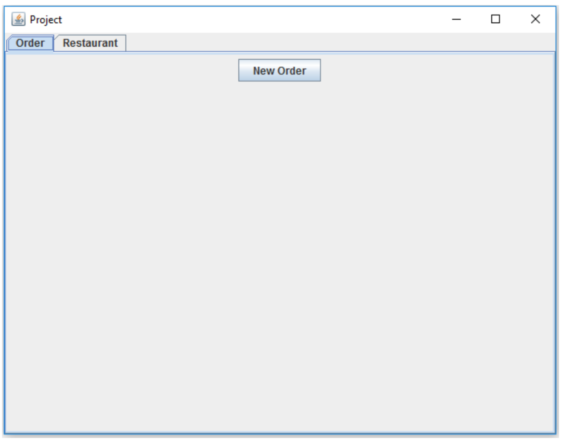
CS 102- Project

ÖZYEĞİN ÜNİVERSİTESİ

Due Date: Saturday, May 20, 2017 at 23:55pm

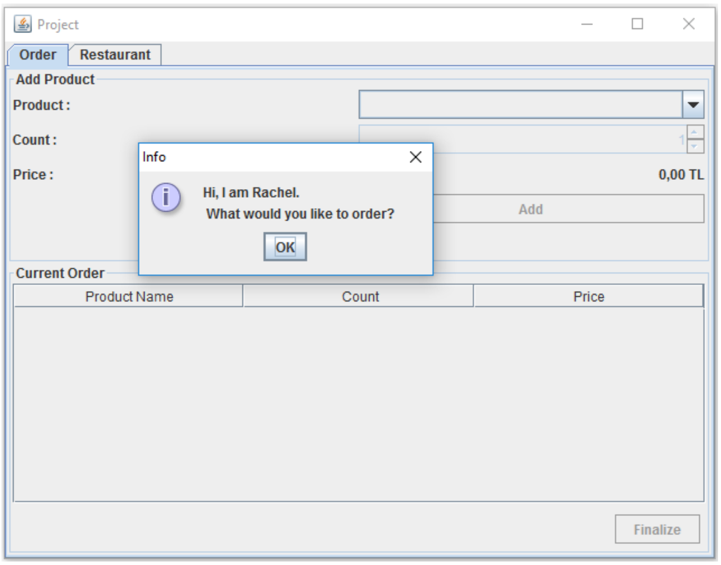
For the term project you are going to implement a restaurant application. This restaurant application can be used by both the restaurant customers and the restaurant manager. Customers can use the application to give their orders. Furthermore, the restaurant manager can use the application to follow the restaurant's expenses and revenues. The manager can also display the employees or add more employees. An example user interface for this restaurant application is presented in below figure.



As seen from the figure the order taking part is separated from the restaurant management part by the use of tabs. Details of how to implement those tabs will be given to you.

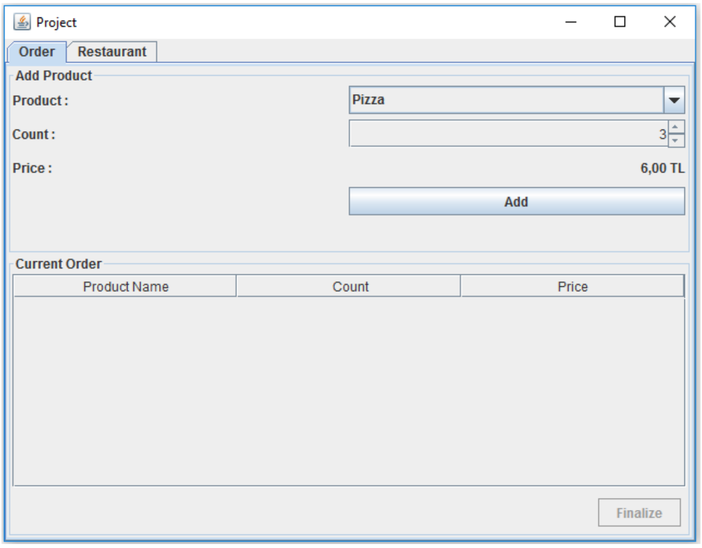
Order Creation:

If the user clicks on the "New Order" button in the initial state of the program (seen in figure above), the following will be shown to the customer.

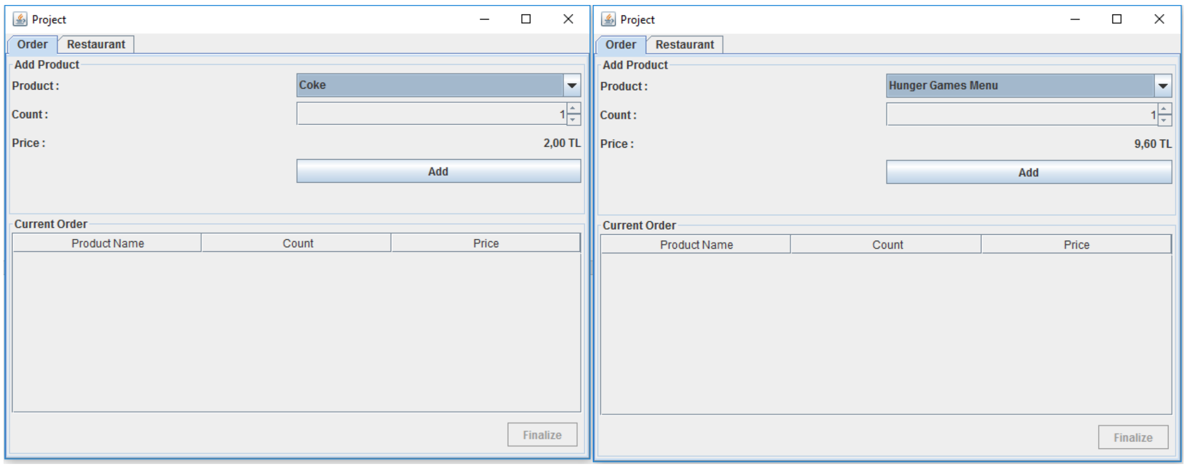


Hi, I am Rachel.
What would you like to order?

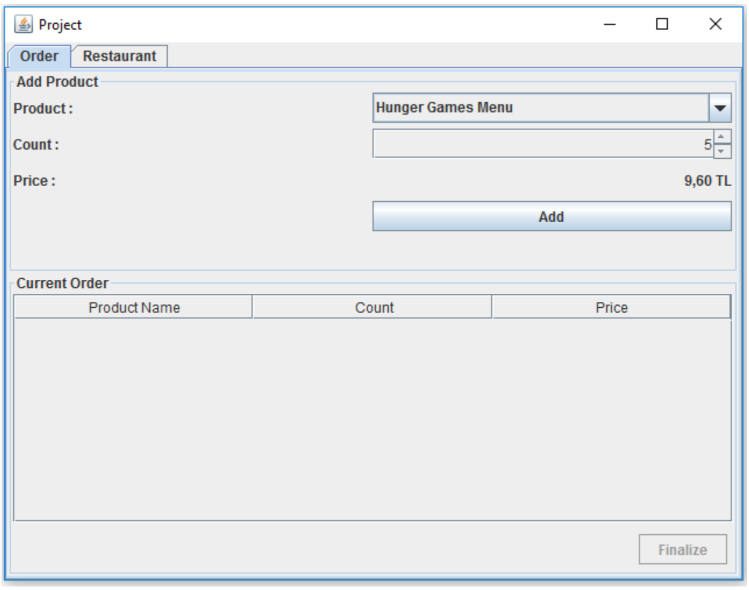
When customer wants to create an order, as the first thing, the system randomly assigns a waiter to the customer. A pop-up message will be displayed with the name of the waiter who is assigned randomly. After the closing the pop-up, the following view is displayed to the user.



