Self-Service Ramen System

Final Report

Group 17

Jiangqi Zhu (Leader)	171045895
Runsheng Jiang	161194680
You Wu	171044670
Shuaibo Zhao	171045220
Peng Xi	171045482
Junhan Chen	171045862

Menu

1. Summary of Responsibilities

Individual Member Contribution：Jiangqi Zhu

1. As a group leader he arranged group meetings to maintain communication and assigned tasks .
2. He implemented part of the codes for customers class and GUI interfaces.
3. He was responsible for writing the weekly report. And finish most parts of final report.

Individual Member Contribution：Junhan Chen

1. In charge of writing core functions mainly.
2. He implemented the function of customers and JUnit code for customers class.
3. At the same time, he was responsible for combining everyone's code together.

Individual Member Contribution：Runsheng Jiang

1. He wrote the codes for GUI interfaces, including the user ordering part and the administrator modifying part.
2. In later iterations, he was primarily responsible for modifying the details, like font size and font color, of GUI.
3. In charge of fixing bugs in the GUI interfaces.

Individual Member Contribution：Yunpeng Ge

1. He wrote the codes for GUI interfaces, including the user ordering part and the administrator modifying part.
2. In later iterations, he was primarily responsible for combining all GUI pages together.
3. In charge of fixing bugs in the GUI interfaces.

Individual Member Contribution：Peng Xi

1. He wrote the codes for Bill class, including writing the customer's billing information to txt and statistics of data.
2. He was responsible for organizing the stories we discussed according to the template.
3. In charge of fixing bugs in Bill class.

Individual Member Contribution：Shuaibo Zhao

1. He wrote the codes for Bill class, including writing the customer's billing information to txt and statistics of data.
2. He tested the code after each iteration and suggested improvements
3. In charge of fixing bugs in Bill class.

Individual Member Contribution：You Wu

1. She wrote the codes for Price class, including reading data from txt and responding when the administrator changes the price.
2. She was responsible for designing GUI prototypes and flow charts. And she put forward a lot of user requirements during meetings.
3. In charge of fixing bugs in Price class.

2. Introduction

The project aims to develop the software of a self-service machine for a ramen restaurant using Agile methods. To develop this system, we meet first to discuss and pool our ideas to identify specific needs. Fact-finding techniques, such as interviews, observations, etc., are used to define needs with sufficient clarity and precision. The java code was then written through an iterative process. Finally, we improved the user interface and wrote user manuals. We hope all restaurants and customers using our system t can be satisfied.

3. Project Management

3.1 Project Planning

This project is to develop the software of a self-service machine for a ramen restaurant using Agile Methods. In the development and iteration of the project, we will use Agile Methods throughout the process. The team members will investigate user needs in advance and adjust the functional structure of the project based on user requirements. By decomposing the entire project into multiple small tasks in different directions for the team members, we can complete the tasks more efficiently. At the same time, the team members will often hold meetings to discuss project progress and keep information exchange at all times.

3.2 Agile method

3.2.1 Overall goals and software architecture

The general objective is to develop the software of a self-service machine for a ramen restaurant. The software can supply a menu to choose from for the customer. Customers need to choose their favorite ramen and ingredients from a fixed selection. At the same time, the software also provides additional dishes, customers can choose freely.

The software provides a membership plan, customers can choose to join or not, through registration to obtain membership status, in the future consumption can get certain discounts. Customers can also choose a variety of payment methods to pay for meals.

It also provides a separate interface for the administrator to operate, you can modify the menu, view statistical data, etc. The administrator can add dishes, change the payment method, the use of the software is flexible and expandable.

3.2.2 Iterative Process

In the first version of the software, we have realized the basic functions, users can choose dishes according to their needs, and generate data in real time to pass to the management side.

In the first iteration, we optimized the data type of the order, so that the overall structure of the software was upgraded to facilitate logical processing.

