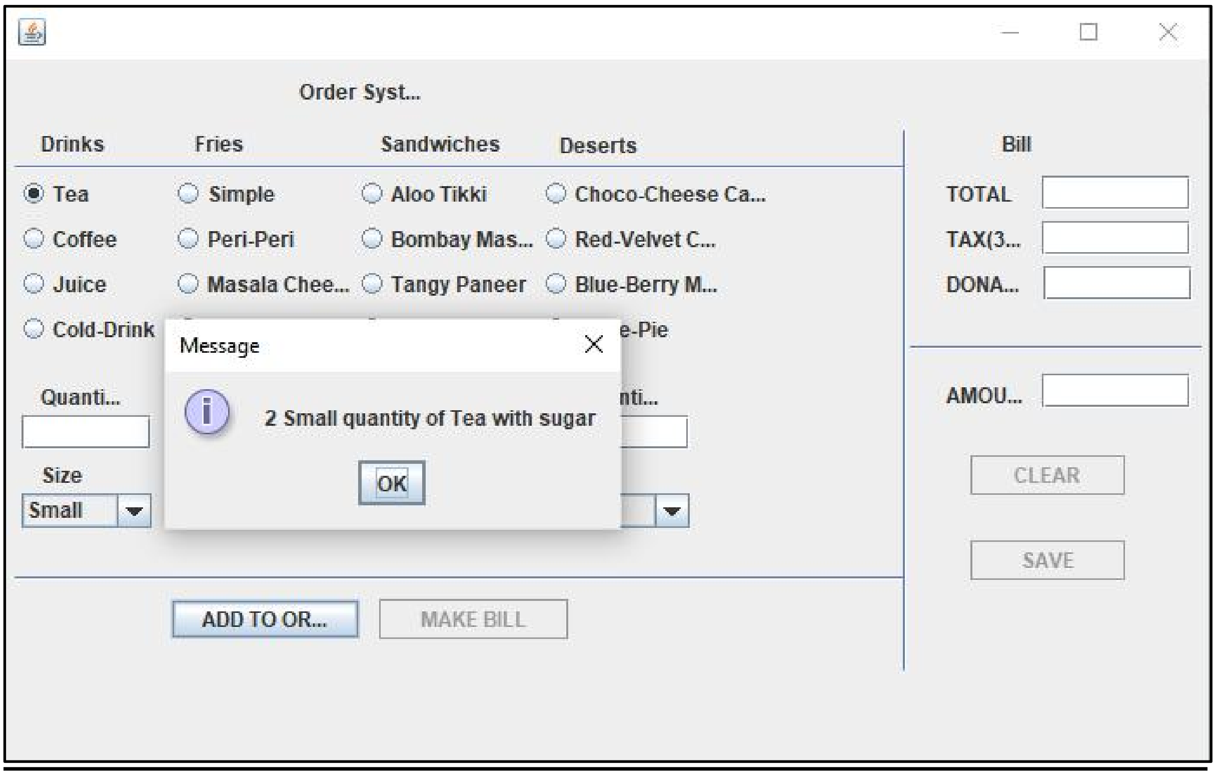


Drinks: Tea [✓]
Quanti...: 2
Size: Small

Sugar

Would you like sugar?

Yes No

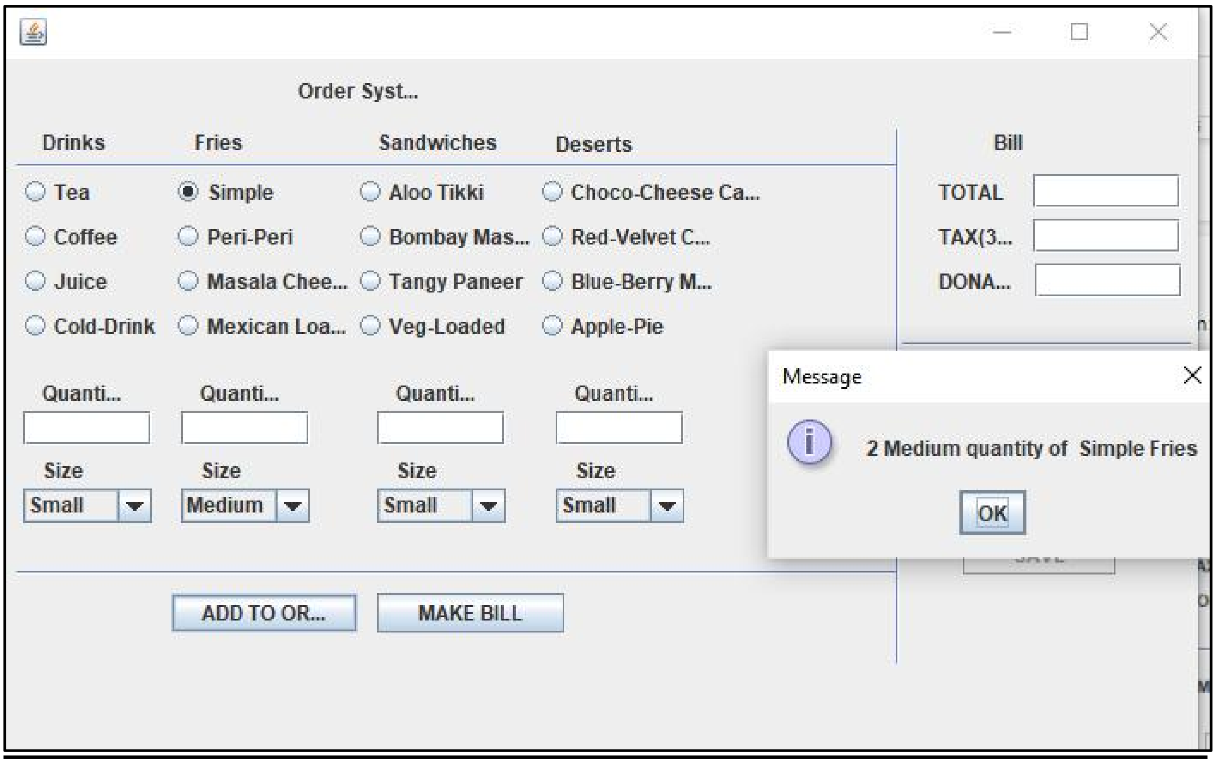


Message

2 Small quantity of Tea with sugar

Drinks: [✓ Tea]

Size: Small

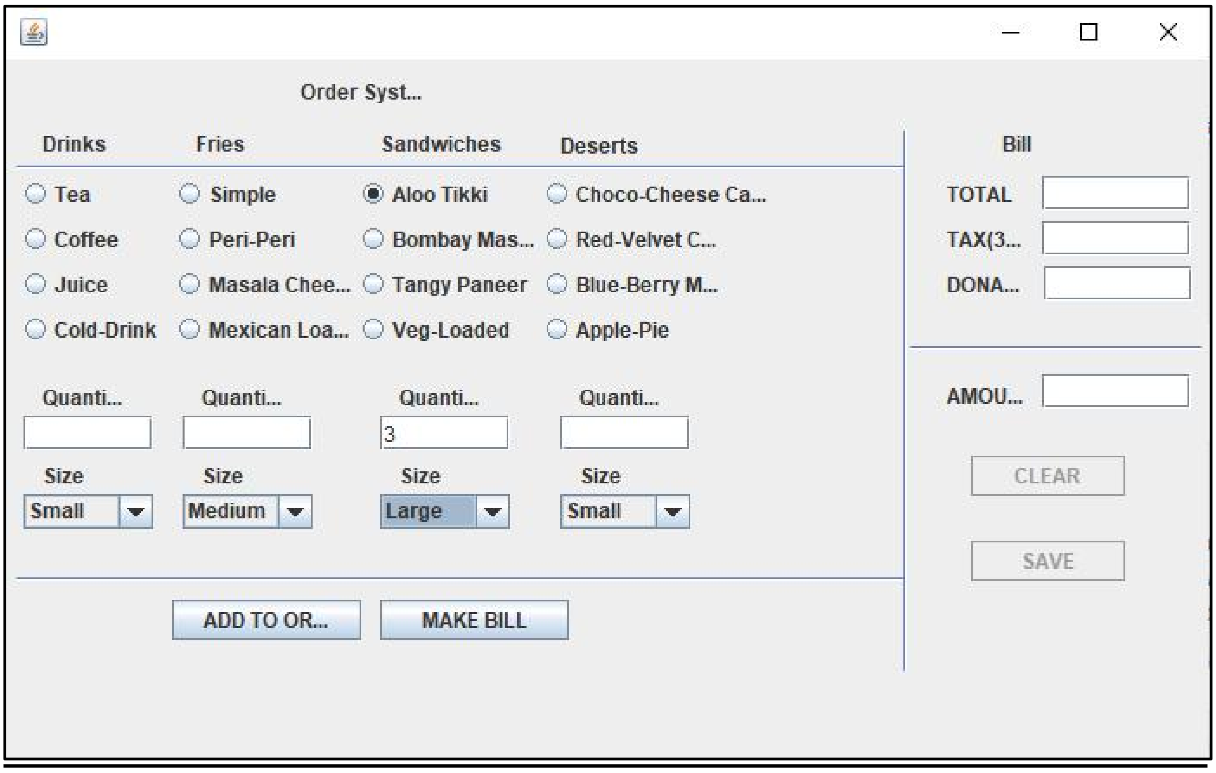


Message

2 Medium quantity of Simple Fries

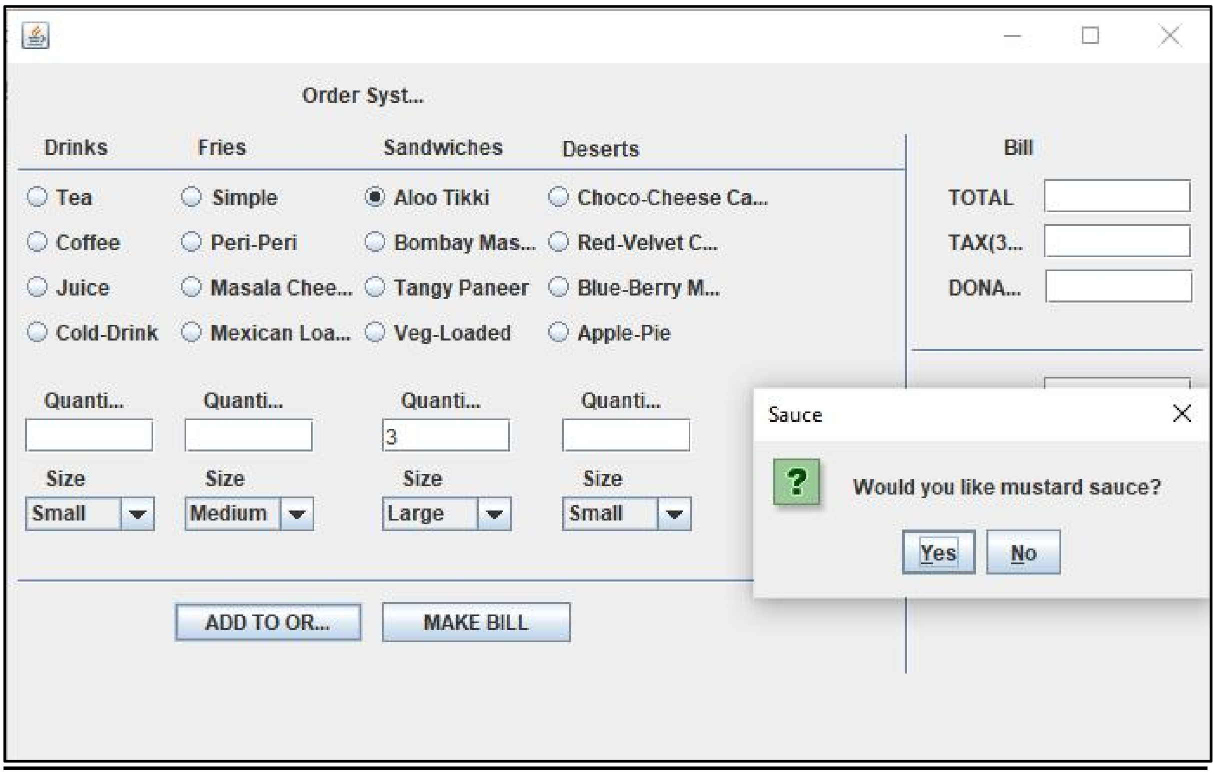
Fries: [✓ Simple]