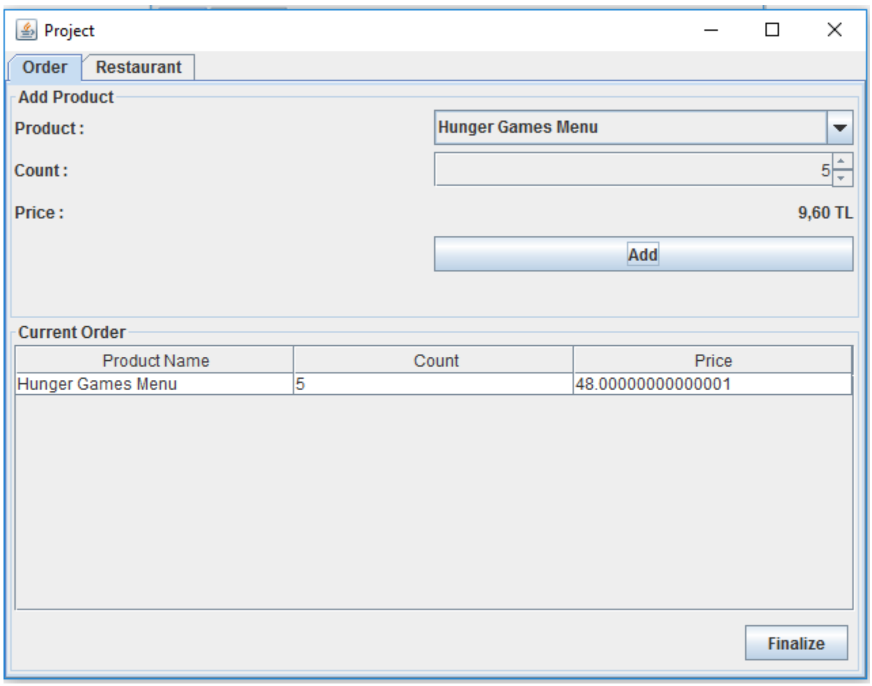
User can select any individual product or menu product from the combobox menu. Whenever a product is selected, the price of the product is displayed next to the Price label. For example, in the following screenshots, 2,00 TL is displayed for the price of the Coke and 9,60 TL is displayed for The Hunger Games Menu.



If user increases the count number, the price does not change as shown:

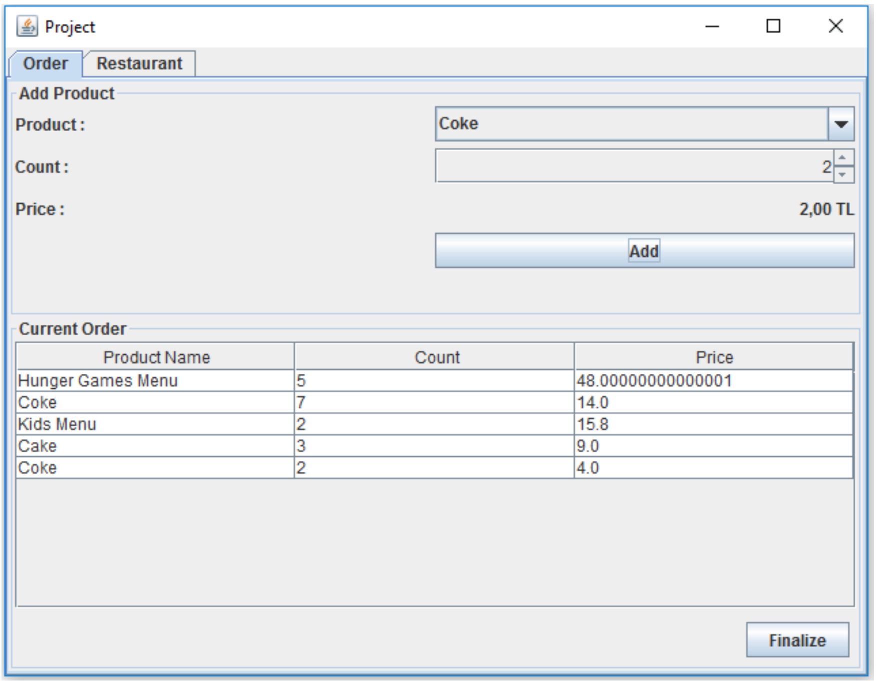


When user clicks on the “Add” button, the selected number of products will be added to the order and displayed in the lower panel as shown:



Product Name	Count	Price
Hunger Games Menu	5	48.00000000000001

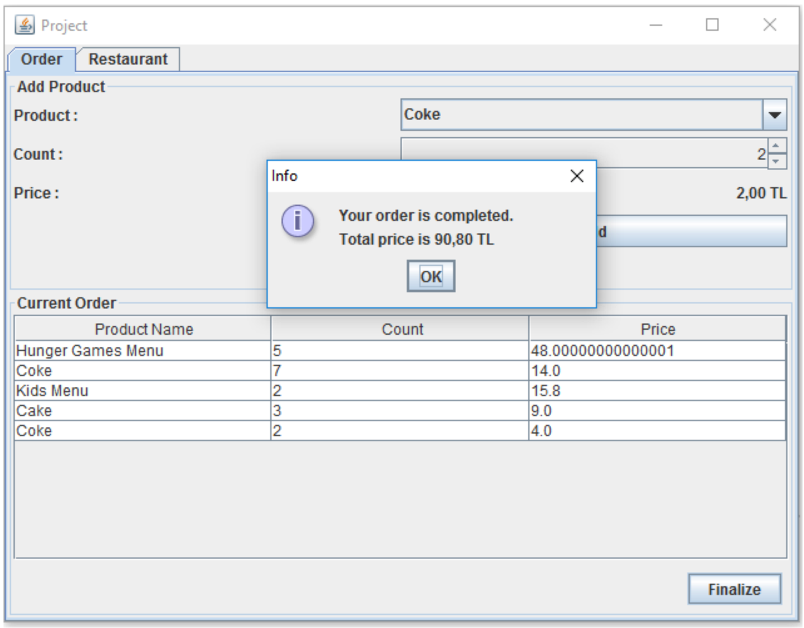
User can continue adding different products or more products of the same type.



Product Name	Count	Price
Hunger Games Menu	5	48.00000000000001
Coke	7	14.0
Kids Menu	2	15.8
Cake	3	9.0
Coke	2	4.0

All individual or menu products user has selected will be displayed in the table.

When user clicks the "Finalize" button, the order will be created and added to the orders list of the waiter. At the same time a pop up with the total price of the order will be displayed to the user. And then after user clicks "OK" button, the app will go to its initial state and continue from there.