In the subsequent iteration process, part of the GUI of the software was changed, such as centering some windows, changing the font color, etc., the software became more beautiful and convenient for customers to operate. At the same time, the team members added some fault tolerance mechanisms to prevent the problems that the software cannot be used normally due to errors in user input information.

Finally, we added a special event, where customers spend half the price on Thursday. Adding events can increase the attractiveness of the restaurant, thereby increasing operating income.

3.3 Time schedule

Item	Time	Task
Early stage	March 5th - March 15th	Analyze the task requirements of the project, investigate user needs, and design the software framework
Mid-term	March 16th - April 20th	Initially complete the first version of the software, check and improve software bugs, discuss the iteration plans
Late stage	April 21th - May 7th	Iteratively update the software to complete the final version
End stage	May 8th - May 29th	Check the operation of the software and write final report

3.4 Group management

Subgroup	Members	Responsibilities
Group 1	Runsheng Jiang, Yunpeng Ge	write the codes for GUI interfaces, including the user ordering part and the administrator modifying part. Combine all GUI pages together and in charge of fixing bugs in the GUI interfaces.
Group 2	Jiangqi Zhu, Junhan Chen	Implement the Junit code of the customer function and customer class, write the core function, combine the code of each member, complete and upload the weekly report
Group 3	Shuaibo Zhao, Peng Xi	Write the codes for Bill class, including writing the customer's billing information to txt and statistics of data.Test the code after each iteration and suggested improvements.In charge of fixing bugs in Bill class.
Group 4	You Wu	Write the codes for Price class, including reading data from txt and responding when the administrator changes the price. Responsible for designing GUI prototypes and flow charts. And ut forward a lot of user requirements during meetings.

3.5 Risk analysis

Risk Type	Risk	Effects	Avoidances Strategies
Project risks	Affected by the epidemic, students cannot return to school, and communication may be problematic	Serious	Conduct online meetings in a timely manner through multimedia software, and team members communicate frequently to avoid problems caused by poor information

Product risks	The product has bugs during the development process and cannot run normally	Serious	Analyze the software, find and solve bugs in time. Ensure the stability of the software through multiple iterations
Technology risks	The team members have insufficient technical ability when developing individual functions	Serious	Through multi-person collaboration, find relevant information online and improve members' programming skills
Time risk	The time to complete the final software version and report may exceed the deadline	Serious	Make good time planning and improve efficiency through a reasonable division of labor

3.6 Decision making

In the process of project advancement, the team leader regularly organizes online meetings to exchange project progress, and someone is responsible for recording the content of each meeting for later review. Whenever a problem is encountered, the team members actively communicate, and when several solutions is reached, a collective vote decides which method to choose. The progress of the project depends on the thinking of each team member, and the members brainstorm and work together to complete the project.

3.7 Adapt to changes

During the development of the project, we will always encounter some problems of change. In order to solve this change in time, we will hold regular meetings and discuss these problems. When faced with new user requirements, team members use an extensible structure in the software to easily add new functions, and deal with and solve problems in a timely manner

4. Requirement

4.1 Fact-finding techniques

4.1.1 Background reading

To better satisfy the customer needs and understand the project deeply, we paid lots of attention to identifying the requirements and read many reataurant reports. From what we have learnt, it required our development team to design a self-service machine for a ramen restaurant which should offer the following basic functions: ordering ramen, choosing their favorite taste, joining the loyalty scheme and paying for their bills for customers; modifying menu and viewing food status for restaurant owner.

4.1.2 Interviewing

To get a deeper and clearer understanding of the user requirements, we are divided into several groups to interview different stakeholders such as owners and waiters of restaurants and customers. When interviewing customers, we paid more attention to the elderly people who might have difficulty in using electric system. To satisfy needs of elderly, we need to make our service machine a simple but convenient one. we also kindly requested our interviewees to describe the steps of ordering meals in the restaurant and list the difficulties they have met.

