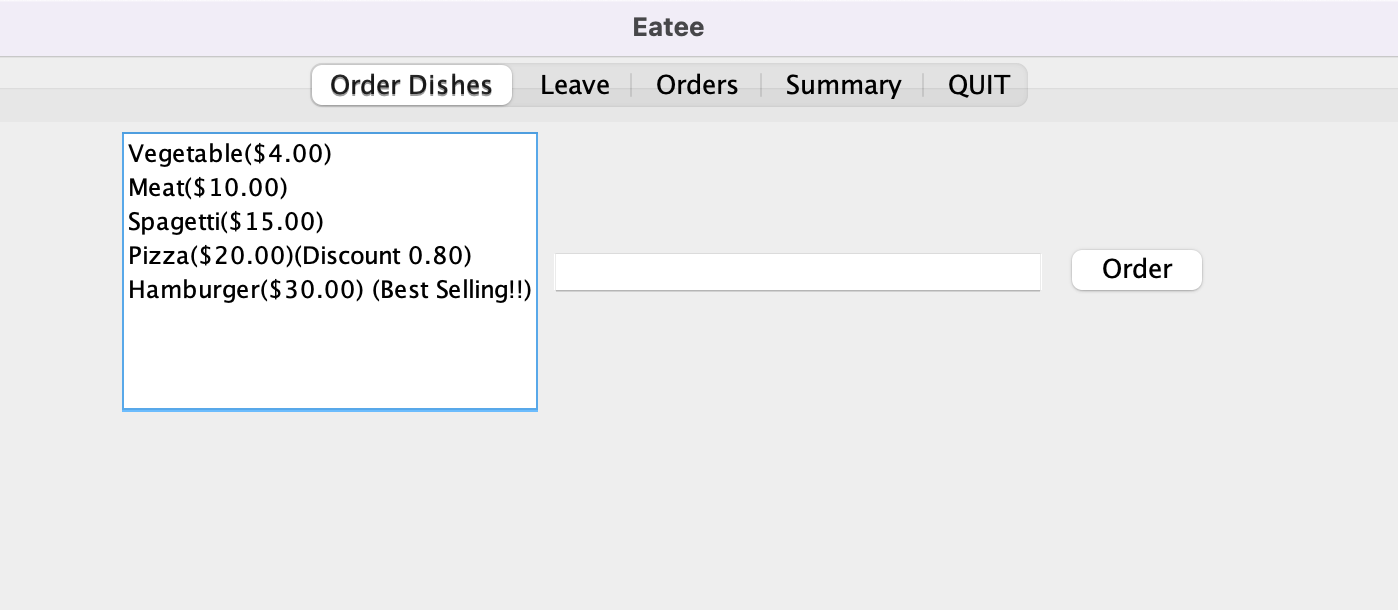
**The GUI Design**

Here is the description of my topic: the topic of my project is restaurant. I open a restaurant called “**Eatee**”. The restaurant has following usecases:

* The restaurant can show a menu of dishes to customers, then customers can order the dishes at the restaurant.
* After customers order the dish, restaurant will add the **order** to the **orders** collection, then it can show the order content of all customers.
* After customers finish eating, they can leave the restaurant, then restaurant will remove the **order** from **orders** collection, at the same time, it will record the order in another collections.
* The restaurant can show the summary of all finished orders, print the content of each order and the total price of all orders.

The GUI design is tab-based, the layout is as follows, all functions are displayed as tabs.



Users can order dishes using multiple selection list, and view current orders in “Orders” tab, view summary in “Summary” tab, it can also leave a customer using “Leave” tab which also uses multiple selection list. The detailed function explanation is in **Explanation** part.