Your order is completed.
Total price is 90,80 TL

Product Name	Count	Price
Hunger Games Menu	5	48.00000000000001
Coke	7	14.0
Kids Menu	2	15.8
Cake	3	9.0
Coke	2	4.0

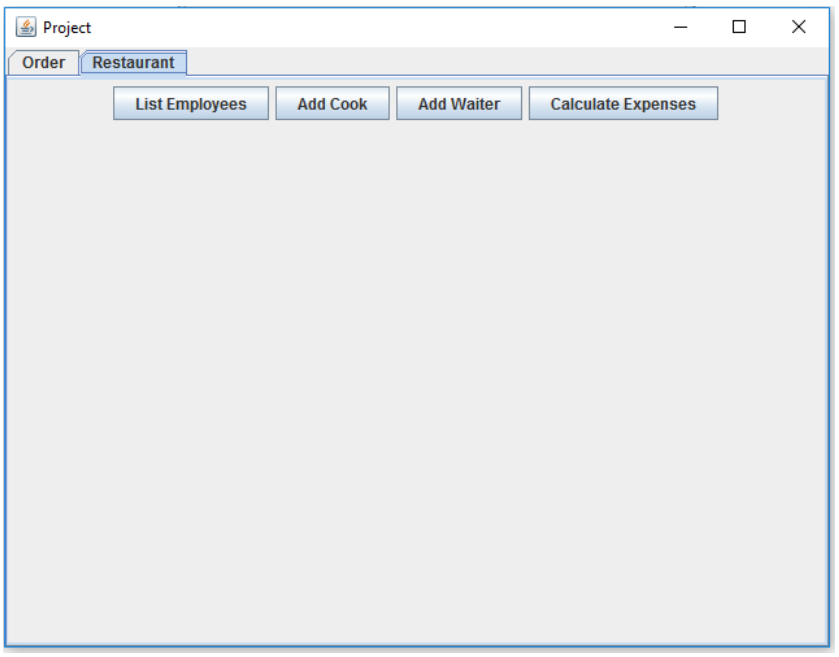
You can organize this view as your liking[1]. For example, instead of a table, you are free to use label but please try to put some formatting to the label so that it looks organized in general. You are supposed to implement all the functionalities shown and display all the information displayed. But you are free

[1] In order to give you some guidance, here is the detailed description of the components used in this part. There are two panels. The first one uses JLabel, JComboBox (to select product), JSpinner (to choose the count), JButton. The second panel uses JTable (to display the order) and JButton.

to do so in a way you like as long as functionally it is the same with provided example. Your GUI does not have to be the exact same view as shown in these snapshots.

Restaurant Management:

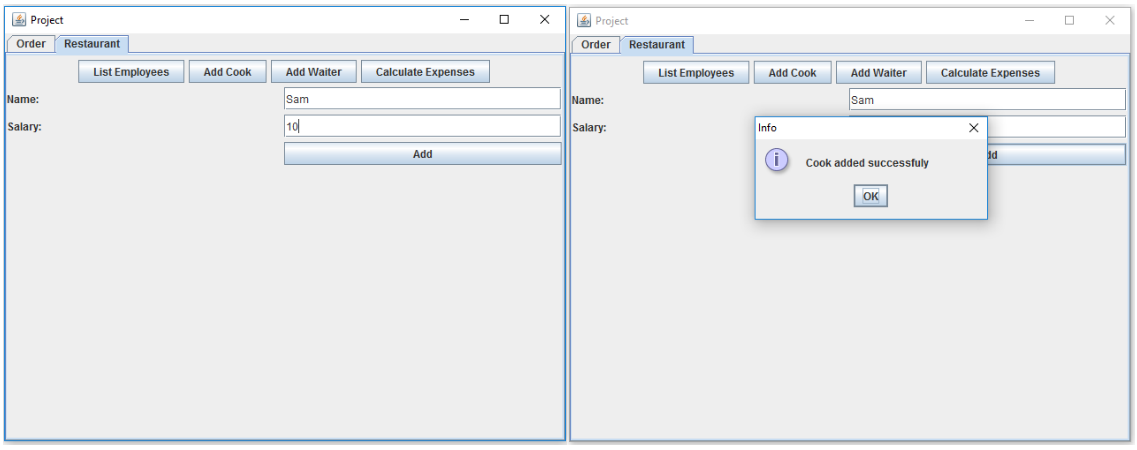
The manager of the restaurant can also use the same program in order to manage the restaurant.



There are 3 different types of things a user can do to manage the restaurant:

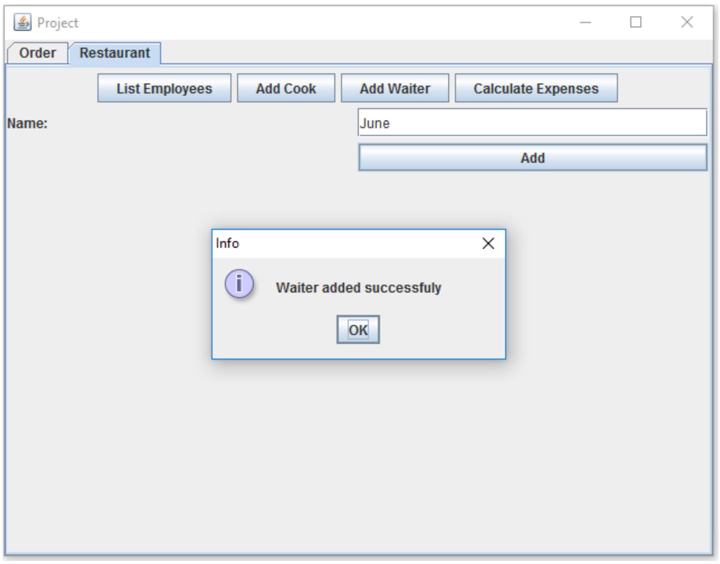
1. List the currently working employees
2. Add a new cook or a waiter
3. Calculate the expenses

As one can notice, there are two types of employees in this restaurant: (1) cook and (2) waiter. When user clicks the "Add Cook" button, the GUI will ask for the name and salary of the cook. Similarly, for the "Add Waiter" button. In that case, user only needs to enter the name of the waiter. The waiters do not have a base salary but instead get some percentage of the order they got from customers. More details will be explained later in the document.
When the user clicks the "Add" button, the cook/waiter is added to the employees. Employee's ID starts from 1 and incremented whenever a new employee is added. A pop up window will be displayed to user after insertion.