Size: Small, Medium, Small, Small



Sandwiches: (●) Aloo Tikki

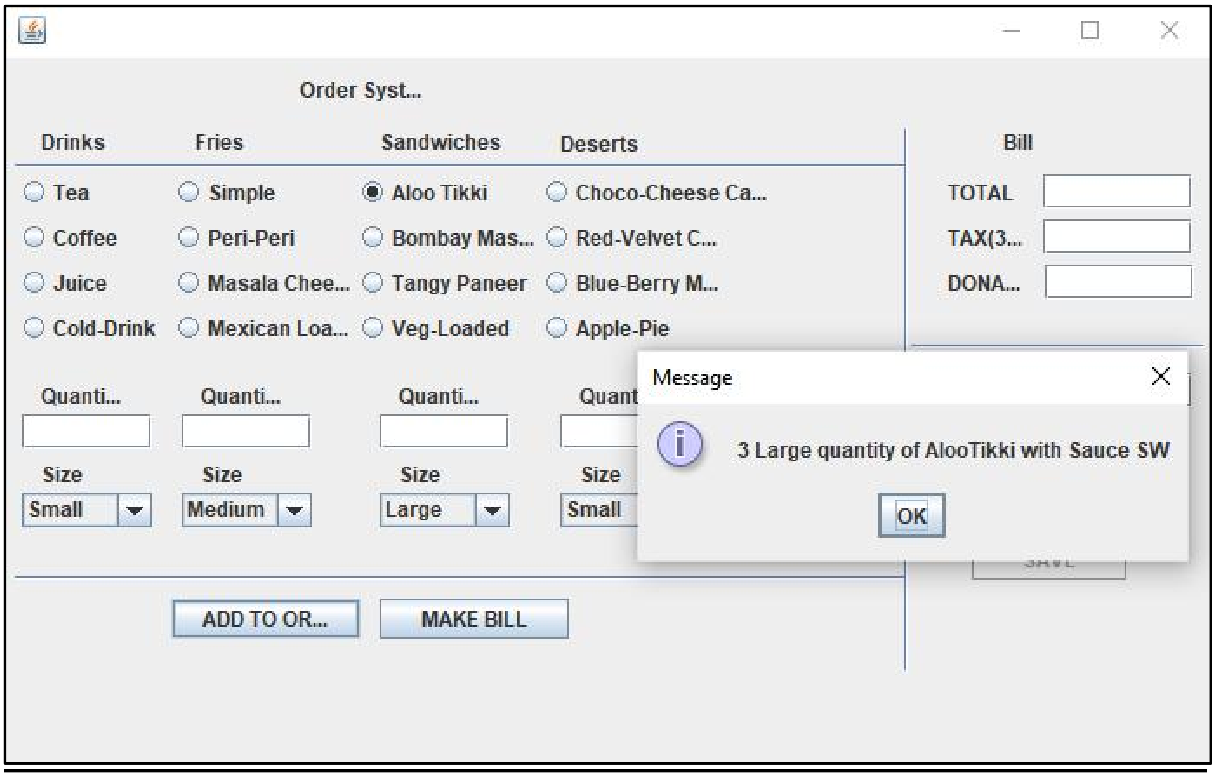
	Drinks	Fries	Sandwiches	Deserts
Quanti...			3	
Size	Small	Medium	Large	Small



Sandwiches: (●) Aloo Tikki

	Drinks	Fries	Sandwiches	Deserts
Quanti...			3	
Size	Small	Medium	Large	Small

Would you like mustard sauce?

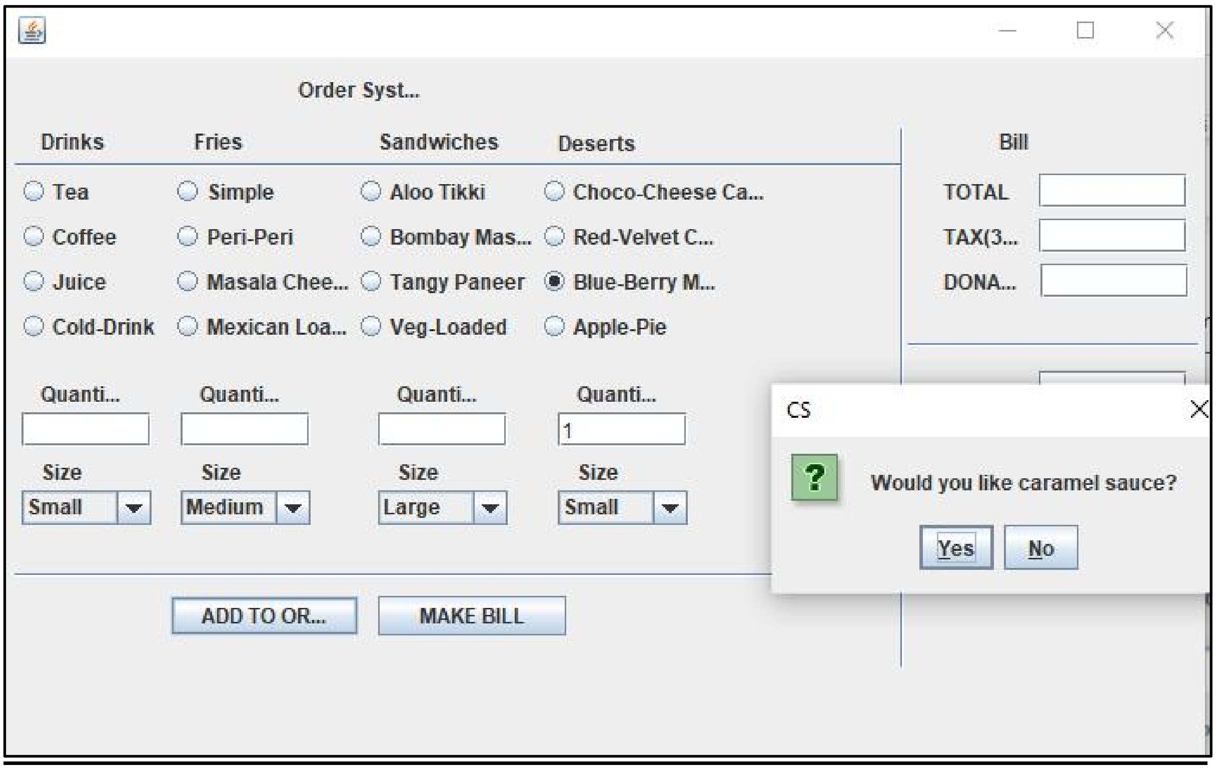


● Aloo Tikki

Size: Small | Medium | Large | Small

Message

3 Large quantity of AlooTikki with Sauce SW



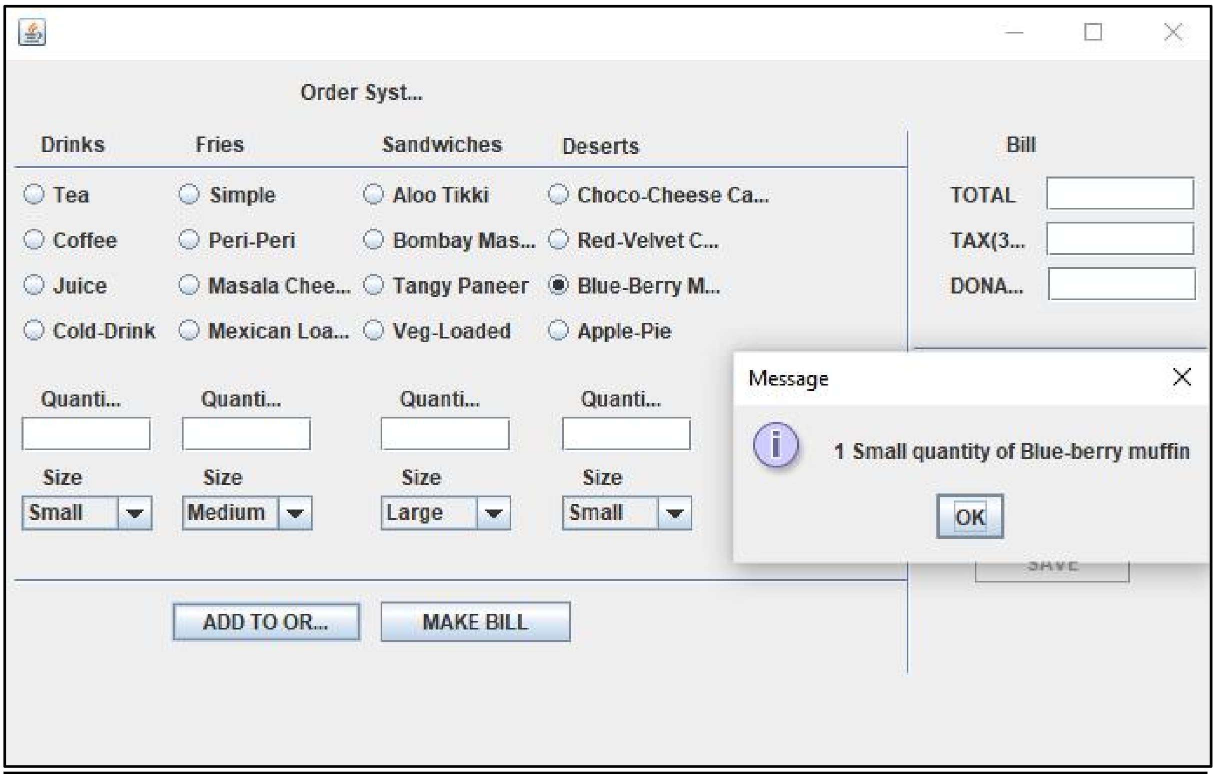
● Blue-Berry M...

Quanti...: 1

Size: Small | Medium | Large | Small

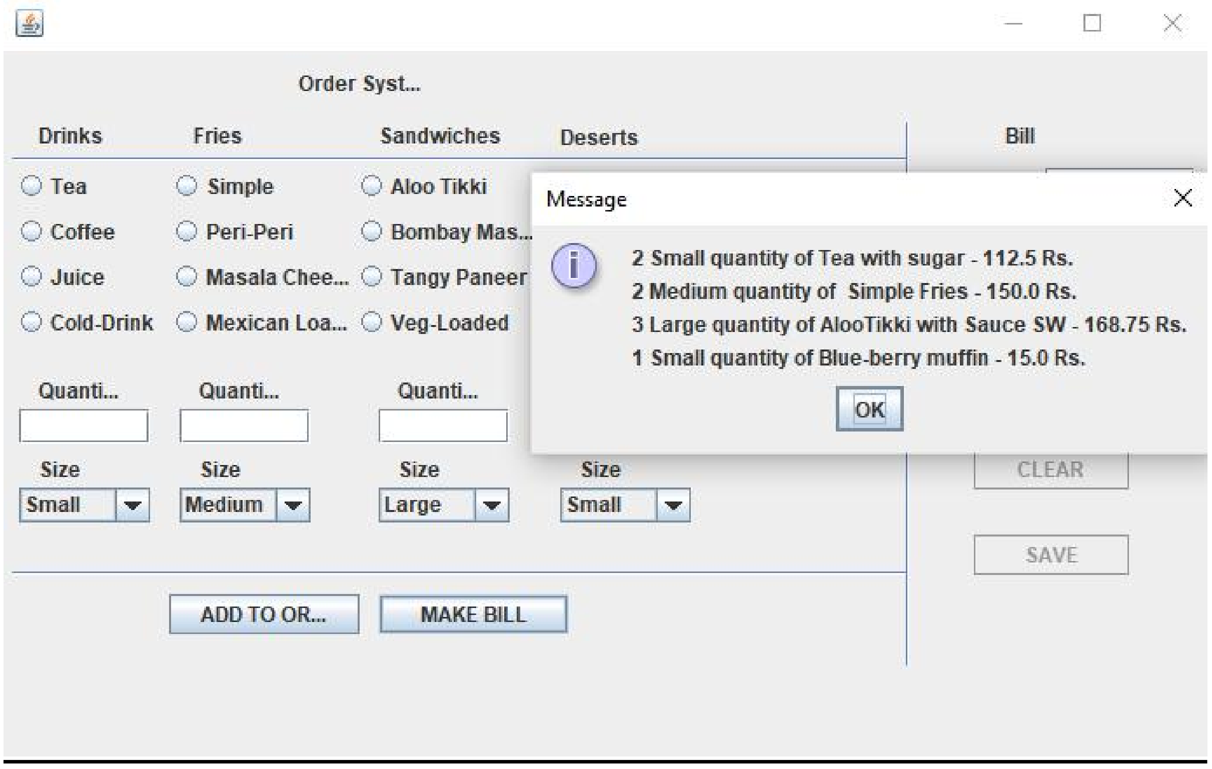
CS

Would you like caramel sauce?