**The code:**

/\*\*

\* Dish of the restaurant

\* Each dish has name and price

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

public class Dish {

private String name;

private double price;

/\*\*

\* Constructor of Dish

\*/

public Dish(String name, double price) {

this.name = name;

this.price = price;

}

/\*\*

\* Name of dish

\*

\* @return dish name

\*/

public String getName() {

return name;

}

/\*\*

\* Price of dish

\*

\* @return dish

\*/

public double getPrice() {

return price;

}

/\*\*

\* Dish representation

\*/

@Override

public String toString() {

return String.format("%s($%.2f)", name, price);

}

}

/\*\*

\* Class represents Hamburger

\*

\*/

public class Hamburger extends Dish {

/\*\*

\* Constructor for Hamburger

\*

\* @param price price of hamburger

\*/

public Hamburger(double price) {

super("Hamburger", price);

}

@Override

public String toString() {

return super.toString() + " (Best Selling!!)";

}

}

/\*\*

\* Pizza dish

\*

\*/

public class Pizza extends Dish {

// Discount of pizza

private double discount;

/\*\*

\*

\* Constructor for Pizza

\*

\* @param price price of Pizza

\* @param discount discount of pizza

\*/

public Pizza(double price) {

super("Pizza", price);

// Pizza is selled at 80% discount by default

this.discount = 0.8;

}

/\*\*

\*

\* Constructor for Pizza

\*

\* @param price price of Pizza

\* @param discount discount of pizza

\*/

public Pizza(double price, double discount) {

super("Pizza", price);

this.discount = discount;

}

/\*\*

\* Get price of pizza, it always has a discount

\*/

@Override

public double getPrice() {

return super.getPrice() \* discount;

}

@Override

public String toString() {

return super.toString() + String.format("(Discount %.2f)", discount);

}

}

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Order in restaurant

\*

\*/

public class Order {

/\*\*

\* Dishes of this order

\*/

private final List<Dish> dishes;

/\*\*

\* Customer who ordered this

\*/

private final String customer;

public Order(String customer) {

this.customer = customer;

dishes = new ArrayList<>();

}

/\*\*

\* Add dish to the order

\*

\* @param dish dish

\*/

public void addDish(Dish dish) {

dishes.add(dish);

}

/\*\*

\* Return the customer who ordered this order

\*

\* @return customer name

\*/

public String getCustomer() {

return customer;

}

/\*\*

\* Total price of this order

\*

\* @return total price

\*/

public double totalPrice() {

double sum = 0;

for (Dish dish : dishes) {

// Each dish has its price

sum += dish.getPrice();

}

return sum;

}

/\*\*

\* Return string representation of order

\*/

@Override

public String toString() {

StringBuilder result = new StringBuilder(String.format("Customer %s ordered:\n", customer));

for (Dish dish : dishes) {

result.append("\t").append(dish).append("\n");

}

return result.toString();

}

}

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

/\*\*

\* Restaurant

\*/

public class Restaurant {

// Name of Restaurant

private final String name;

// All dishes that restaurant have

private final List<Dish> dishes;

// All current orders

private final List<Order> orders;

// All finished orders

private final List<Order> finishedOrders;

private final Set<String> customers;

private final MyObserver myObserver = new MyObserver();

public void registerObserver(Eatee eatee) {

myObserver.registerObserver(eatee);

}

/\*\*

\* Constructor for Restaurant

\*/

public Restaurant(String name) {

this.name = name;

// Initial all dishes that restaurant have

dishes = new ArrayList<>();

dishes.add(new Dish("Vegetable", 4));

dishes.add(new Dish("Meat", 10));

dishes.add(new Dish("Spagetti", 15));

dishes.add(new Pizza(20));

dishes.add(new Hamburger(30));

orders = new ArrayList<>();

finishedOrders = new ArrayList<>();

customers = new HashSet<>();

}

/\*\*

\* Show menu of restaurant

\*/

public String getMenus() {

StringBuilder str = new StringBuilder();

str.append("Here are menus:\n");

for (int i = 0; i < dishes.size(); ++i) {

str.append(i + 1).append(". ").append(dishes.get(i))

.append("\n");

}

str.append("\n");

return str.toString();

}

public List<Dish> getDishes() {

return dishes;

}

public Set<String> getCustomers() {

return customers;

}

/\*\*

\* Get dish from restaurant

\*

\* @param index index of dish

\* @return dish in restaurant

\*/

public Dish getDish(int index) {

if (index <= 0 || index > dishes.size()) {

// No such dish

System.out.println("No such dish");

return null;

}

return dishes.get(index - 1);

}

/\*\*

\* Add an order

\*

\* @param order new order from customer

\*/

public void addOrder(Order order) {

orders.add(order);

customers.add(order.getCustomer());

myObserver.update();