Name: Sam
Salary: 10

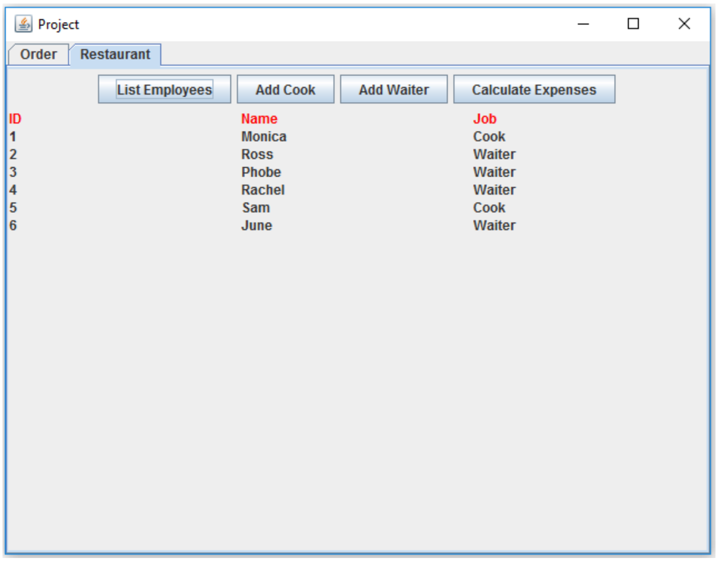
Cook added successfuly



Name: June

Waiter added successfuly

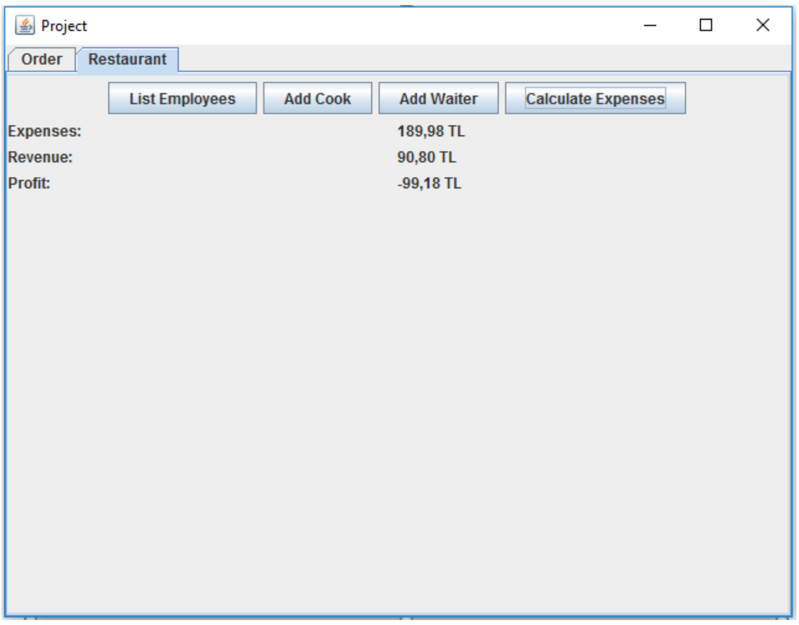
When user clicks “List Employees” button the all employees (waiter or cook) will be displayed to the user as shown. These employees will be sorted and printed with their employee IDs.



ID	Name	Job
1	Monica	Cook
2	Ross	Waiter
3	Phobe	Waiter
4	Rachel	Waiter
5	Sam	Cook
6	June	Waiter

Finally, when the “Calculate Expenses” button is clicked, the expenses and the revenue so far will be calculated and displayed to the user together with the profit[2]. Details of these calculations will be explained in the upcoming parts of this document.

[2] Profit amount is the difference of Revenue from Expenses.



```
Expenses:      189,98 TL
Revenue:       90,80 TL
Profit:        -99,18 TL
```

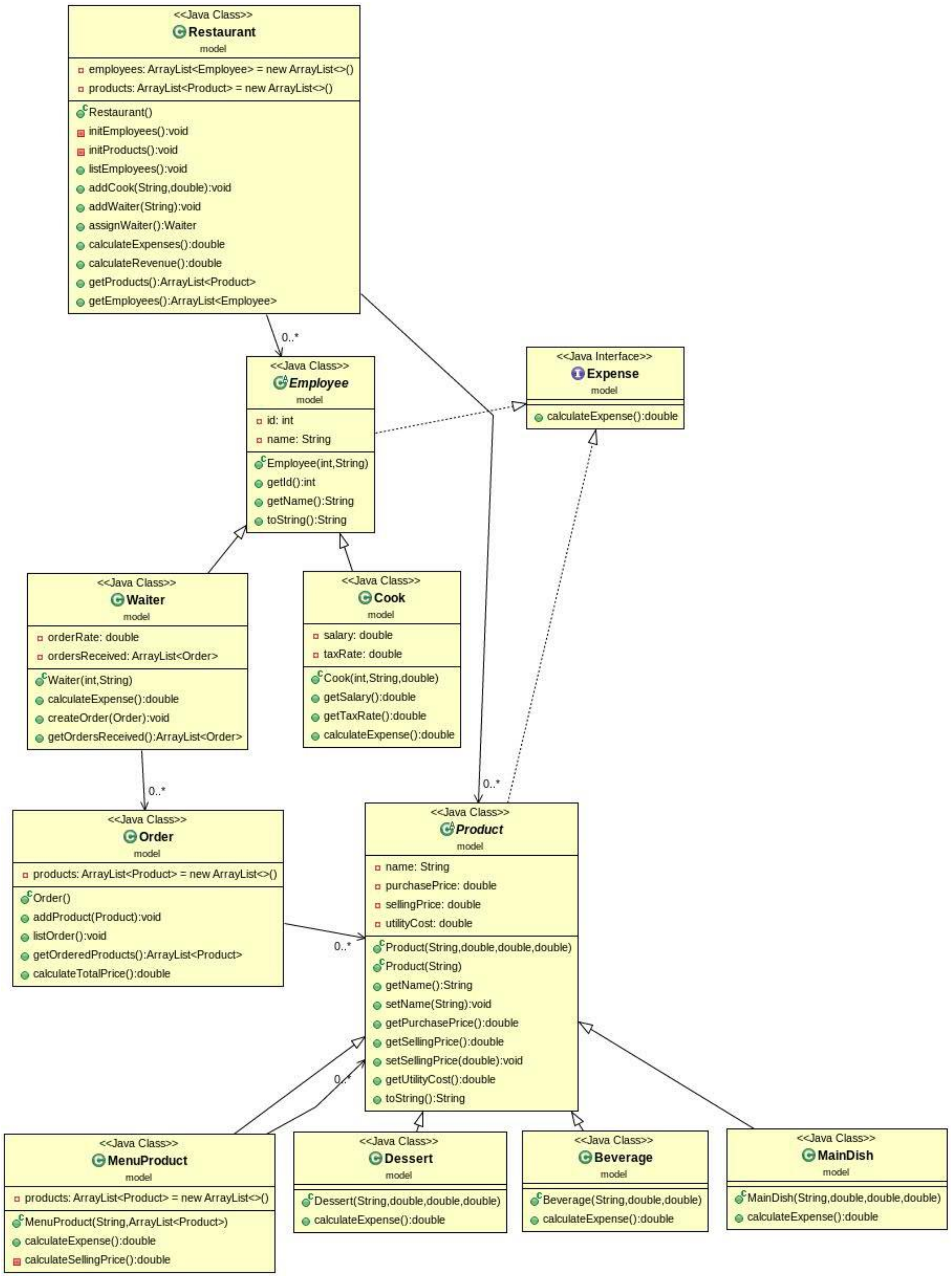
An example executable of this application is provided to you. Please refer to that for any unclear part. If you still have any unresolved question, feel free to ask it through Piazza.

Implementation Details

In this project, you are going to use the MVC pattern and separate your model from the GUI related parts. We strongly recommend you to implement the model part first and then implement the necessary GUI view and controllers. An example test main class is provided to you so that you can try and check your model. Please see the details of this main in Appendix 1.