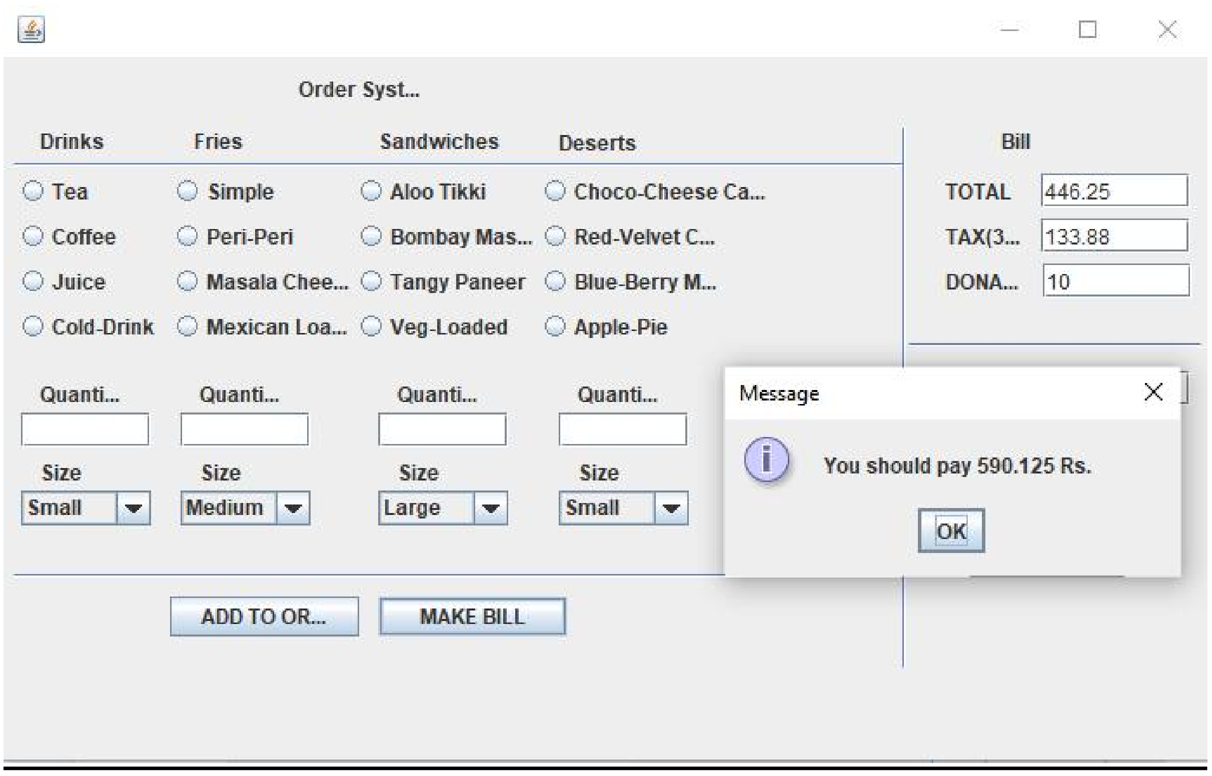
Message

1 Small quantity of Blue-berry muffin



Message

2 Small quantity of Tea with sugar - 112.5 Rs.
2 Medium quantity of Simple Fries - 150.0 Rs.
3 Large quantity of AlooTikki with Sauce SW - 168.75 Rs.
1 Small quantity of Blue-berry muffin - 15.0 Rs.



TOTAL 446.25

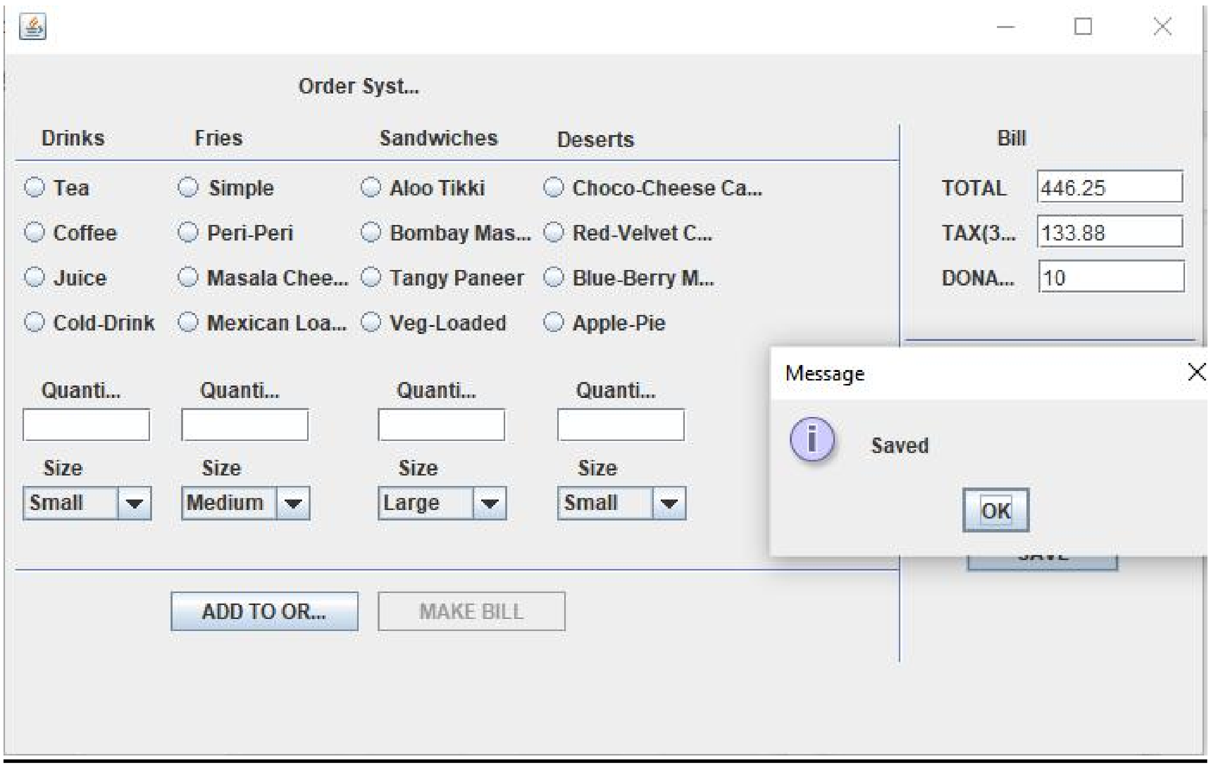
TAX(3... 133.88

DONA... 10

Size: Small, Medium, Large, Small

Message

You should pay 590.125 Rs.