}

/\*\*

\* A customer leave

\*

\* @param customer customer who leaves

\*/

public void leave(String customer) {

List<Order> removeLists = new ArrayList<>();

for (Order order : orders) {

if (customer.equals(order.getCustomer())) {

removeLists.add(order);

}

}

if (removeLists.isEmpty()) {

return;

}

for (Order order : removeLists) {

// Remove from current order

orders.remove(order);

// The order finished

finishedOrders.add(order);

}

customers.remove(customer);

myObserver.update();

}

/\*\*

\* Show current orders of restaurant

\*/

public String getOrders() {

if (orders.isEmpty()) {

return "No orders.";

}

StringBuilder str = new StringBuilder();

str.append("Here are orders:\n");

for (int i = 0; i < orders.size(); ++i) {

Order order = orders.get(i);

str.append(i + 1).append(". ").append(order).append("\n");

}

str.append("\n");

return str.toString();

}

/\*\*

\* Show the summary of restaurant

\*/

public String getSummary() {

StringBuilder str = new StringBuilder();

str.append("Here are summary:\n");

double sells = 0.0;

for (int i = 0; i < finishedOrders.size(); ++i) {

Order order = finishedOrders.get(i);

str.append(i + 1).append(". ").append(order).append("\n");

sells += order.totalPrice();

}

str.append("\n").append("Total sells: ").append(sells).append("\n");

return str.toString();

}

/\*\*

\* Return the name of restaurant

\*/

@Override

public String toString() {

return name;

}

}

public class MyObserver {

private Eatee eatee;

public void registerObserver(Eatee eatee) {

this.eatee = eatee;

}

public void update() {

// update views

eatee.updateViews();

}

}

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Set;

import java.util.stream.Collectors;

public class Eatee {

private final List<MenuItem> menus;

private final Restaurant restaurant;

private final List<JPanel> panels;

private final JTabbedPane jTabbedPane;

public Eatee() {

this.restaurant = new Restaurant("Eatee");

this.menus = Arrays.stream(MenuItem.values())

.filter(menuItemEnum -> menuItemEnum != MenuItem.INVALID)

.collect(Collectors.toList());

this.panels = new ArrayList<>();

this.restaurant.registerObserver(this);

for (int i = 0; i < this.menus.size(); ++i) {

this.panels.add(new JPanel());

}

jTabbedPane = new JTabbedPane();

for (int i = 0; i < menus.size(); ++i) {

MenuItem name = menus.get(i);

JPanel panel = panels.get(i);

jTabbedPane.addTab(name.getMenu(), panel);

updatePanels(name, panel);

}

}

private void createUIComponents() {

JFrame.setDefaultLookAndFeelDecorated(true);

JFrame frame = new JFrame("Eatee");

frame.setSize(1000, 700);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(jTabbedPane);

frame.setVisible(true);

}

public void updatePanels(MenuItem name, JPanel panel) {

switch (name) {

case ORDER\_DISHES -> updateOrderDishes(panel);

case SHOW\_ORDERS -> updateShowOrders(panel);

case LEAVE -> updateLeave(panel);

case SUMMARY -> updateSummary(panel);

case QUIT -> updateQuit(panel);

default -> {}

}

}

public void updateViews() {

for (int i = 0; i < menus.size(); ++i) {

MenuItem name = menus.get(i);

JPanel panel = panels.get(i);

updatePanels(name, panel);

}

jTabbedPane.updateUI();

jTabbedPane.repaint();

}

private void updateOrderDishes(JPanel panel) {

panel.removeAll();

List<Dish> dishes = restaurant.getDishes();

final DefaultListModel<String> dishList = new DefaultListModel<>();

for (Dish dish: dishes) {

dishList.addElement(dish.toString());

}

JTextField customerTextField = new JTextField(20);

JList<String> list = new JList<>(dishList);

JScrollPane scrollPane = new JScrollPane(list);

JButton orderButton = new JButton("Order");

orderButton.addActionListener((event) -> {

if (list.getSelectedIndex() != -1 && customerTextField.getText() != null

&& !customerTextField.getText().equals("")) {

Order order = new Order(customerTextField.getText());

for (String dishString: list.getSelectedValuesList()) {

for (Dish dish: dishes) {

if (dishString.equals(dish.toString())) {

order.addDish(dish);