The Model:

This part summarizes the details of the model implementation. You will be implementing several classes for the model part. A UML diagram which displays the classes and their relations is presented in below figure. The description of the classes is also summarized in detail.

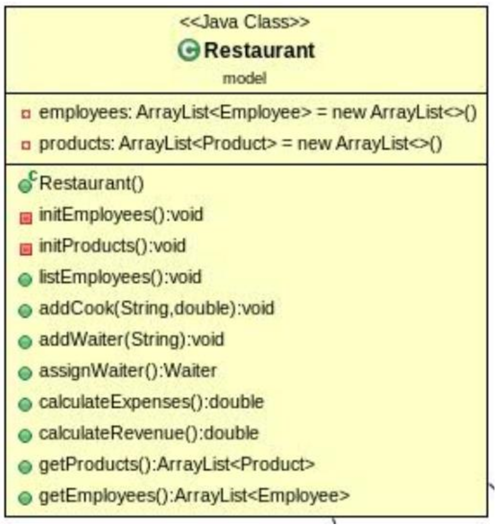


<<Java Class>>
Restaurant
model
employees: ArrayList<Employee> = new ArrayList<>()
products: ArrayList<Product> = new ArrayList<>()
Restaurant()
initEmployees():void
initProducts():void
listEmployees():void
addCook(String,double):void
addWaiter(String):void
assignWaiter():Waiter
calculateExpenses():double
calculateRevenue():double
getProducts():ArrayList<Product>
getEmployees():ArrayList<Employee>
0..*
<<Java Class>>
Employee
model
id: int
name: String
Employee(int,String)
getId():int
getName():String
toString():String
<<Java Interface>>
Expense
model
calculateExpense():double
<<Java Class>>
Waiter
model
orderRate: double
ordersReceived: ArrayList<Order>
Waiter(int,String)
calculateExpense():double
createOrder(Order):void
getOrdersReceived():ArrayList<Order>
<<Java Class>>
Cook
model
salary: double
taxRate: double
Cook(int,String,double)
getSalary():double
getTaxRate():double
calculateExpense():double
0..*
0..*
<<Java Class>>
Order
model
products: ArrayList<Product> = new ArrayList<>()
Order()
addProduct(Product):void
listOrder():void
getOrderedProducts():ArrayList<Product>
calculateTotalPrice():double
0..*
<<Java Class>>
Product
model
name: String
purchasePrice: double
sellingPrice: double
utilityCost: double
Product(String,double,double,double)
Product(String)
getName():String
setName(String):void
getPurchasePrice():double
getSellingPrice():double
setSellingPrice(double):void
getUtilityCost():double
toString():String
0..*
<<Java Class>>
MenuProduct
model
products: ArrayList<Product> = new ArrayList<>()
MenuProduct(String,ArrayList<Product>)
calculateExpense():double
calculateSellingPrice():double
<<Java Class>>
Dessert
model
Dessert(String,double,double,double)
calculateExpense():double
<<Java Class>>
Beverage
model
Beverage(String,double,double)
calculateExpense():double
<<Java Class>>
MainDish
model
MainDish(String,double,double,double)
calculateExpense():double

Figure. UML Diagram of the Restaurant Application Model

Restaurant.java:

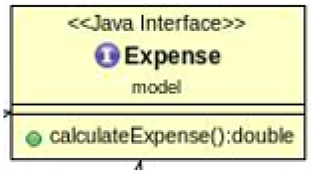
Partial implementation of the Restaurant.java will be provided to you. You need to implement the rest of this class yourself.
The constructor, initEmployees and initProducts methods are implemented for your convenience. You need to implement the other methods yourself. These three methods are implemented to initially create some employee and product objects. When your app starts, these products and employees will be available initially without the need for any user interaction.



<<Java Class>>
Restaurant
model
employees: ArrayList<Employee> = new ArrayList<>()
products: ArrayList<Product> = new ArrayList<>()
Restaurant()
initEmployees():void
initProducts():void
listEmployees():void
addCook(String,double):void
addWaiter(String):void
assignWaiter():Waiter
calculateExpenses():double
calculateRevenue():double
getProducts():ArrayList<Product>
getEmployees():ArrayList<Employee>

- **listEmployees** method prints out all employees.
- **addCook** method adds a cook to the restaurant. It takes the name and salary of the cook.
- **addWaiter** method adds a waiter to the restaurant. It takes the name of the waiter.
- **assignWaiter** method randomly selects a waiter and returns it.
- **calculateExpenses** method calculates the employee expenses and order expenses and returns the sum.
  In a restaurant, there are two types of expenses:
  - Employees' salaries and tax if any
  - Purchase costs of ingredients and utility cost of cooking ordered meals.
- **calculateRevenue** method calculates the revenue (earned money) from orders and returns it. This is not the profit. Profit is the Revenue – Expenses.
- **getProducts** returns the products.
- **getEmployees** returns the employees.

Expense.java:

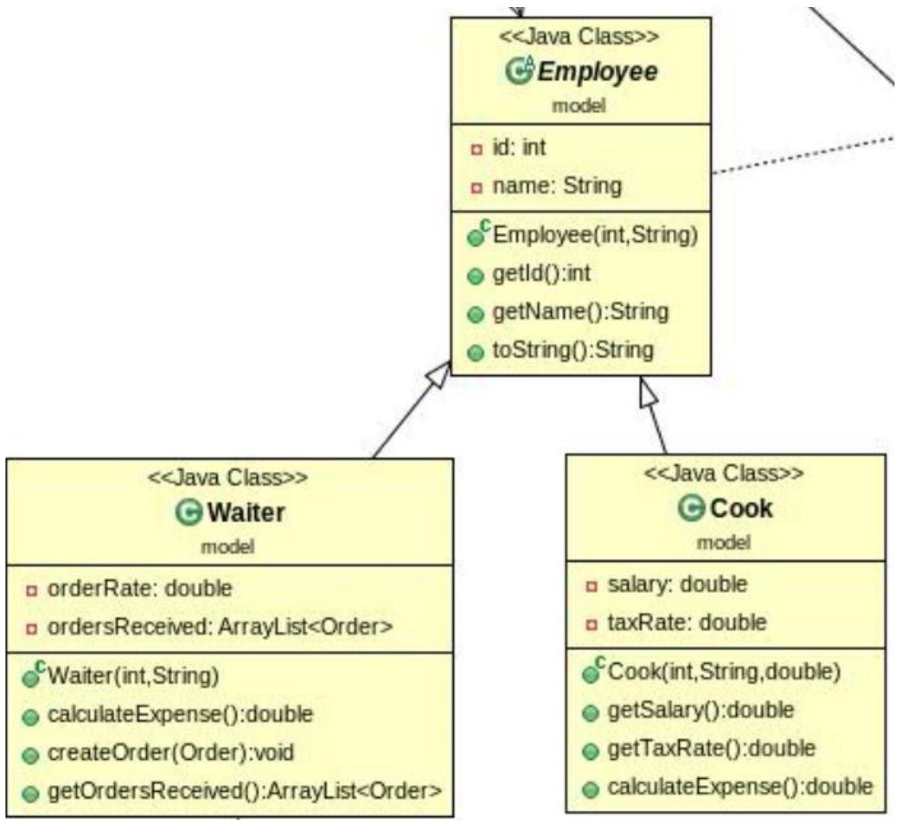


This class is an interface. A restaurant can have several type of expenses like the salary of the employees, utility costs and ingredient expenses. Create an interface with the calculateExpense() method which will be implemented by employee and product classes. This method returns the calculated expense.