TOTAL 446.25

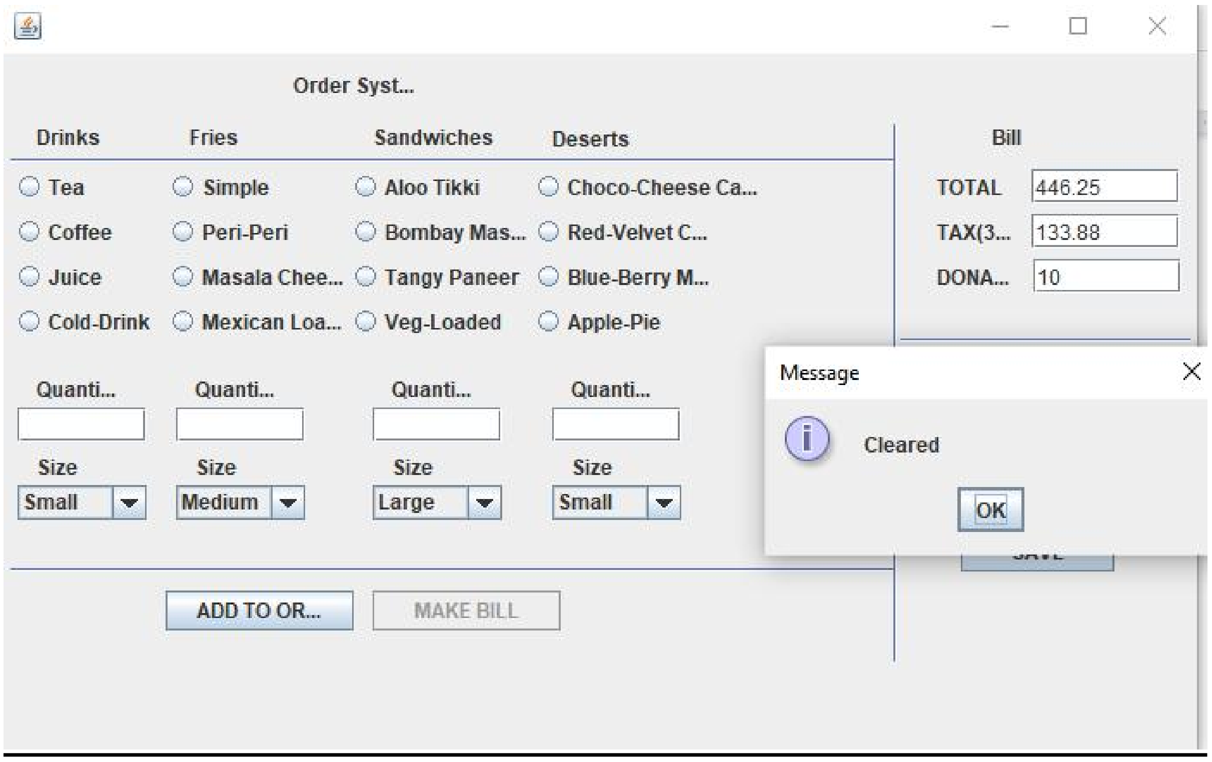
TAX(3... 133.88

DONA... 10

Size: Small, Medium, Large, Small

Message

Saved

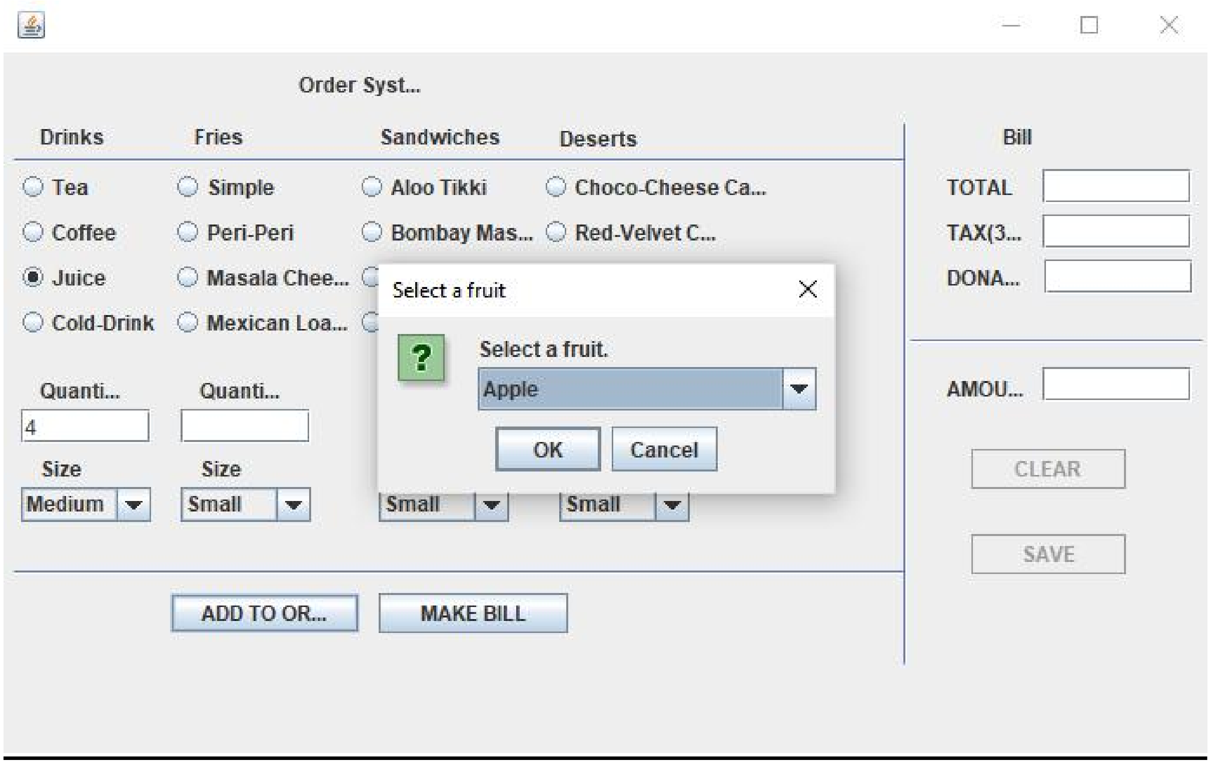


TOTAL 446.25
TAX(3... 133.88
DONA... 10

Size: Small, Medium, Large, Small

Message

Cleared



● Juice

Quanti... 4

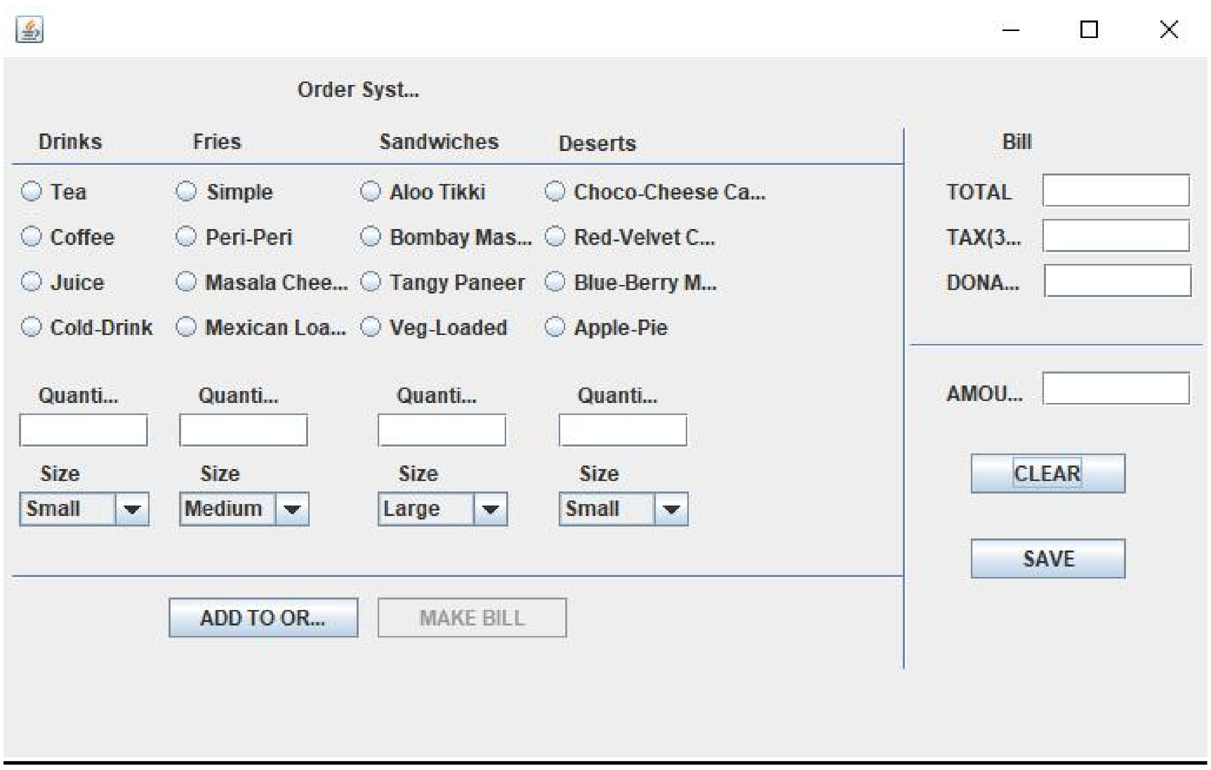
Size: Medium, Small, Small, Small

TOTAL
TAX(3...
DONA...
AMOU...

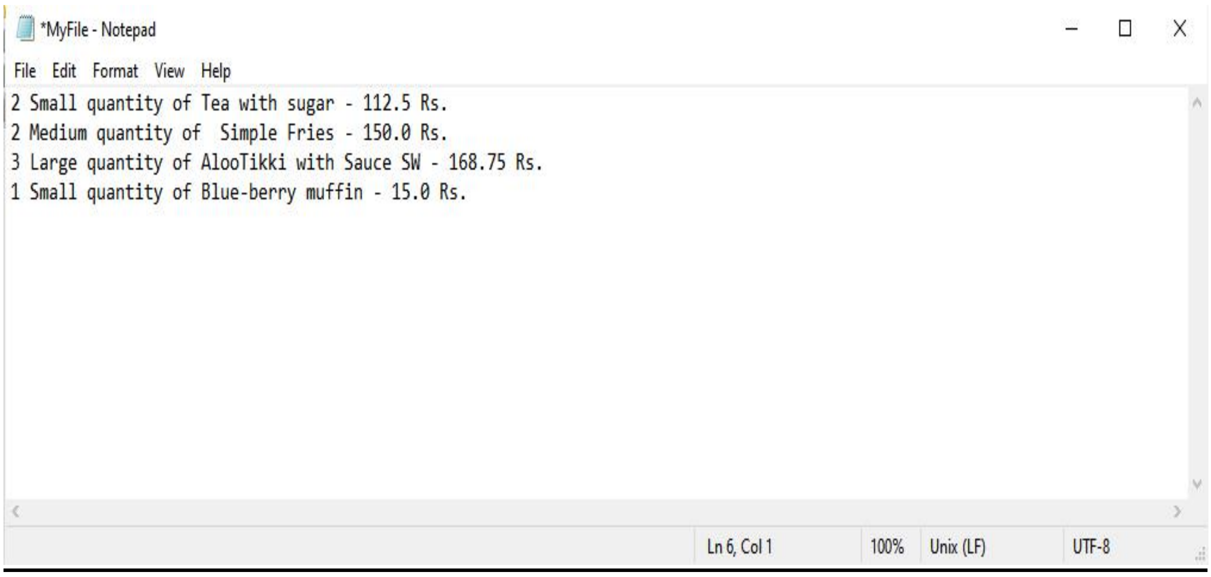
Select a fruit

Select a fruit.

Apple

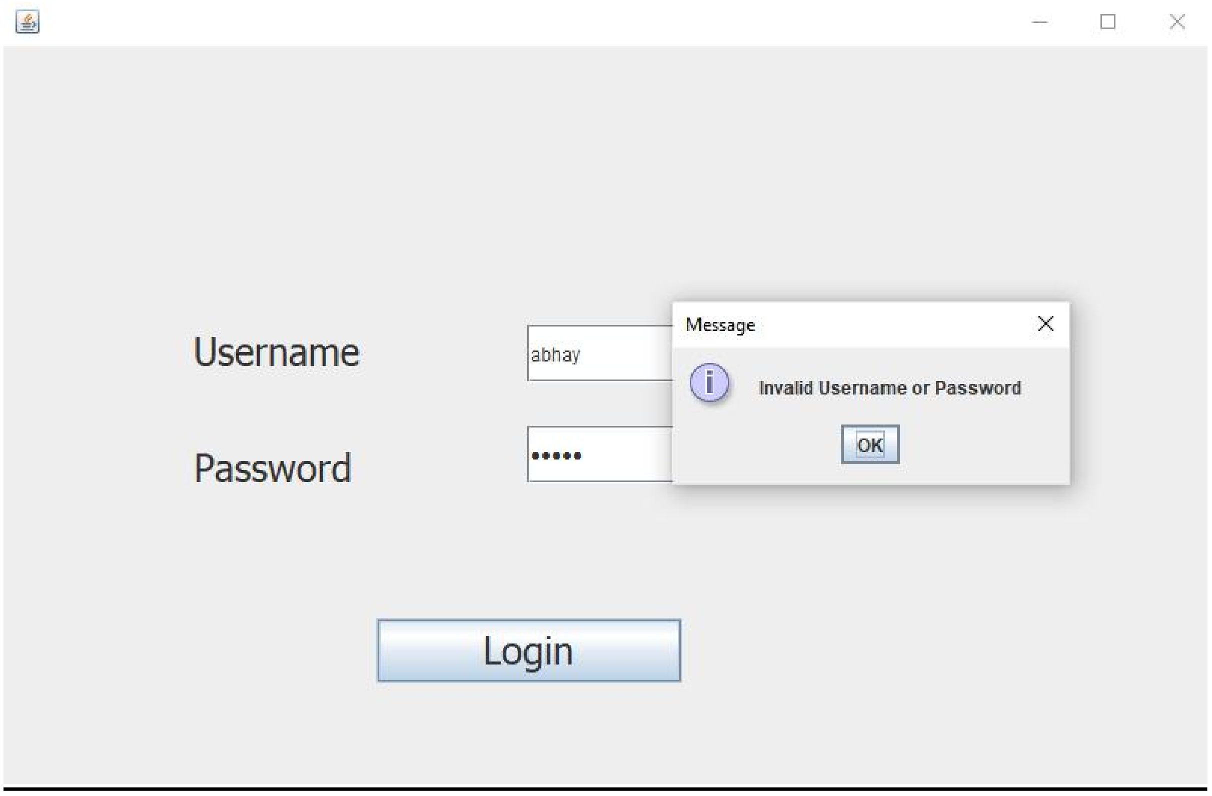


RECORDS FILE:-

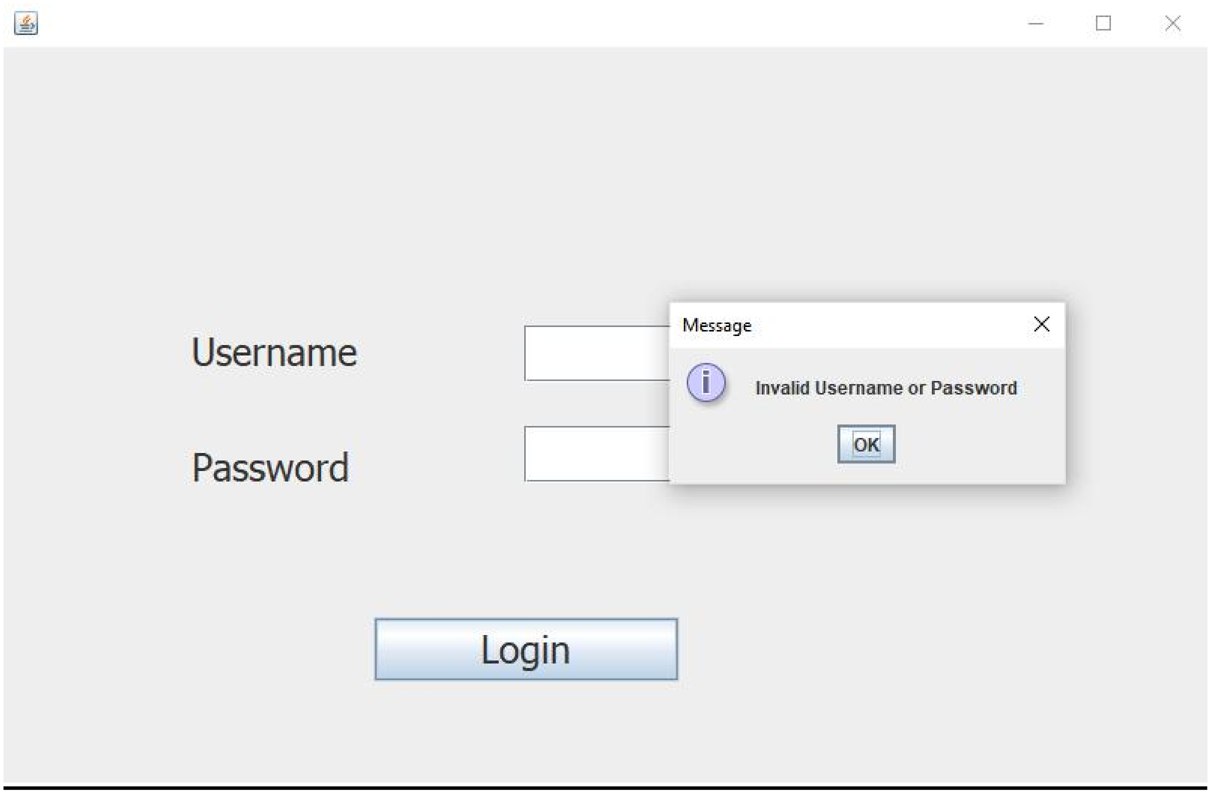


```
2 Small quantity of Tea with sugar - 112.5 Rs.
2 Medium quantity of  Simple Fries - 150.0 Rs.
3 Large quantity of AlooTikki with Sauce SW - 168.75 Rs.
1 Small quantity of Blue-berry muffin - 15.0 Rs.
```