}

}

}

restaurant.addOrder(order);

}

});

panel.add(scrollPane);

panel.add(customerTextField);

panel.add(orderButton);

}

private void updateLeave(JPanel panel) {

panel.removeAll();

Set<String> customers = restaurant.getCustomers();

final DefaultListModel<String> customerList = new DefaultListModel<>();

for (String customer: customers) {

customerList.addElement(customer);

}

JList<String> list = new JList<>(customerList);

JScrollPane scrollPane = new JScrollPane(list);

JButton orderButton = new JButton("Leave");

orderButton.addActionListener((event) -> {

if (list.getSelectedIndex() != -1) {

for (String customer: list.getSelectedValuesList()) {

restaurant.leave(customer);

}

}

});

panel.add(scrollPane);

panel.add(orderButton);

}

public void updateShowOrders(JPanel panel) {

panel.removeAll();

TextArea textArea = new TextArea(restaurant.getOrders(), 20, 0, TextArea.SCROLLBARS\_VERTICAL\_ONLY);

textArea.setEditable(false);

panel.add(textArea);

}

public void updateSummary(JPanel panel) {

panel.removeAll();

TextArea textArea = new TextArea(restaurant.getSummary(), 20, 0, TextArea.SCROLLBARS\_VERTICAL\_ONLY);

textArea.setEditable(false);

panel.add(textArea);

}

public void updateQuit(JPanel panel) {

panel.removeAll();

JButton button = new JButton("Quit");

button.addActionListener((event) -> System.exit(0));

panel.add(button);

}

public static void main(String[] args) {

javax.swing.SwingUtilities.invokeLater(new Eatee()::createUIComponents);

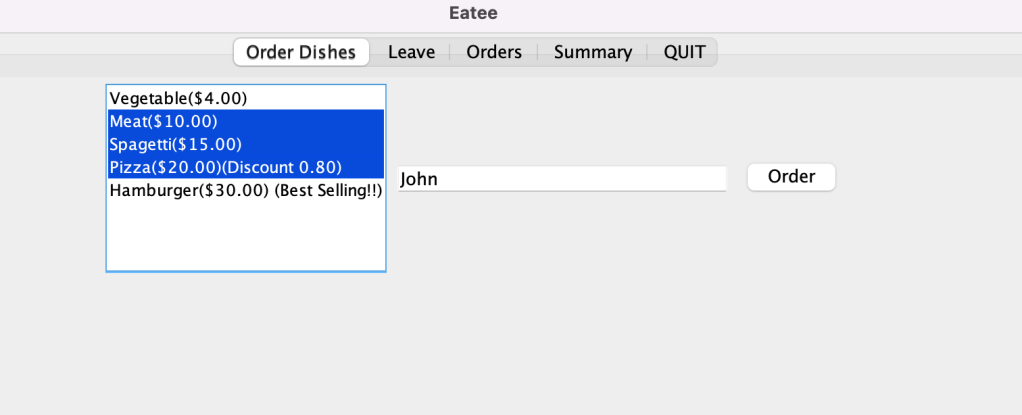
}

}

**The explanation:**

The **Eatee** class is the GUI components, it has a field restaurant (**Restaurant** class) which implements all functions of my restaurant. When **Eatee** initialized, it also initialized a new **Restaurant** instance, then call the **registerObserver** method of **Restaurant** to register current **Eatee** instance to **Restaurant** observer. **Restaurant** instance has a field myObserver (**MyObserver** class) which implements the observer pattern. When **Restaurant** instance updates data, it will call the **update** method of **MyObserver** class, the **update** method of **MyObserver** class will then call the **updateViews** method of **Eatee** class, which eventually updates the UI.