Employee Classes:

Employee.java, Cook.java and Waiter.java

In this restaurant you will be implementing two types of employees: cook and waiter. All employees have some common attributes like their employee identification number (id) and name. Implement the necessary getMethods to access these attributes. Employee class implements the Expense Interface but there is not a default implementation of the calculateExpense() method. We should not allow the instantiation of an Employee. It needs to be either a cook or a waiter.

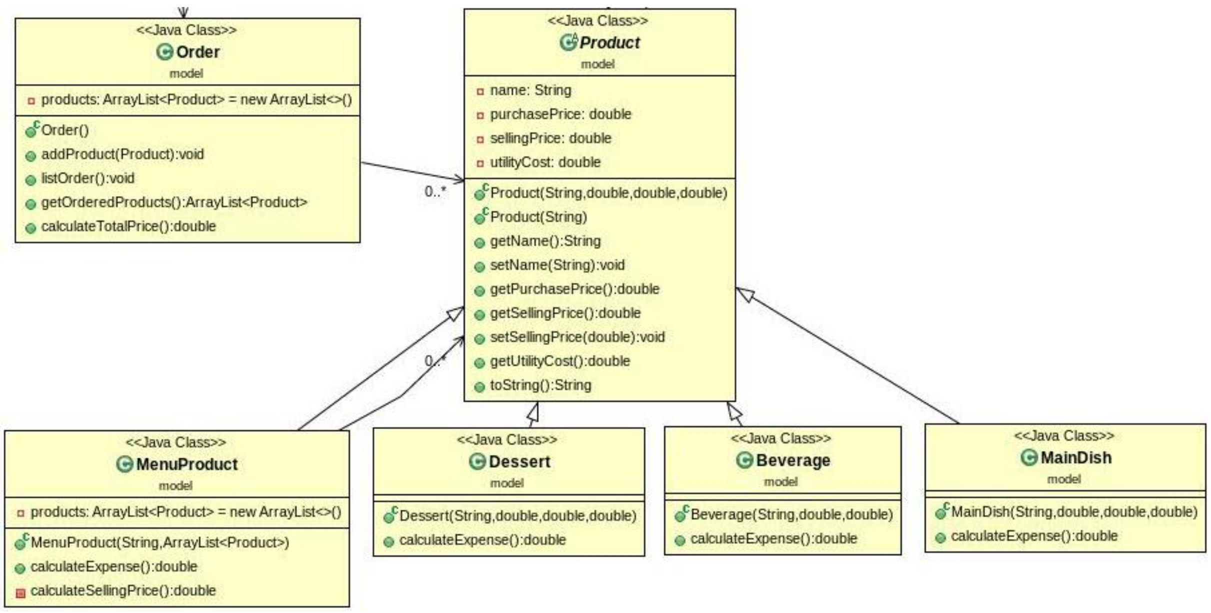


Cooks get a base salary. The restaurant pays their base salary as well as their tax which is calculated by multiplying the base salary with the tax rate (18%). The cook class holds these additional two attributes: the salary and tax rate.

Waiters are paid over the orders they serve. From each order they earn 10% of total price of the order. Even though it is not visible at the UML diagram Waiter class also holds an ArrayList which for the received orders (more details of this order class is explained in the rest of this document). You need to iterate over this orders list in order to calculate the salary of a waiter. The restaurant does not pay tax for the waiters. When manager wants to see the orders, they need to call **getOrdersReceived** method from each waiter. In order to add an order to this list of orders **createOrder** method needs to be called with the order sent as an argument.

Product Classes:

Product.java, MainDish.java, Dessert.java, Beverage.java and MenuProduct.java



<<Java Class>>
Order
model
products: ArrayList<Product> = new ArrayList<>()
Order()
addProduct(Product):void
listOrder():void
getOrderedProducts():ArrayList<Product>
calculateTotalPrice():double
0..*
<<Java Class>>
Product
model
name: String
purchasePrice: double
sellingPrice: double
utilityCost: double
Product(String,double,double,double)
Product(String)
getName():String
setName(String):void
getPurchasePrice():double
getSellingPrice():double
setSellingPrice(double):void
getUtilityCost():double
toString():String
0..*
<<Java Class>>
MenuProduct
model
products: ArrayList<Product> = new ArrayList<>()
MenuProduct(String,ArrayList<Product>)
calculateExpense():double
calculateSellingPrice():double
<<Java Class>>
Dessert
model
Dessert(String,double,double,double)
calculateExpense():double
<<Java Class>>
Beverage
model
Beverage(String,double,double)
calculateExpense():double
<<Java Class>>
MainDish
model
MainDish(String,double,double,double)
calculateExpense():double

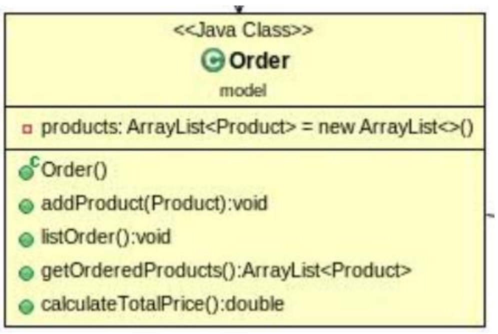
Implement a product class which holds the name of the product, its selling price, its purchase price (the cost of ingredients), and its utility cost (cost of cooking). The necessary get and set methods needs to be implemented. This product class also implements the Expense interface but leaves the implementation of the **calculateExpense** method to its subclasses.

The restaurant sells both individual products and menus which are a combination of individual products. Restaurant has three types of individual products. These are main dish, dessert and beverage.

The expense of main dish and desserts are calculated as the sum of the purchase price as well as the utility cost (like the gas used while cooking). However, the expense of beverages is just its purchase price (beverages do not need a processing therefore they don't have utility costs).

The menu product class contains an ArrayList of individual products. The **calculateExpense** method of the menu product calls the **calculateExpense** method from all these products and sum these total expenses to find the total expense of the menu product. Similarly, the selling price of the menu product is calculated by calling the **getSellingPrice** method of these products. The selling price of the menu is calculated and set by calling this method. The total price of menu is lower than buying those individual products alone. When calculating the selling price of the menu a 10% discount is applied to main dishes, 20% discount is applied to desserts and 50% discount to beverages.

Order.java:



<<Java Class>>
Order
model
products: ArrayList<Product> = new ArrayList<>()
Order()
addProduct(Product):void
listOrder():void
getOrderedProducts():ArrayList<Product>
calculateTotalPrice():double

An order is created by waiters for each group of customers. Order class holds an ArrayList of ordered products (either individual or menu).

- **addProduct** method is used to add a product to the order.
- **listOrder** method displays the ordered products.
- **getOrderedProducts** method returns the list of ordered products.
- **calculateTotalPrice** method calculates the total selling price of the order by iterating over products.