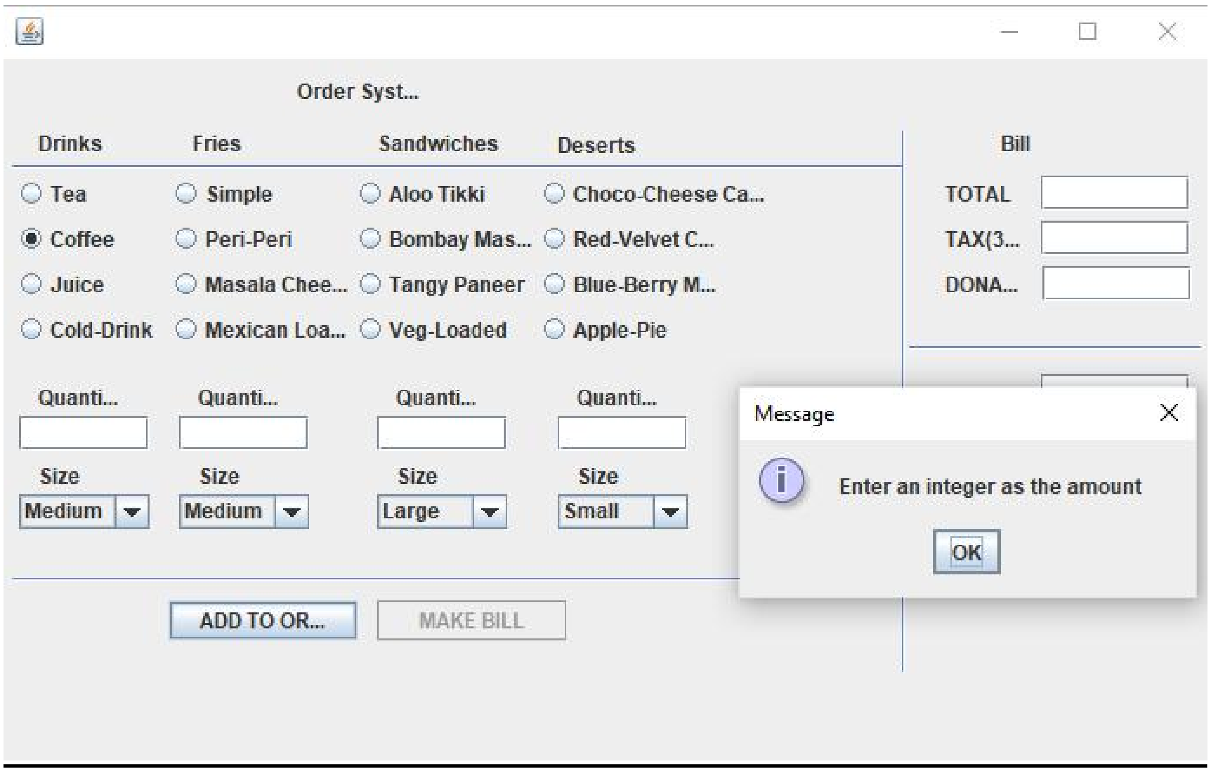
INVALID INPUTS:-



Invalid Username or Password



Invalid Username or Password

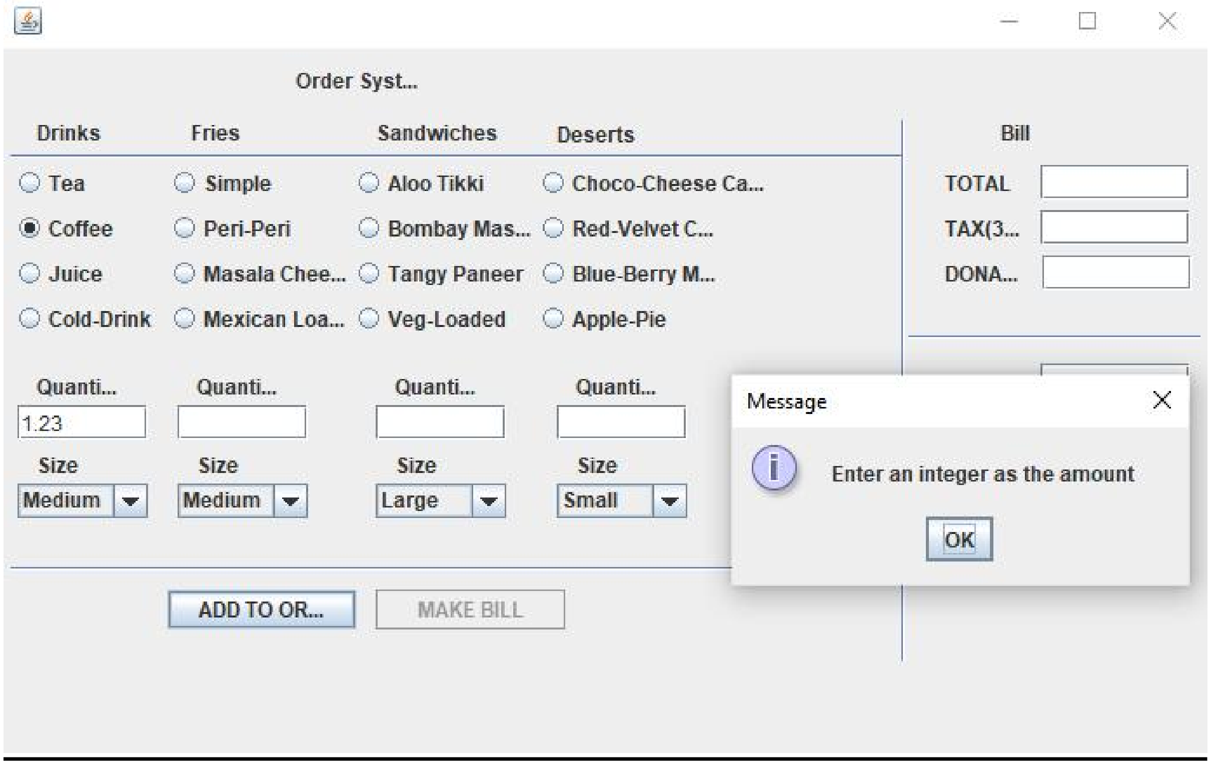


Drinks: Tea [✓ Coffee] Juice Cold-Drink

Size: Medium | Medium | Large | Small

Message

Enter an integer as the amount



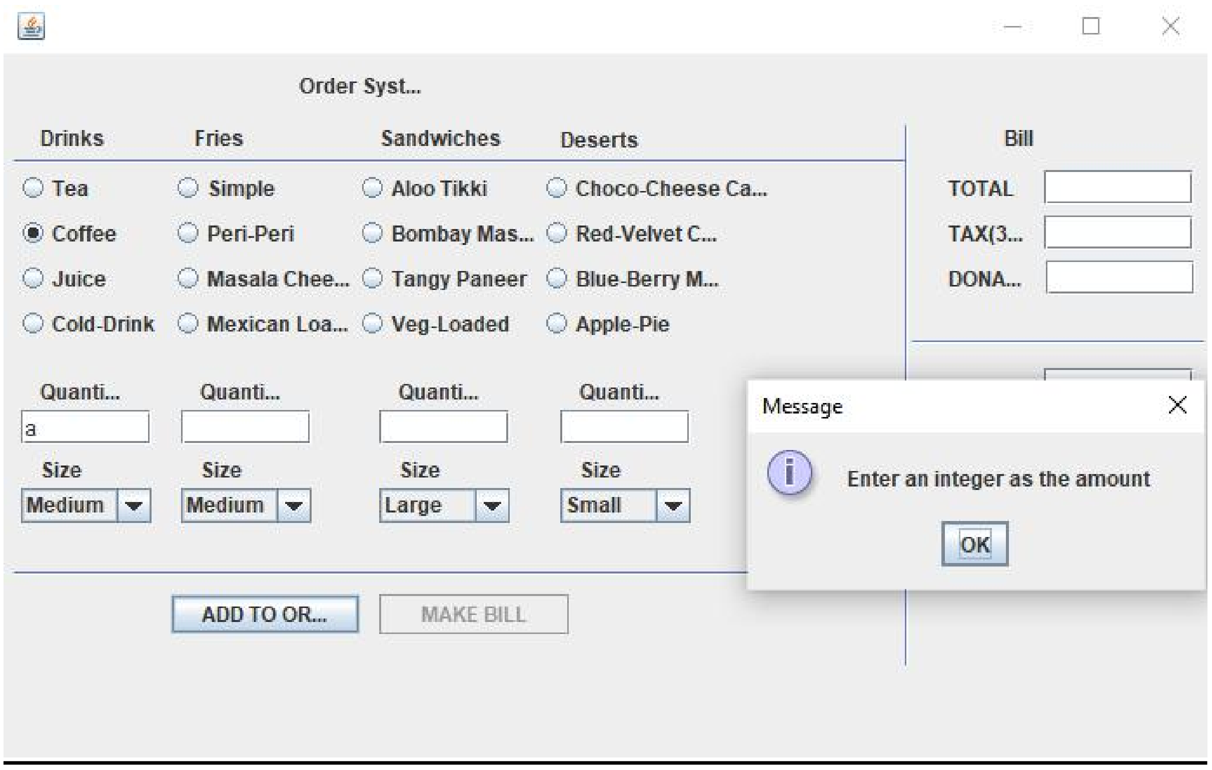
Drinks: Tea [✓ Coffee] Juice Cold-Drink

Quanti...: 1.23

Size: Medium | Medium | Large | Small

Message

Enter an integer as the amount



Drinks: [✓ Coffee]

Quanti...: a

Size: Medium | Medium | Large | Small

Enter an integer as the amount

TEST PLAN

Introduction :-

This application is intended to calculate the total bill of an order. Operator is required to enter the required items in the cart. The bill will be generated based on various items and different add-ons from the menu. The final bill will be generated and given to the user.

Intended Audience :-

This test plan is made for system testing of this application.
The test plan will be referred by:-
▪ Development manager and development team
▪ Restaurant owners
▪ Cafe owners

Intended Users :-

This application will be used by small café and restaurant owners.

Test Scope :-

System testing shall cover :-
▪ User interface testing
▪ Functionality testing

Out of Scope :-

The following types of testing are out of scope for this application :-
▪ Security testing
▪ Performance, volume, and stress testing
▪ Usability testing

Test Objectives :-

Targeted number of defects—500
▪ Targeted number of test scenario—100
▪ Targeted number of test cases—2000
▪ Number of iterations required—5
▪ Test scenario writing per day—10
▪ Test case writing per day—50
▪ Test case execution per day—100
▪ Coverage of requirements P1 = 100% P2 = 50% P3 = 10%

Assumptions of Testing :-

▪ Users understand English and they are capable of using a computer.
▪ Users understand only integers (they are not expected to use decimal)

▪ Application will be working on Windows 10 professionals. Other operating systems, if any, are compatible.
▪ Application is tested on Core i5, quad-core processor. Other configurations used, if any, are compatible.
▪ Testing considers right-handed users as well as left-handed users.

Risk Analysis :-

▪ Application will not be able to handle non-integers.
▪ Application will not handle char data.
▪ User may not understand English.

Workflow :-

The project manager communicates the availability of new builds along with the records of reviews and unit testing to the test manager. The application is installed on the machine as defined in prerequisites (Core i5).
Test manager checks the entry criteria for testing. If the application passes the entry criteria, it is taken for iteration testing. Test cases are stored in the same library.

Test Design :-

The application will be tested as per steps described in Table 1.

Testing Levels	Responsibility
Unit Testing when code is generated.	Developers
Integration testing.	Developers
System testing.	Testers

Test Data Definition :-

Test data is defined using the following techniques:-
▪ Equivalence partitioning.
▪ Error guessing.