4.1.3 Observation

Along with the interviews, we went to several restaurants to observe the process of ordering meals and recorded the improvements we can make. It proved to be a very effective way to understand better about our service machine and gave us first-hand information of how to improve it.

4.1.4 Document or record sampling

We sorted out the document and record we had collected, including other restaurant reports, interview notes and observation notes. Then, we listed the features of our project based on these record.

4.2 Changes of the Product Backlog

Customer	Want a wonderful software UI		So that attract more businesses to choose .	
	Want Generality： The software should be flexible and extensible E.g. to be used in another restaurant, add dish, add payment options etc.		So that it can be used in a general market in the future and improve product value.	
End user	easy to use	Want to be able to operate the software with common sense or with simple instructions,	So that save time	So that customers know what they should do in each phase.
		Want a simple registration process		So that avoid disturbing customers.
	user friendly	Want the restaurant to Provide more ingredients and ingredients, including ingredients, ingredients, spicy degree,	So that attract customers use it.	So that meet the preferences of customers.
		Want have a place to evaluate the dishes or the software		So that I can express my opinions or protect my rights.
	place	Want the restaurant give more dining options (‘Eat in’ and ‘Take-away’) so that improve the convenience.	So that improve the convenience	

		Want to take-out option	So that can eat at home
	ticket and order	Want to print tickets, including order information and customer information	So that help customer to make show what they've ordered
		Want to modify order	So that improve fault tolerance
		Want a wide variety of payment methods	customers will feel very flexible
	Recommendation system (extra)	Want recommendations based on the sales of the product in several weeks	So that better help customer order
Development member	Data type	Want Basic restrictions and error checking	So that reduce software errors.
	File format	Want to use plain Text (txt), CSV, JSON, or XML，	So that help data processing.
	Software iteration	Want to be capable of adapting to such future changes	So that help software iteration.
	Code decoupling	Want to split code into several parts that are not highly dependent	So that different members of team can develop at the same time.
Management	Time saving	Want to develop the software of a self-service machine，	So that avoid wasting time.
		Want the customers pay all the food they need at one time,	so that reduce the service cost of managers.
	Loyalty scheme	want have a Loyalty scheme，count customers' consumption times and provide feedback plan	so that appear more loyal customer
		want to visualize how many virtual stamps s/he received so far, an email and/or SMS will be sent to inform him of the number of stamps collected so far	so that attract customers
		want more concessions	so that attract customers

	independent manager interface for order and performance statistics	Want to manage the menu, you can modify the menu content and price (operation: add, delete, modify and check).	so that the business can maintain the restaurant business
		Want to add new dish directly by operating software and without change txt by myself.	so that I can operate it easily
	Want to manage orders, count order information, turnover, the most popular dishes of the week (Monday to Sunday) and the total revenue of the week,		so that facilitate the management of restaurants.
	Want a message board that allows users to comment on dishes, management delete and reply users' comments,		so that adjust the menu in time to meet customers' preferences.
Technology provider	Want to develop using Java as a stand-alone application. Simple graphic user interface (GUI) should be used. Latest Java SE (10 or above) should be used.		So that reduce the equipment cost of managers.
	want to limit software without the specific payment function, mail function and database		So that this is a simple simulation software .

4.3 Iteration and Estimation of the Stories

We have 4 iterations to develop a self-service machine for a ramen restaurant.
Iteration 1: This iteration is the foundation of the whole project. We confirmed interface and instance variables.
Iteration 2: In this iteration, we implemented all the basic functions, checked file-reading and writing interface and confirmed GUI interface.
Iteration 3: In the third iteration, we integrated our codes, fixed up some bugs, and tried to add some new features.
Iteration 4: In the final iteration, we completed debug of the final program and tested all the features. Finally, we finished the report.