The GUI View and Controller:

In this GUI, the order taking part is separated from the restaurant management part by the use of tabs. Your GUI will use the same features and keep order menu separate from restaurant menu. The Java code that contains the main and necessary code to create these two tabbedpanes is provided to you within the file named "GUI.java". You will start with this code and implement other GUI view and controller classes as you see fit. All the functionalities of the GUI are summarized in this document with screenshots (see the first part of the document). Your program needs to have all these functionalities.

Additional Details:

You learned (or will learn in upcoming weeks) about exception handling. During the project, you are responsible from handling the any necessary exceptional cases. One way to handle those exceptions is to use pop-up windows. Leave this exception handling part to the end. First make sure that your program works with expected inputs without any error.

Submission

You will submit this homework via the LMS system. You should follow the file and method naming conventions for the model part. Point will be deducted if you don't do so.

Submit the following java files related to your model:

- Beverage.java
- Cook.java
- Dessert.java
- Employee.java
- Expense.java
- MainDish.java
- MenuProduct.java
- Order.java
- Product.java
- Restaurant.java
- Waiter.java

Put all these 11 files into a package named "Model".

Rest of your classes for GUI View and Controller implementation, create another package named as "GUI". Overall you will have two packages. Then zip these two packages.

Name your classes "exactly" the same as stated above. Capitalization of letters are important, too. (Do the same for the methods if stated in the pdf)

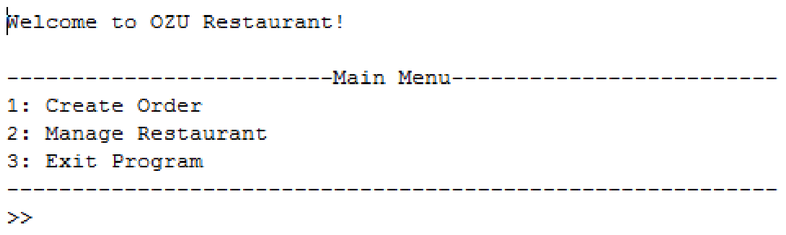
You should submit your source files as a **ZIP** archive file (**NOT** RAR or other formats). The name of the file should be in format “**<USER-ID>_project.zip**”. For example, if your username is un1234, then the name of the submitted file should be “un1234_project.zip”. Pay attention that all the letters are in lower-case.

Late submissions, missing submissions and source files that do not compile are **not** accepted and will receive 0 as the grade. After submission, download your assignment file to a location different than where your original assignment codes are. Make sure that the downloaded files contain everything to compile/build your assignment. Projects that do not compile will receive 0.

Do all the implementations yourself. In case of a plagiarism, you will receive -100.

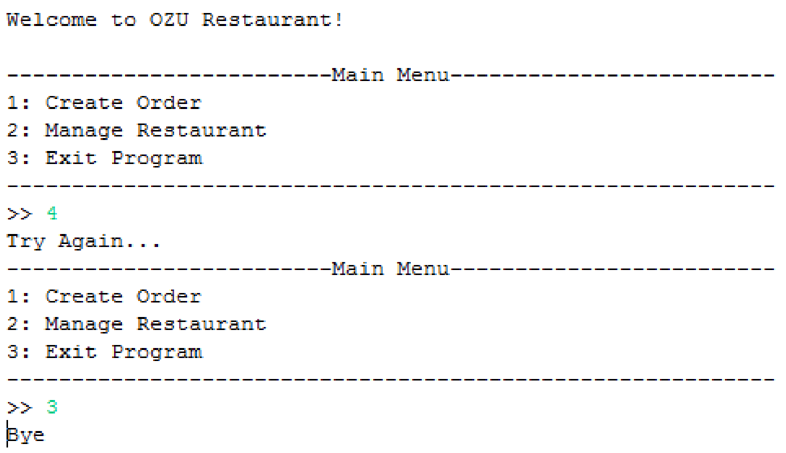
Appendix 1

An example main test class is provided to you so that you can test your model before moving to the GUI implementation (Do not submit this main test class). When this program starts, the following options will be displayed to users:



```
Welcome to OZU Restaurant!

-------------------------Main Menu-------------------------
1: Create Order
2: Manage Restaurant
3: Exit Program
-----------------------------------------------------------
>>
```

User can input 1 in order to create an order or 2 to manage the restaurant. If user enters a number that is not 1, 2 or 3, the program displays the menu again and wait for user input. In order to exit the program user needs to enter 3. An example snapshot of the program is as follows:



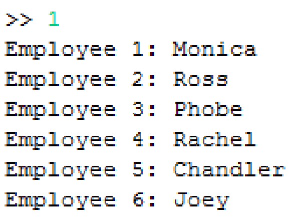
```
Welcome to OZU Restaurant!

-------------------------Main Menu-------------------------
1: Create Order
2: Manage Restaurant
3: Exit Program
-----------------------------------------------------------
>> 4
Try Again...
-------------------------Main Menu-------------------------
1: Create Order
2: Manage Restaurant
3: Exit Program
-----------------------------------------------------------
>> 3
Bye
```

The provided main.java contains the main method for displaying all these options. You don’t need to change this main. This main will be using other classes that you will be implementing.

Manage Restaurant

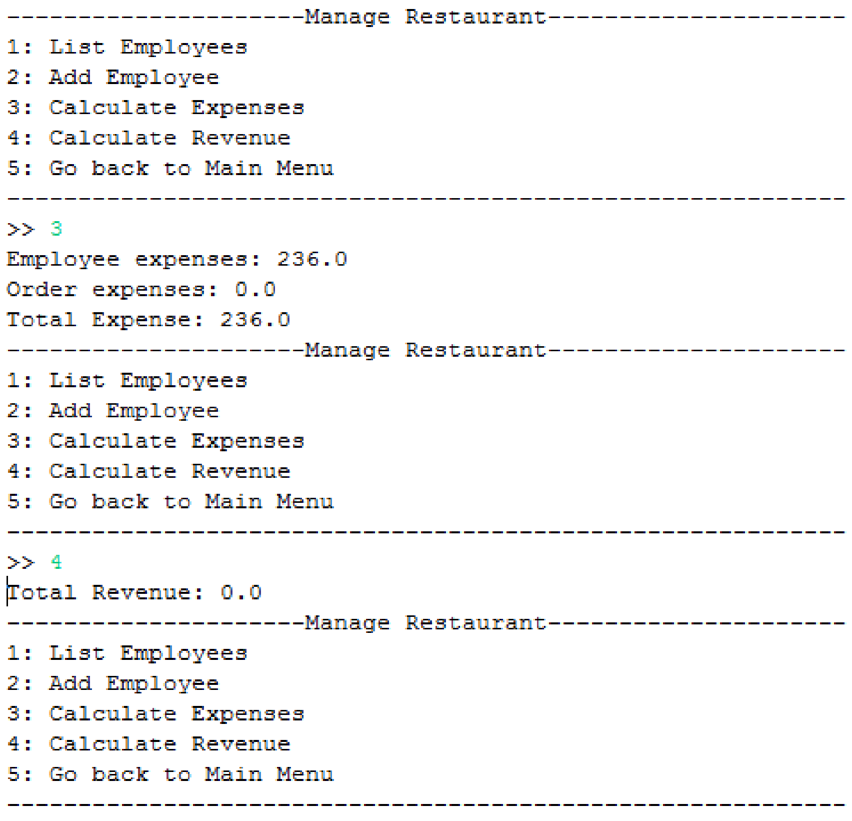
If user enters 2, manage restaurant menu will be displayed. In this menu, the manager can either list the employees, add a new employee, calculate expenses of the restaurant or calculate the revenue. Manager can also enter 5 in order to get back to Main menu. An example run of the manage restaurant is as following:

```
>> 2
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
>> 1
Employee 1: Monica
Employee 2: Ross
Employee 3: Phobe
Employee 4: Rachel
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
>> 2
---------------------Add Employee---------------------
1: Add Cook
2: Add Waiter
3: Go back to Manage Restaurant Menu
-----------------------------------------------------------
>> 1
Name of the Cook:
>> Chandler
Salary of the Cook:
>> 100
---------------------Add Employee---------------------
1: Add Cook
2: Add Waiter
3: Go back to Manage Restaurant Menu
-----------------------------------------------------------
>> 2
Name of the Waiter:
>> Joey
Returning to Main Menu
---------------------Add Employee---------------------
1: Add Cook
2: Add Waiter
3: Go back to Manage Restaurant Menu
-----------------------------------------------------------
>> 3
Returning to Manage Restaurant Menu
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
```


```
>> 1
Employee 1: Monica
Employee 2: Ross
Employee 3: Phobe
Employee 4: Rachel
Employee 5: Chandler
Employee 6: Joey
```