Test Schedule and Planned Resources :-

The project started on 28th Aug 2020
Test planning—Aug 29th, 2020
Test scenario writing, review and updation— Aug 29th, 2020
Test case writing, review and updation— Aug 30th, 2020
Test case execution 1st iteration— Aug 31st, 2020
Test case execution 2nd iteration— Sept 01st, 2020
Test execution 3rd iteration— Sept 02nd, 2020

Test Data Management :-

▪ Test plan, test scenario and test cases will be under configuration management control and kept on the local system.
▪ Test reports will be stored on the same local system.
▪ Backup will be taken on a cloud platform.

Test Environment :-

Test environment will be made of individual machines.
▪ Windows 10 professional
▪ 4 GB RAM
▪ Quad-core processor, Core i5.

Test Entry Criteria :-

▪ All reviews and comments are closed.
▪ All unit testing defects are closed.
▪ Application is installed and launched.

Test Exit Criteria :-

Test cases are completed and all defects found in testing are closed, retested and found to be closed.

Test Suspension Criteria :-

Tests will be suspended in the following circumstances:-

▪ Test cases written do not find sufficient number of defects.
▪ Test case writing and test data definition are not completed.
▪ Application cannot be installed.
▪ Defects found in 1st/2nd/3rd iteration are more than 500.

Test Resumption Criteria :-

▪ All defects are reviewed and corrective/preventive actions are taken.
▪ All review comments are closed.
▪ All unit testing defects are fixed, retested and found to be closed.
▪ Application can be installed.

Tester’s Training :-

The following training is planned for testers:-
▪ Testers will be trained on basics of the order taking system in restaurants.
▪ Tester’s training for writing test plans, test scenarios, test cases, execution of test cases and defect logging is already done.

Communication :-

▪ Test team will be meeting every day in the morning for 15 minutes over online platform to discuss about previous day’s work and the current day’s planned work with any obstacles.
▪ Test team will be meeting with development team at the end of test iteration, over online platforms.

Tools :-

▪ Bugs are added in a common repository on the local system.
▪ Defect communication and build release communication will be through internal communication process

TEST SCENARIO

LOGIN PAGE

Common for All Test Scenarios (TS00.0—Priority P1) :-

The application can be launched by clicking an icon on the desktop. It can be launched from ‘Start’, ‘Programs’ and selecting the application from the list. There are two labels with captions ‘Username’, ‘Password’ one below another in the given order. Background colour of the label is light grey without border. There are two text boxes horizontally aligned to labels. Text boxes have white backgrounds and borders. There is also a login button. All Captions are centrally aligned; the font is black. Login button background is white.

Scenario 1 (TS00.1—Priority P1) Positive scenario where login is done successfully :-

On inserting the correct combination of username and password, the application windows minimizes and the menu window opens.

Successfully logged in to the application.

Scenario 2 (TS00.2—Priority P1) Negative scenario where username or password is empty :-

Negative scenario as the password and username cannot be validated in order to open the Main window.

Scenario 3 (TS00.3—Priority P1) Negative scenario where username is correct or password is empty or incorrect :-

Negative scenario as the password and username cannot be validated in order to open the Main window.

Scenario 4 (TS00.4—Priority P1) Negative scenario where username is incorrect or empty :-

Negative scenario as the password and username cannot be validated in order to open the Main window.

Scenario 5 (TS00.5—Priority P1) Negative scenario where username or password is incorrect:-

Negative scenario as the password and username cannot be validated in order to open the Main window.

TEST SCENARIO

FOR INVOICE GENERATION

Common for All Test Scenarios (TS00—Priority P1) :-

The application can be launched by clicking an icon on the desktop. It can be launched from 'Start', 'Programs' and selecting an application from the list. For larger resolution, the screen appears smaller, and is aligned to top and left side of the screen. Four main menu options appear on the screen along with the food items. Each item has its own radio-button associated with it which needs to be enabled to select the item. After selecting the item, other buttons 'Add To Order' and 'Make Bill' are present. The former will be used to add orders to the cart. On completion of the cart, the latter will prepare a statement of orders and make the final bill. The final bill will be printed in the respective output boxes. Along with that each food item also has quantity and size associated with it. So there are 4 size buttons and 4 quantity buttons for selecting the size and quantity of the food item. On only selecting particular items, some sub-menus will appear providing more add-ons to the items. There are 4 text-boxes called 'Total', 'Amount', 'Tax', 'Donate' to display final bill amounts of the transaction. Buttons have a Gray background and captions are in black font, and aligned at the centre of the box. Tab sequence is as follows, When application is launched, the cursor by default all fields are empty. For each click of the button on the menu, it moves to the text box(quantity) and combo-box(Size) of the particular item in the menu. When the user clicks the button 'Add To Order', the item is added to the cart and the final bill. When the user clicks the 'Make Bill', the final bill is generated. The user can use the left button of the mouse to go to any control directly. The clear button is used to clear all selections in the application.

Scenario 1 (TS01—Priority P1) Positive scenario where bill is calculated successfully :-

When the application is launched, user selects any items that user wants in the cart. On selecting the item, user then selects the quantity and size of the item and then clicks on AddToOrder button, and the item gets added in to the cart. User then selects MakeBill and the bill is generated. User is shown the receipt and total amount to be paid.
Example. User selects Tea, size = Small, Quantity = 1;
User selects add to order. The MakeBill button becomes active. On clicking it, the final bill and report is generated:-
Small Size Tea is ordered quantity 1 – 10 Rs.

You have to pay 10 Rs.

Scenario 2 (TS02—Priority P2) Negative scenario :-

There can be four combinations where equivalence partitioning can be applied. One of the four items, the quantity is taken as ‘0’ each time which may represent equivalence class.
Negative scenario as item cannot be added to order as quantity is ‘0’ and ‘Add to Order’ doesn’t work successfully.

Scenario 3 (TS03—Priority P3) :-

There can be six combinations where equivalence partitioning can be applied. Two of the four main items’ quantities are taken as ‘0’ each time which may represent equivalence class.
Negative scenario as items cannot be added to order as the quantities are ‘0’ and ‘Add to Order’ doesn’t work successfully.

Scenario 4 (TS04—Priority P3) :-

Negative scenario as the items cannot be added to order as all the quantities of the selected items are ‘0’ and ‘Add to Order’ doesn’t work successfully.

Scenario 5 (TS05—Priority P2) :-

There can be four combinations where equivalence partitioning can be applied. One of the four main items, the quantity is taken as negative each time which may represent equivalence class.
Negative scenario as the items cannot be added to order as the quantity of the selected item is negative and ‘Add to Order’ doesn’t work successfully.

Scenario 6 (TS06—Priority P3) :-

There can be six combinations where equivalence partitioning can be applied. Two of the four main sides are taken as negative each time which may represent equivalence class.
Negative scenario as the items cannot be added to order as the quantity of the selected items is negative and ‘Add to Order’ doesn’t work successfully.

Scenario 7 (TS07—Priority P3) :-

Negative scenario as the items cannot be added to order as all the quantities of the selected items are ‘negative’ and ‘Add to Order’ doesn’t work successfully.

Scenario 8 (TS08—Priority P2) :-

There can be four combinations where equivalence partitioning can be applied. One of the four main items, the quantity is taken as character value other than integers each time which may represent equivalence class.
Negative scenario as the items cannot be added to order as the quantity of the selected item is character value and 'Add to Order' doesn't work successfully.

Scenario 9 (TS09—Priority P3) :-
There can be six combinations where equivalence partitioning can be applied. Two of the four main sides are taken as character value other than integers each time which may represent equivalence class.
Negative scenario as the items cannot be added to order as the quantity of the selected items is character value and 'Add to Order' doesn't work successfully.

Scenario 10 (TS10—Priority P3) :-

Negative scenario as the items cannot be added to order as all the quantities of the selected items are character value other than integers and 'Add to Order' doesn't work successfully.

Scenario 11 (TS11—Priority P1) :-

Positive scenario where 'Make Bill' works successfully

Scenario 12 (TS09—Priority P3) :- There can be four combinations where equivalence partitioning can be applied. One of the four main items is added to the order and 'Make Bill' button is used.

Positive scenario where 'Make Bill' works successfully

Scenario 13 (TS13—Priority P3) :-

There can be six combinations where equivalence partitioning can be applied. Two of the four main items are added to the order and 'Make Bill button is used.
Positive scenario where 'Make Button' works successfully

Scenario 14 (TS14—Priority P3) :-

All items are added to order and 'Make Bill' button is used.
Positive scenario where 'Make Bill' works successfully.

Scenario 15 (TS15—Priority P3) :-

All items are added to order and 'Make Bill' button is used.
Clear button is pressed for clearing the previous order.

The Clear button works successfully.

Definitions at test suit level :-

Project Name :- Invoice Generation System.
Test Suite ID and Name :- ST-01 System testing test suit.
Version Date, Number, Author of Test Case :- 01.00.000 28/08/2020
Mehul I., Abhay N.
Environment Prerequisites :- Windows 10 professional, Core i5, Quad-core processor, 4 GB RAM.

TEST CASES

Test case 1

Test Precondition:- Application must be installed and Computer must be working.
Test Sequence:- NA
Test Scenario Traceability:- TS00.0
Test Case Name and Number:- ST/0001
Type of Testing:- Smoke Testing.
Objectives:- To check whether app can be launched or not.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- Launching an application by clicking an icon on desktop. (This is considered as equivalent to launching application from 'Start', 'Programs' and selecting the application from the list)
Steps to Reproduce:- Start computer and Click the icon on desktop
Expected Results:- Application is launched (Screen is seen with the login of restaurant).

Test case 2

Test Precondition:- Application is launched
Test Sequence:- ST/0001
Test Scenario Traceability :- TS00.0
Test Case Name and Number :- UI/0001
Type of Testing :- User Interface testing
Objectives :- To check that application opens and occupies the screen
Priority :- P1
Valid/Invalid Conditions :- Valid condition
Test Data :- Not applicable
Steps to Reproduce:- Start computer Click the icon on desktop
Expected Results :- Complete screen is occupied by the layout mentioned. No scroll bar appears.

Test case 3

Test Precondition:- Application opened.
Test Sequence:- UI/0001
Test Scenario Traceability:- TS00.1
Test Case Name and Number:- UI/0001
Type of Testing:- Functional Testing.
Objectives:- To login in the application.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- correct Username and Password
Expected Results:- Application is launched (Screen is seen with the menu of restaurant).

Test case 4

Test Precondition:- Application opened.
Test Sequence:- UI/0001
Test Scenario Traceability:- TS00.2
Test Case Name and Number:- FUN/0002
Type of Testing:- Functional Testing.
Objectives:- To login in the application.
Valid/invalid Conditions:- InValid
Priority:- P1
Test Data:- incorrect Username and Password combination.
Expected Results:- Application is not launched.

Test case 5

Test Precondition:- Application is launched and logged in
Test Sequence:- UI/0001
Test Scenario Traceability:- TS00
Test Case Name and Number:- UI/0002
Type of Testing:- User interface testing
Objectives:- To check that menu is shown on opening of the application.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- NA
Steps to Reproduce:- Launch App
Expected Results:- Menu of the restaurant

<u>Test case 6</u>

Test Precondition:- App is launched
Test Sequence:- UI/0002
Test Scenario Traceability:- TS00
Test Case Name and Number:- UI/0003
Type of Testing:-User interface testing
Objectives:- To check that 'Add to order' button and 'Make bill' button appears with the Make Bill button as false.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- NA
Steps to Reproduce:- Launch App
Expected Results:- Add to Order and Make bill buttons are visible and Make Bill is not accessible.

<u>Test case 7</u>

Test Precondition:- App is launched.
Test Sequence:- UI/0003
Test Scenario Traceability:- TS01
Test Case Name and Number:- FUN/0001
Type of Testing:- Functional Testing
Objectives:- To check whether radio buttons representing the items can be selected.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- Select 'Tea' from Drinks tab. (This is considered as equivalent to all valid entries in radiobutton. This is input equivalence partitioning)
Steps to Reproduce:- Select 'Tea' from Drinks tab.
Expected Results:- Tea is selected.