5. Analysis and Design

5.1 Class Diagram and Justification

5.1.1 Class Analysis

Boundary classes	CheckMenu, DirectPay, DirectPay2, Login1, Login2, Manage, ManageLogin, ModifyBegin, ModifyMenu1, ModifyMenu2, ModifyMenu3, Ordering1, Ordering2, Ordering3, Ordering4, Pay1, Pay2, Registered, Registered1, Registered2, Sales, Start, StatisticsGUI.		
Control Classes	Dao	Implements: AvailableDaoImpl, BillDaoImpl, CustomerDaoImpl, PriceDaoImpl, ReceiptDaoImpl.	Interface: AvailableDao,BillDao, CustomerDao, PriceDao, ReceiptDao.
	Service	Implements: AvailableServiceImpl, BillServiceImpl, CustomerServiceImpl, PriceServiceImpl, ReceiptServiceImpl.	Interface: AvailableService, BillService, CustomerService, PriceService, ReceiptService.
Entity Classes	Available, Bill, Customer, MostPopMeal, Price, Statistics.		

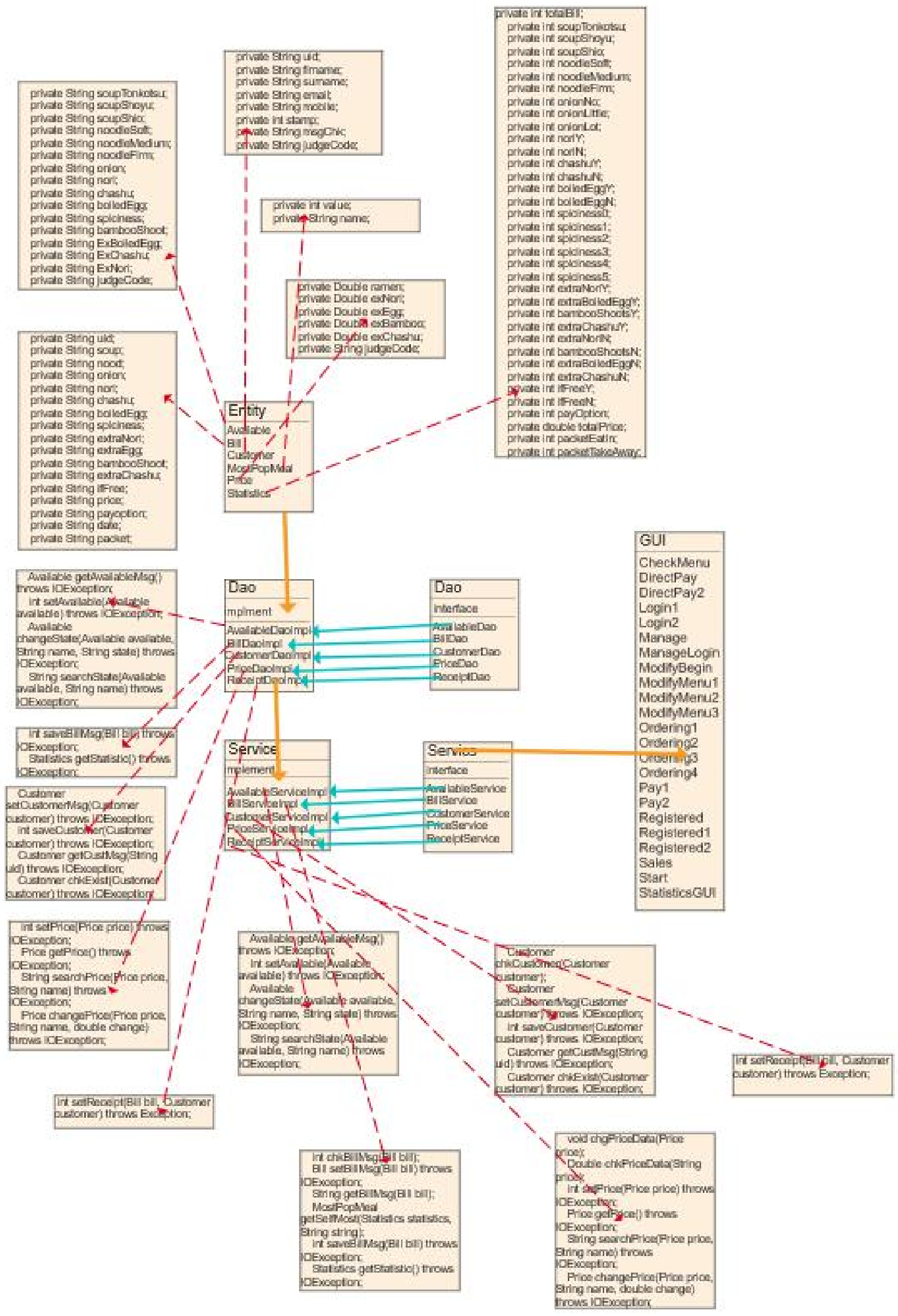
5.1.2 Design Idea of Class

We deign to divide our Toroto Ramen System using MVC architecture. The Dao layer is the Model part, the GUI layer is the View part and the Service is the Control part. At the same time, to achieve the basic functionality of these three layers, we use the entity package to hold entity classes. GUI layer has lots of parts like login, register, pay, etc. Each class in the GUI package corresponds to a specific interface. The Dao layer is mainly used for reading and writing files, including availability, price, customer message, bill message and receipt. Service layer still have these parts, which is used to control the software and access the Dao layer.

5.1.3 Adaptability to Changes

Our Toroto Ramen System is very adaptable to changes. From the class diagram, we can see that both the Dao layer and the Service layer have a corresponding abstract class set for the functional back-end code. If the code needs to be added or modified, only the implementation classes under the Impl package need to be modified.

5.1.4 Class Diagram



```
private String soupTonkotsu;
private String soupShoyu;
private String soupShio;
private String noodleSoft;
private String noodleMedium;
private String noodleFirm;
private String onion;
private String nori;
private String chashu;
private String boiledEgg;
private String spiciness;
private String bambooShoot;
private String ExBoiledEgg;
private String ExChashu;
private String ExNori;
private String judgeCode;
```

```
private String uid;
private String fname;
private String surname;
private String email;
private String mobile;
private int stamp;
private String msgChk;
private String judgeCode;
```

```
private int value;
private String name;
```

```
private Double ramen;
private Double exNori;
private Double exEgg;
private Double exBamboo;
private Double exChashu;
private String judgeCode;
```

```
private int totalBill;
    private int soupTonkotsu;
    private int soupShoyu;
    private int soupShio;
    private int noodleSoft;
    private int noodleMedium;
    private int noodleFirm;
    private int onionNo;
    private int onionLittle;
    private int onionLot;
    private int noriY;
    private int noriN;
    private int chashuY;
    private int chashuN;
    private int boiledEggY;
    private int boiledEggN;
    private int spiciness0;
    private int spiciness1;