When manager wants to list all employees, all waiters are displayed in the following format:

Employee *employeeID*: *employeeName*


```
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
>> 3
Employee expenses: 236.0
Order expenses: 0.0
Total Expense: 236.0
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
>> 4
Total Revenue: 0.0
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
```

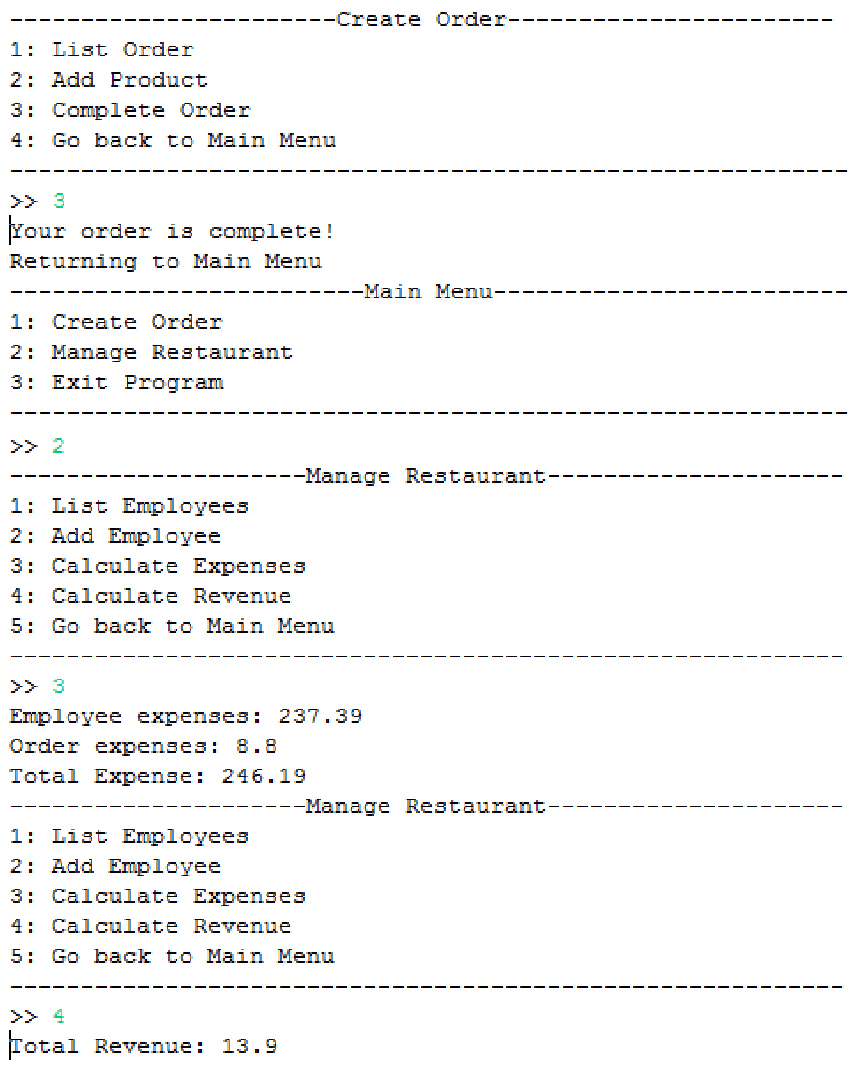
At the above example, there are no waiter costs since there is not any order given. There are two cooks, both with salary 100. There is a 18% cost for the tax. The total expense for the employees is therefore 2*(100 + 100*18%) = 236.

Since the restaurant did not get any order, the revenue is 0.

Create Order

When user wants to create an order, the system randomly assigns a waiter as shown below.

```
-------------------------Main Menu-------------------------
1: Create Order
2: Manage Restaurant
3: Exit Program
-----------------------------------------------------------
>> 1
Hi, I am Phobe.
I will be our waiter today.
What would you like to get today?
-----------------------Create Order-----------------------
1: List Order
2: Add Product
3: Complete Order
4: Go back to Main Menu
-----------------------------------------------------------
>> 1
You have not ordered anything yet!
-----------------------Create Order-----------------------
1: List Order
2: Add Product
3: Complete Order
4: Go back to Main Menu
-----------------------------------------------------------
>> 2
-----------------------Add Product-----------------------
1: Pizza
2: Burger
3: Coke
4: Lemonade
5: Tiramusu
6: Cake
7: Ice Cream
8: Hunger Games Menu
9: Kids Menu
>> 9
Kids Menu: 7.9
-----------------------Create Order-----------------------
1: List Order
2: Add Product
3: Complete Order
4: Go back to Main Menu
-----------------------------------------------------------
>> 2
-----------------------Add Product-----------------------
1: Pizza
2: Burger
3: Coke
4: Lemonade
5: Tiramusu
6: Cake
7: Ice Cream
8: Hunger Games Menu
9: Kids Menu
-----------------------------------------------------------
>> 1
Kids Menu: 7.9
Pizza: 6.0
```



```
-----------------------Create Order-----------------------
1: List Order
2: Add Product
3: Complete Order
4: Go back to Main Menu
-----------------------------------------------------------
>> 3
Your order is complete!
Returning to Main Menu
-------------------------Main Menu-------------------------
1: Create Order
2: Manage Restaurant
3: Exit Program
-----------------------------------------------------------
>> 2
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
>> 3
Employee expenses: 237.39
Order expenses: 8.8
Total Expense: 246.19
---------------------Manage Restaurant---------------------
1: List Employees
2: Add Employee
3: Calculate Expenses
4: Calculate Revenue
5: Go back to Main Menu
-----------------------------------------------------------
>> 4
Total Revenue: 13.9
```

If user does not enter 3, the order will not be completed.
After the order is given, the expenses and revenue has been changed as you can see above.