The GUI layout is as follows:

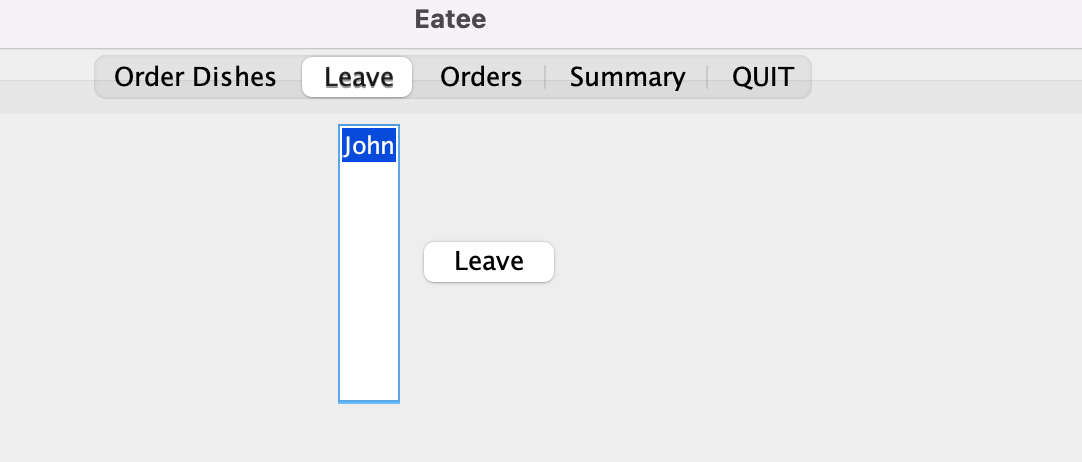


In the “Order Dishes” tab, users can select multiple dishes using the multiple selection list, then input the customer name in the text field, finally click the order button to order the dishes, it will create an order and add the order to **Restaurant**.

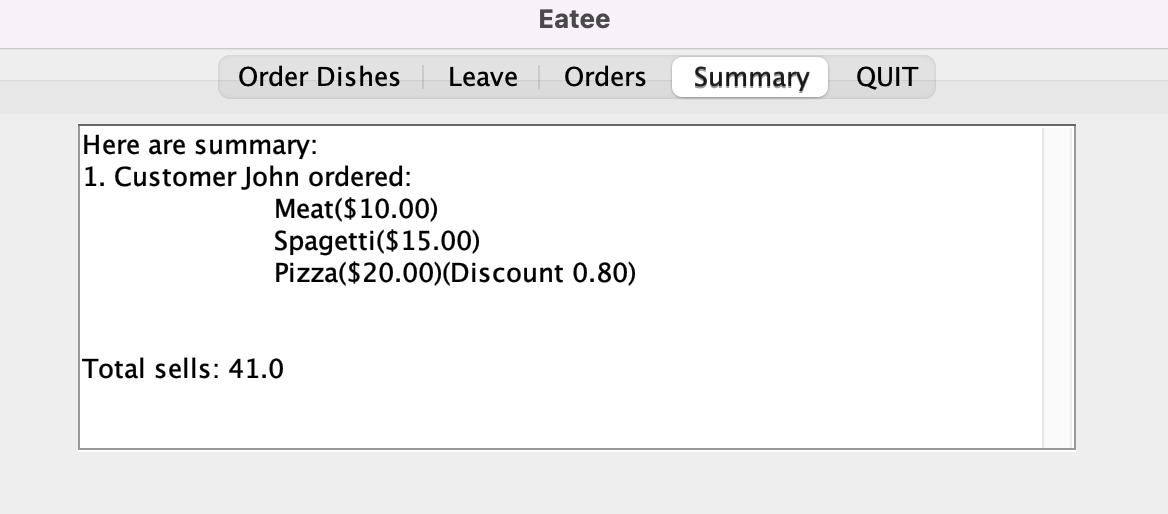
The UI is updated immediately by observer pattern, then we can view all orders in the “Orders” tab:



In the “Orders” tab, it shows all orders which wait to be proceed, then we can leave a customer in the “Leave” tab:



In the “Leave” tab, it shows all customers in the restaurant, then user can choose a customer to leave, user can also select multiple customers to leave since it uses multiple selection list. After clicking the **Leave** button, customer will be removed from **Restaurant**, and restaurant will add up the orders to summary. We can then view the summary of our restaurant:



In the summary, we can view the order history and the total sells of our restaurant.

Finally, we can quit the restaurant in “QUIT” tab:



Click the “Quit” button to quit Eatee program.

User can also click the default close button of **Eatee** window to quit.