    private int spiciness2;
    private int spiciness3;
    private int spiciness4;
    private int spiciness5;
    private int extraNoriY;
    private int extraBoiledEggY;
    private int bambooShootsY;
    private int extraChashuY;
    private int extraNoriN;
    private int bambooShootsN;
    private int extraBoiledEggN;
    private int extraChashuN;
    private int ifFreeY;
    private int ifFreeN;
    private int payOption;
    private double totalPrice;
    private int packetEatIn;
    private int packetTakeAway;
```

```
private String uid;
private String soup;
private String nood;
private String onion;
private String nori;
private String chashu;
private String boiledEgg;
private String spiciness;
private String extraNori;
private String extraEgg;
private String bambooShoot;
private String extraChashu;
private String ifFree;
private String price;
private String payoption;
private String date;
private String packet;
```

```
    Available getAvailableMsg()
throws IOException;
    int setAvailable(Available
available) throws IOException;
    Available
changeState(Available available,
String name, String state) throws
IOException;
    String searchState(Available
available, String name) throws
IOException;
```

```
    int saveBillMsg(Bill bill) throws
IOException;
    Statistics getStatistic() throws
IOException;
```

```
    Customer
setCustomerMsg(Customer
customer) throws IOException;
    int saveCustomer(Customer
customer) throws IOException;
    Customer getCustMsg(String
uid) throws IOException;
    Customer chkExist(Customer
customer) throws IOException;
```

```
    int setPrice(Price price) throws
IOException;
    Price getPrice() throws
IOException;
    String searchPrice(Price price,
String name) throws
IOException;
    Price changePrice(Price price,
String name, double change)
throws IOException;
```

```
int setReceipt(Bill bill, Customer
customer) throws Exception;
```

```
    Available getAvailableMsg()
throws IOException;
    int setAvailable(Available
available) throws IOException;
    Available
changeState(Available available,
String name, String state) throws
IOException;
    String searchState(Available
available, String name) throws
IOException;
```

```
    Customer
chkCustomer(Customer
customer);
    Customer
setCustomerMsg(Customer
customer) throws IOException;
    int saveCustomer(Customer
customer) throws IOException;
    Customer getCustMsg(String
uid) throws IOException;
    Customer chkExist(Customer
customer) throws IOException;
```

```
int setReceipt(Bill bill, Customer
customer) throws Exception;
```

```
    int chkBillMsg(Bill bill);
    Bill setBillMsg(Bill bill) throws
IOException;
    String getBillMsg(Bill bill);
    MostPopMeal
getSelfMost(Statistics statistics,
String string);
    int saveBillMsg(Bill bill) throws
IOException;
    Statistics getStatistic() throws
IOException;
```

```
    void chgPriceData(Price
price);
    Double chkPriceData(String
price);
    int setPrice(Price price) throws
IOException;
    Price getPrice() throws
IOException;
    String searchPrice(Price price,
String name) throws
IOException;
    Price changePrice(Price price,
String name, double change)
throws IOException;
```