<u>Test case 8</u>

Test Precondition:- App is launched
Tea is selected

Test Sequence:- FUN/0001
Test Scenario Traceability:- TS01
Test Case Name and Number:- FUN/0002
Type of Testing:- Functional Testing
Objectives:- To check whether the quantity button works for Drinks.
Valid/invalid Conditions:- Valid
Priority:-P1
Test Data:- Enter '2' in quantity tab. (This is considered as equivalent to all valid entries in text box. This is input equivalence partitioning)
Steps to Reproduce:- Enter 2 as the required quantity from keyboard.
Expected Results:- 2 cups of tea will be selected.

<u>Test case 9</u>

Test Precondition:- 2 quantity of tea is selected.
Test Sequence:- FUN/0002
Test Scenario Traceability:- TS01
Test Case Name and Number:- FUN/0003
Type of Testing:- Functional Testing
Objectives:- To check if the size button works for drinks.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- Select 'medium' size for tea (This is considered as equivalent to all valid entries in text box. This is input equivalence partitioning).
Steps to Reproduce:- Select Medium from combo-box.
Expected Results:- 2 quantity or tea with medium size are selected.

<u>Test case 10</u>

Test Precondition:- Tea , quantity as 2, size as medium.
Test Sequence:-FUN/0003
Test Scenario Traceability:- TS01
Test Case Name and Number:- FUN/0004
Type of Testing:- Functional Testing
Objectives:- To check if 'Add To Order' button works properly.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- NA
Steps to Reproduce:- Left-click by mouse on Add to Order button, at bottom of the page.

Expected Results:- Items are added to the order and the Make bill button is enabled.

Test case 11

Test Precondition:- Add to Order is clicked.
Test Sequence:- FUN/0004
Test Scenario Traceability:- TS01
Test Case Name and Number:- FUN/0005
Type of Testing:- Functional Testing.
Objectives:- To check whether Make Bill button is enabled and works.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- NA
Steps to Reproduce:- Click on Make bill button using Left-mouse button.
Expected Results:- Bill Is generated.

Test case 12

Test Precondition:- Make Bill button is pressed
Test Sequence:- FUN/0005
Test Scenario Traceability:- TS01
Test Case Name and Number:- FUN/0006
Type of Testing:- Functional testing
Objectives:- To check if control can go to bill tab by clicking Make bill button.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- NA
Steps to Reproduce:- NA
Expected Results:-A report of all the orders along with final bill, including all taxes.

Test case 13

Test Precondition:- App is launched and menu is loaded.
Test Sequence:- UI/0001
Test Scenario Traceability:- TS02
Test Case Name and Number:- FUN/NEG_TS02/0001
Type of Testing:- Functional Testing

Objectives:- To check if app works in case of not selecting any item from the menu.
Valid/invalid Conditions:- Invalid
Priority:- P2
Test Data:-NA
Steps to Reproduce:- Click add to order without selecting any item.
Expected Results:- No changes are to seen in the app.

Test case 14

Test Precondition:- App is launched and menu is loaded.
Test Sequence:- UI/0001
Test Scenario Traceability:- TS02
Test Case Name and Number:- FUN/NEG_TS02/0002
Type of Testing:- Functional Testing
Objectives:- To check if item is selected but quantity is 0.
Valid/invalid Conditions:- Invalid
Priority:- P2
Test Data:- The quantity of Item is '0'. (This is considered as equivalent to all 0 entries in text box. This is input equivalence partitioning).
Steps to Reproduce:- Enter the value 0 in quantity tab.
Expected Results:- Pop-up message denoting to enter an integer

----

Test case 15

Test Precondition:- App is launched and menu is loaded.
Test Sequence:- UI/0001
Test Scenario Traceability:-TS05
Test Case Name and Number:- FUN/NEG_TS05/0001
Type of Testing:- Functional Testing
Objectives:- To check if quantity entered is a negative number.
Valid/invalid Conditions:- Invalid
Priority:- P2
Test Data:- A negative number in quantity tab. (This is considered as equivalent to all negative entries in text box. This is input equivalence partitioning).
Steps to Reproduce:- Enter a negative number in the quantity tab
Expected Results:- Pop-up message denoting to enter an integer

----

Test case 16

Test Precondition:- App is launched and menu is loaded.
Test Sequence:- UI/0001
Test Scenario Traceability:- TS08
Test Case Name and Number:- FUN/NEG_TS08/0001
Type of Testing:- Functional Testing
Objectives:- To check if quantity entered is a character value.
Valid/invalid Conditions:- Invalid
Priority:- P2
Test Data:-Enter an 'p' in quantity tab. . (This is considered as equivalent to all character entries in text box. This is input equivalence partitioning).
Steps to Reproduce:- Enter a char p in quantity tab.
Expected Results:- Pop-up message denoting to enter an integer

----

<u>Test case 17</u>

Test Precondition:- Bill is generated.
Test Sequence:- FUN/0006
Test Scenario Traceability:- TS15
Test Case Name and Number:- FUN/CLR/0001
Type of Testing:- Functional Testing
Objectives:- Clear the screen of the previous order.
Valid/invalid Conditions:- Valid
Priority:- P3
Test Data:- NA
Steps to Reproduce:- Left-click on clear button.
Expected Results:- The entire menu is cleared.

<u>Test case 18</u>

Test Precondition:- Bill is generated
Test Sequence:- FUN/0006
Test Scenario Traceability:- TS16
Test Case Name and Number:- FUN/SAVE/0001
Type of Testing:- Functional testing
Objectives:- To save the log of orders into a file on the local system.
Valid/invalid Conditions:- Valid

Priority:- P2
Test Data:- NA
Steps to Reproduce:- Click on the save button.
Expected Results:- Pop up message indicating successful save.

<u>Test case 19</u>

Test Precondition:- Bill is generated
Test Sequence:- FUN/0006
Test Scenario Traceability:- TS16
Test Case Name and Number:- FUN/SAVE/0002
Type of Testing:- Functional testing
Objectives:- To save the log of orders into a file on the local system.
Valid/invalid Conditions:- Invalid
Priority:- P2
Test Data:- NA
Steps to Reproduce:- Click on the save button.
Expected Results:- Pop up message indicating unsuccessful save.

<u>Test case 20</u>

Test Precondition:- App is launched
Test Sequence:- UI/0001
Test Scenario Traceability:- TS00
Test Case Name and Number:- FUN/ALL/0001
Type of Testing:- Functional testing
Objectives:- To check system behaviour if all fields are selected.
Valid/invalid Conditions:- Valid
Priority:- P1
Test Data:- All buttons and Fields are filled and selected
Steps to Reproduce:- Click on all radio buttons and select size and quantities and click Add to order.
Expected Results:- Successfully added to order.

JUNIT TEST CASES

Unit tests were created as JUnit tests. Not all classes were tested as there were lots of user classes. It was thought to be unnecessary to test those classes because overall functionality is similar. The tested classes, the functions of classes and JUnit test are listed in the table below. JUnit tests were executed using Eclipse and JUnit version 5.

The JUnit tests were run once, discovered problems fixed and then ran again to detect any new problems that might have appeared. No problems were found, after the bugs were fixed.
A detailed explanation of the bugs and their fixes is also discussed below.

Table :- Tested classes.

FILE	TEST CASE
Beverage.class	Beverage_Test.class
Tea.class	Tea_Test.class
Main_Page.class	Main_Page_Test.class
Login.class	Login_Test.class

Table :- Tested functions.

Test Class Name	Test Function
Beverage_Test.class	Object_create_check()
	set_price_check()
	get_amount_check()
	get_size_check()
	message_check()
Tea_Test.class	Object_create_check()

	set_price_check()
	get_amount_check()
	get_size_check()
	message_check()

Main_Page_Test.class	Make_Bill_check()
	Drinks_Quantity_check()
	Fries_Quantity_check()
	Dessert_Quantity_check()
	SW_Quantity_check()
	Clear_check()

JUNIT Test cases:-

Test case 1 – Beverage class assigning values to variables
DESCRIPTION :- This JUnit test case was run to ensure that the variables are assigned values correctly on calling the respective functions.
ERRORS FOUND :- No errors or failures were collected.

Test case 2 – Beverage class returning string message
DESCRIPTION :- This JUnit test case was run to ensure that the class successfully return a string showing the item ordered.
ERRORS FOUND :- No errors or failures were collected.

Test case 3 – Beverage class returning values
DESCRIPTION :- This JUnit test case was run to ensure that the class is correctly returning values on calling respective functions.

ERRORS FOUND :- No errors or failures were collected.

Test case 4 – Make Tea as a child class of Tea (similar for rest of the items).
DESCRIPTION :- This JUnit test case was run to ensure that proper derived class Tea creates an object if tea is selected and inheriting qualities of Beverage class successfully.
ERRORS FOUND :- No errors or failures were collected.

Test case 5 – TeaFinal Price Check
DESCRIPTION :- This JUnit test case was run to ensure that on making various additions to tea order, the final price is set correctly.
ERRORS FOUND :- No errors or failures were collected.

Test case 6 – User Interface visualize
DESCRIPTION :- This JUnit test case was run to ensure that the UI is loaded correctly on successful login.
ERRORS FOUND :- BUG 1 was found. It was corrected in the next run.

Test case 7 – Radio Button action check
DESCRIPTION :- This JUnit test case was run to ensure that the radio buttons depicting the items are correctly storing the selected item.
ERRORS FOUND :- BUG 2 was found. It was corrected in the next run.

Test case 8 – Quantity and Size check
DESCRIPTION :- This JUnit test case was run to ensure that the size and quantity field correctly obtained data from the user and was set to null after every addition to order.
ERRORS FOUND :- BUG 3 was found and was corrected in the next run.

Test case 9 – Make Bill button check

DESCRIPTION :-This JUnit test case was run to ensure that the Make Bill button works and is only enabled if there is at least one item in the order list added to order.
ERRORS FOUND :- BUG 4 was found. Later it was corrected.

Test case 10 – Clear Button check
DESCRIPTION :- This JUnit test case was run to ensure that the clear button functions properly and on clicking that clears the menu and also sets Make Bill button invalid.
ERRORS FOUND :- BUG 5 was found. Later it was corrected

<u>FOUND AND FIXED BUGS</u>

In this section, the bugs we found during the first execution of test cases are listed. Also the fixes for bugs are listed. All bugs listed here are fixed.

<u>BUG 1</u>
Description :- When the App was launched, the Order window was not appearing after the login screen. The app just ended abruptly.

Fix :- It was made sure that the object of the order page was created correctly and was called correctly by the login page function.

<u>BUG 2</u>
Description :- When items were added to cart, it was possible to select more than one items simultaneously of the same type. Thus there was ambiguity on the quantity and size of item.

Fix :- At one time only one sub-item of a particular Cuisine could be selected, thus removing the ambiguity. It was solved by grouping the radio buttons into a group of main items.

<u>BUG 3</u>

Description :- Quantity text field is empty by default. Hence when an item is selected and if it is added to order without adding the quantity, the app gives garbage results.

Fix :- Added a condition to check if the quantity entered is not null or 0.

<u>BUG 4</u>

Description :- When the App was launched, the Make Bill button was enabled. It enabled the user to click on the Make Bill without adding the items to order. It makes the app function improperly and then crash.

Fix :- The Make Bill button was enabled only on adding at least one item in the order by adding a condition to check the order cart.

<u>BUG 5</u>

Description :- On clicking the clear button after the bill was generated to take new order, only the bill fields were cleared but the initial order values were still present in the variables, which resulted in incorrect bill generation.

Fix :- The used variables were again set to a default state after every transaction.