5.2 Design Principles

5.2.1 Single Responsibility Principle (SRP)

A class should have only one reason for change. Each class in our Toroto Ramen system meets the single responsibility requirement, which means that there is only one reason for the class change. We do not use conditions to judge the execution of the upper classes. Instead, our system divides them into different classes to satisfy SPR.

5.2.2 Open-Closed Principle (OCP)

If requirements change or new requirements arise, we can make the function operate in a completely different way. But we should be able to do this without changing the function code.

5.2.3 Don’t Repeat Yourself (DRY)

The Don't Repeat Yourself (DRY) principle states that duplication in logic should be eliminated by abstraction; duplication should be eliminated through automation. In our design, we focus on avoiding the use of duplicate code, and strive to make our code more efficient and high-quality. For example, in our code, we have clear judgment criteria for the price data format and user input format, so we extract them from the GUI and set them as a method to avoid repeated writing in the GUI.

5.2.4 Interface Segregation Principle (ISP)

Each individual interface is implemented by a GUI class, so that it implements ISP. Classes that implement interfaces will not be forced to rely on methods they do not use. In addition, if other interfaces need to be implemented in further development, we can easily implement them. Compliance with ISP can enable our system to implement useful methods without introducing unnecessary methods.

5.2.5 Dependency Inversion Principle (DIP)

This system follows the design method of high-level-> abstract-> low-level instruction.

5.2.6 Liskov Substitution Principle (LSP)

The Liskov substitution principle states that you should not rewrite methods before changing their previous behavior. So in our design, we ensure that new derived classes that extend other classes will not replace any functions in the old class

6. Implementation and Testing

6.1 Implementation

6.1.1 Requirement Achieved

Customer part:

- Order the meal
- Choose staple food
- Choose the basic ingredient (soup,noodles,spring onion,nori,chashu,boiled egg,spiciness)
- Choose extra ingredient (nori, boiled eggs, shoots, chashu)
- Check the order again before pay it
- Choose eat-in or take-out

- Become the member
- Member login
- Select the mode of payment(cash/postcard). The number of stamps will automatically grow if purchased with cash
- Check the final bill

Manager part:

- Login to the manager account
- Change the price / availability of staple food or ingredient.
- View statistics
- Check the sale situation for all food.
- View sales report for different types of product.

Discount: (Just a Painted Eggshell for fun !)

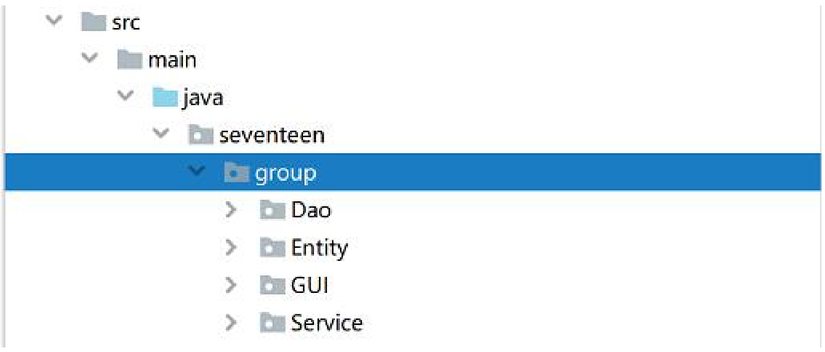
- The price of food will halve on Thursday!

6.1.2 Implementation Strategy

The system is implemented in Java language. The process includes integration, class implementation, and subsystems. We first realized the main functions, then designed the GUI, and realized the data synchronization of the front and back ends, and finally conducted a lot of tests according to each class.

6.1.3 File Structure

To better manage the project and make the code structure clearer. In this project, our Java source code was assigned to four different folders based on its capabilities.



```
src
  main
    java
      seventeen
        group
          Dao
          Entity
          GUI
          Service
```

1. Dao:

Under the Dao directory, a number of abstract classes are first included. There is also an internal IMPL folder that contains the Java code that implements these abstract classes.

The code in the Dao is mainly used to interact with the database. We can use these codes to read or store user information, order information, menu information, and so on.

In this project, we used txt instead of the database.

2. Entity:

The java code in Entity directory mainly contains Bill.java, Customer.java, Price.java, and so on. They all have their own attributes. They're all entities that we're going to be working with on this project.

3. GUI:

The Java code in this folder is primarily each GUI page. The source code for all of our interfaces in this project is in this folder.

4. Service:

There is also a number of abstract classes are first contained in this directory. And there is an internal IMPL folder that contains the Java code that implements thest abstract classes.

Service contains both control methods used to call methods in Dao in order to interact with file stream. It also includes some methods that not interact with file stream.

Summary:

The user interacts with the GUI to modify the data in database(txt). However, it is not reasonable to modify the data directly through the GUI. So when we want to modify the data. We will use the method in Service to call methods in the Dao. At this point, the Service works like a controller.

6.2 Testing

6.2.1 Testing Strategy

- 80% of the tests will be automatically executed while the other will be manually run.
- Each subject use case will be tested for its normal behavior and also two alternative ones. And make sure that they also have wrong input.
- Success standard – 85% passed.

6.2.2 Testing environment

- JDK version 11.0.4 & 12.0.2
- Windows 10

6.2.3 Testing Techniques

1. Regression Testing

Regression testing is a type of software testing that ensures that previously developed and tested software still performs the same way after it is changed or interfaced with other software. For integration testing, test cases are created to test the function of each build version. These test cases will be tested at each iteration and compose what are known as Regression Tests. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.

2. White Box Testing

White-box testing is to test the internal program logic.

An example with CustomerServiceImpl.java

Test Class	Test Case 1	Test Case 2	Test Case 3
Function	chkCustomer(Customer customer)	setCustomerMsg(Customer customer)	saveCustomer(Customer customer)
Conditions & Results	If JudgeCode is not '000', the value is false; otherwise, return true.	if the uid is XXABCD (e.g. JC0001), the value is true; otherwise it is false.	If the return value is 0, the 'save' is false; otherwise it is true.