JUNIT TEST CASES

TeaClass Test:-

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;
class TeaCheck {

    @Test
    void Object_create_check() {
        Beverage bvg = new Tea("Large", 2, null);
    }

    @Test
    void set_price_check() {
        Beverage bvg = new Tea("Large", 2, null);
        assertEquals(5.0, bvg.getPrice());
    }

    @Test
    void get_amount_check() {
        Beverage bvg = new Tea("Large", 2, null);
        assertEquals(2, bvg.getAmount());
    }

    @Test
    void get_size_check() {
        Beverage bvg = new Tea("Large", 2, null);
        assertEquals("Large", bvg.getSize());
    }

    @Test
    void message_check() {
        Beverage bvg = new Tea("Large", 2, null);
        assertEquals("2 Large quantity of Tea", bvg.toString());
    }



}
```

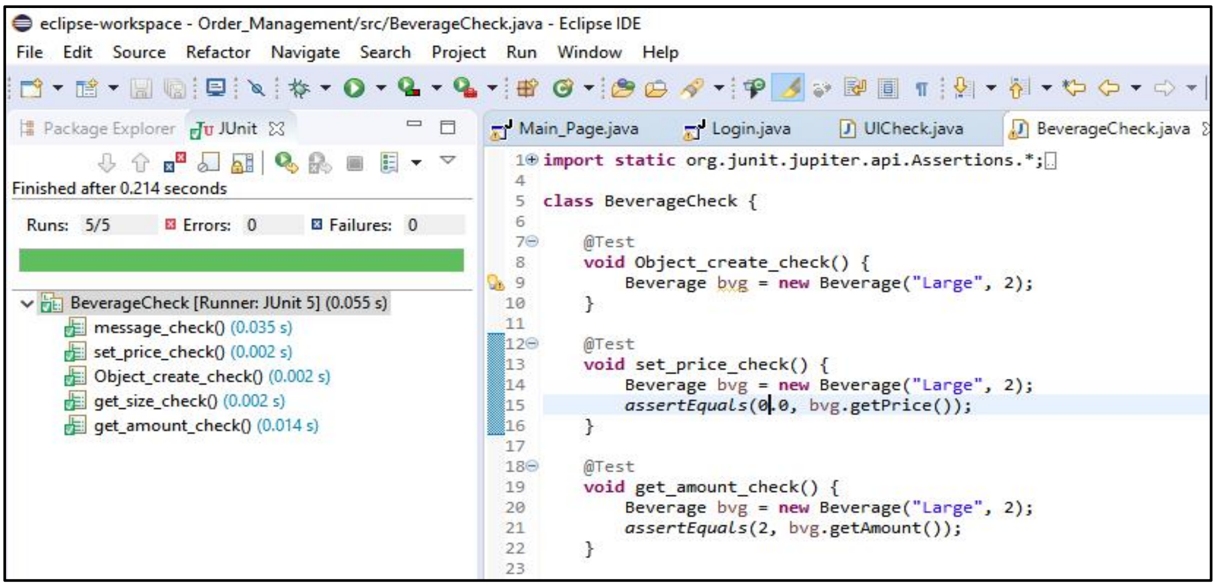
BeverageClass Test :-

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class BeverageCheck {

    @Test
    void Object_create_check() {
        Beverage bvg = new Beverage("Large", 2);
    }

    @Test
    void set_price_check() {
        Beverage bvg = new Beverage("Large", 2);
        assertEquals(0.0, bvg.getPrice());
    }

    @Test
    void get_amount_check() {
        Beverage bvg = new Beverage("Large", 2);
        assertEquals(2, bvg.getAmount());
    }

    @Test
    void get_size_check() {
        Beverage bvg = new Beverage("Large", 2);
        assertEquals("Large", bvg.getSize());
    }

    @Test
    void message_check() {
        Beverage bvg = new Beverage("Large", 2);
        assertEquals("2 Large quantity of ", bvg.toString());
    }

}
```

OrderPage Test :-

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class UICheck {

    Main_Page p = new Main_Page();


    @Test
    void Make_Bill_check()
    {
        assertEquals(false, Main_Page.btnMakeBill.isEnabled());
    }

    @Test
    void Drinks_Quantity_check()
    {
        p.initialize();
        assertNotEquals(0, Main_Page.drinkquan);
    }

    @Test
    void Fries_Quantity_check()
    {
        assertNotEquals(0, Main_Page.friesquan);
    }

    @Test
    void SW_Quantity_check()
    {
        assertNotEquals(0, Main_Page.swquan);
    }

    @Test
    void Dessert_Quantity_check()
    {
        assertNotEquals(0, Main_Page.dessertquan);
    }

    @Test
    void Clear_check()
    {
        assertEquals(null, Main_Page.Drinksgroup.getSelection());
        assertEquals(null, Main_Page.Friesgroup.getSelection());
        assertEquals(null, Main_Page.SWgroup.getSelection());
        assertEquals(null, Main_Page.Dessertgroup.getSelection());
    }

}
```

Screenshots from Eclipses JUnit test panel

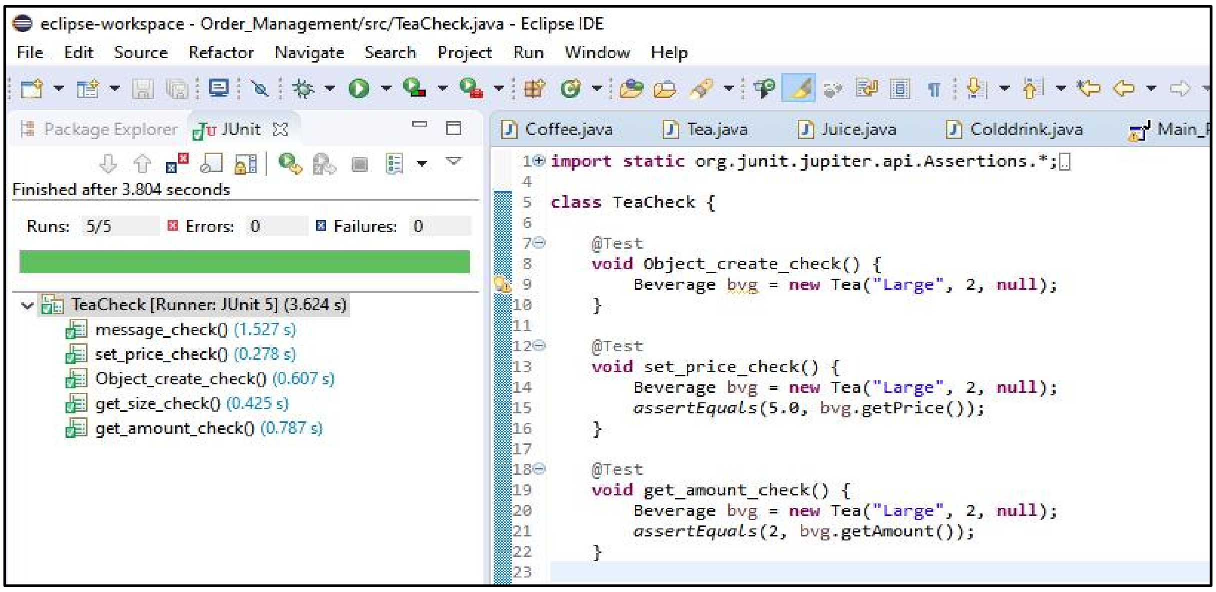

eclipse-workspace - Order_Management/src/BeverageCheck.java - Eclipse IDE

Finished after 0.214 seconds

Runs: 5/5 Errors: 0 Failures: 0

```
BeverageCheck [Runner: JUnit 5] (0.055 s)
    message_check() (0.035 s)
    set_price_check() (0.002 s)
    Object_create_check() (0.002 s)
    get_size_check() (0.002 s)
    get_amount_check() (0.014 s)
```

```
import static org.junit.jupiter.api.Assertions.*;

class BeverageCheck {

    @Test
    void Object_create_check() {
        Beverage bvg = new Beverage("Large", 2);
    }

    @Test
    void set_price_check() {
        Beverage bvg = new Beverage("Large", 2);
        assertEquals(0.0, bvg.getPrice());
    }

    @Test
    void get_amount_check() {
        Beverage bvg = new Beverage("Large", 2);
        assertEquals(2, bvg.getAmount());
    }
```

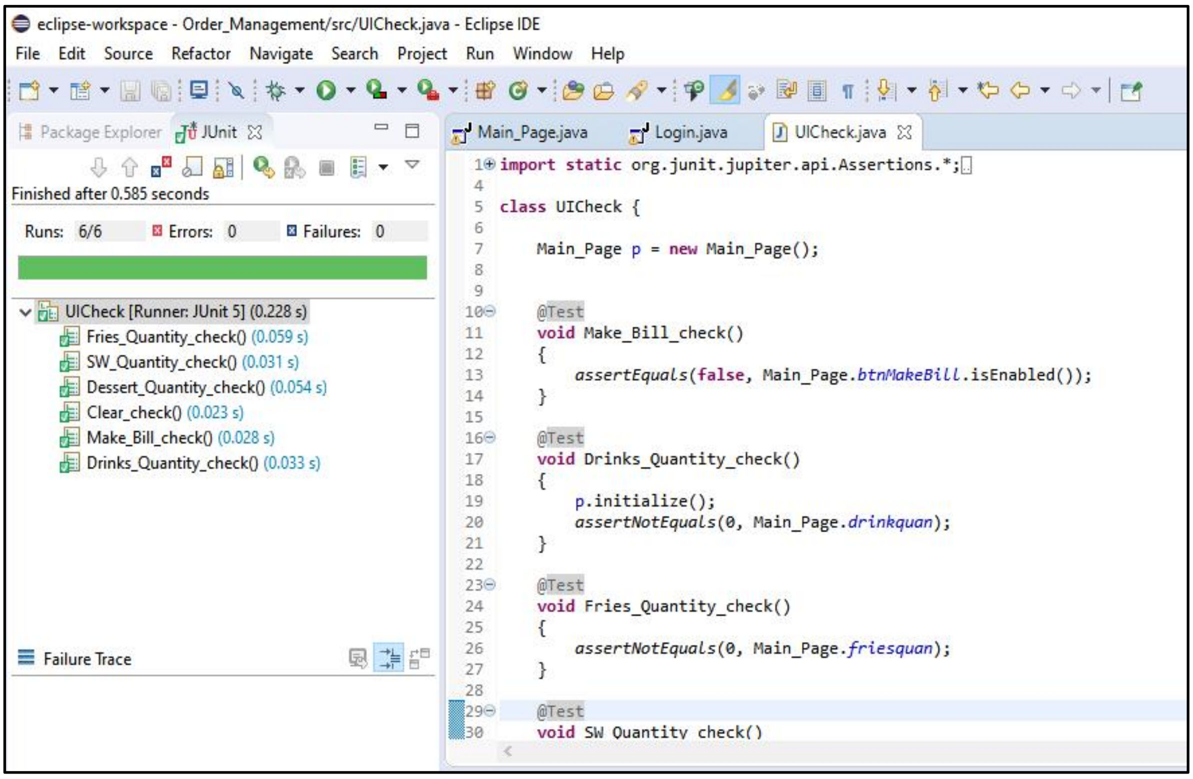

eclipse-workspace - Order_Management/src/TeaCheck.java - Eclipse IDE

Finished after 3.804 seconds

Runs: 5/5 Errors: 0 Failures: 0

```
TeaCheck [Runner: JUnit 5] (3.624 s)
    message_check() (1.527 s)
    set_price_check() (0.278 s)
    Object_create_check() (0.607 s)
    get_size_check() (0.425 s)
    get_amount_check() (0.787 s)
```

```
import static org.junit.jupiter.api.Assertions.*;

class TeaCheck {

    @Test
    void Object_create_check() {
        Beverage bvg = new Tea("Large", 2, null);
    }

    @Test
    void set_price_check() {
        Beverage bvg = new Tea("Large", 2, null);
        assertEquals(5.0, bvg.getPrice());
    }

    @Test
    void get_amount_check() {
        Beverage bvg = new Tea("Large", 2, null);
        assertEquals(2, bvg.getAmount());
    }
```


eclipse-workspace - Order_Management/src/UICheck.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Main_Page.java
Login.java
UICheck.java
Failure Trace

Finished after 0.585 seconds

Runs: 6/6 Errors: 0 Failures: 0

```
UICheck [Runner: JUnit 5] (0.228 s)
    Fries_Quantity_check() (0.059 s)
    SW_Quantity_check() (0.031 s)
    Dessert_Quantity_check() (0.054 s)
    Clear_check() (0.023 s)
    Make_Bill_check() (0.028 s)
    Drinks_Quantity_check() (0.033 s)
```

```java
import static org.junit.jupiter.api.Assertions.*;

class UICheck {

    Main_Page p = new Main_Page();


    @Test
    void Make_Bill_check()
    {
        assertEquals(false, Main_Page.btnMakeBill.isEnabled());
    }

    @Test
    void Drinks_Quantity_check()
    {
        p.initialize();
        assertNotEquals(0, Main_Page.drinkquan);
    }

    @Test
    void Fries_Quantity_check()
    {
        assertNotEquals(0, Main_Page.friesquan);
    }

    @Test
    void SW_Quantity_check()
```

GITHUB LINK FOR THE WORKING PROJECT

https://github.com/A3r4Xa5/Manual-Testing-Invoice-Generator