Test Case 1	Test Description	Expected Result	Actural Result	No. of Bugs	Bugs #	Comments
chkCustomer(Customer customer)	850475@qq.com	True	True	0	No bugs here	"850475@qq.com" meets the requirements
chkCustomer(Customer customer)	850475@qq.c.om	False	False	0	No bugs here	850475@qq.c.om is not the right form.
chkCustomer(Customer customer)	850475.qq.com	False	False	0	No bugs here	There's no "@" in the string.

3. Black Box Testing

Black box testing is a software testing method that checks the function of an application without checking its internal structure or working method. We tried to find the missing function or the incorrect function. We mainly use partition testing and scenario-based testing to check whether the requirements are met.

An example with Customer.java

Test Class	Test Case 1	Test Case 2	Test Case 3
Function	getUid()	getFirname()	getSurname()
Conditions & Results	Customer's uid should return	Customer's firstname should return	Customer's surname should return

Test Case 3	Test Description	Expected Result	Actural Result	No. of Bugs	Bugs #	Comments
getUid()	No input	JC0001	JC0001	0	No	Correct Uid is returned

6.2.4 TDD

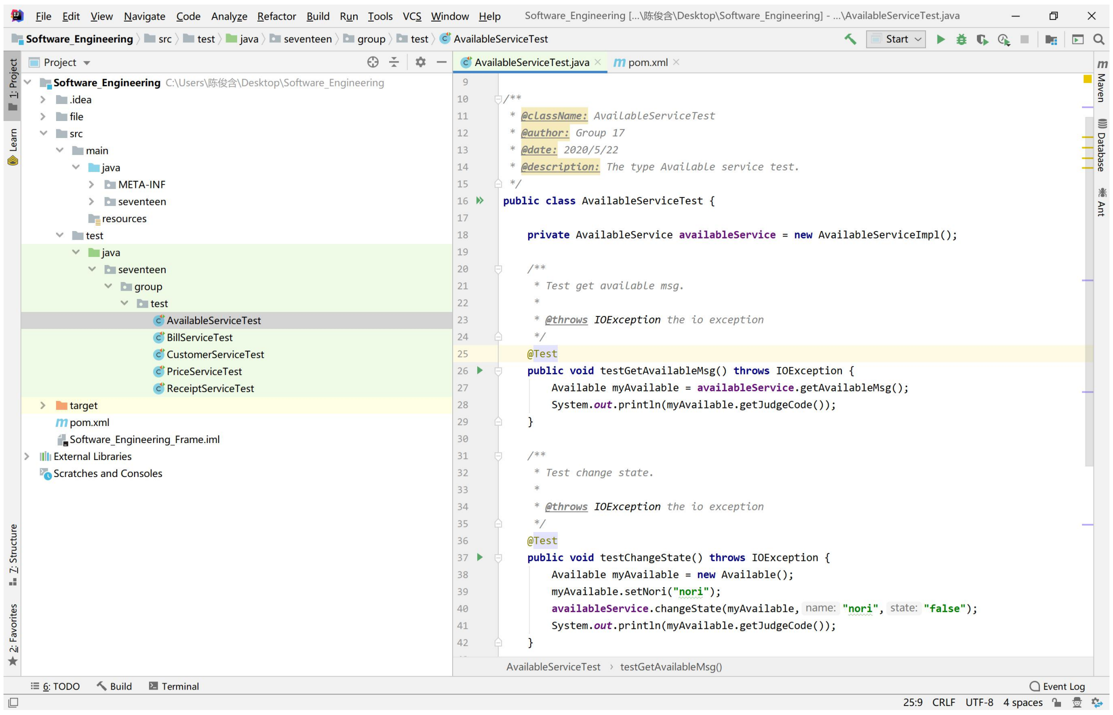
"Test-driven development" refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring). TDD is to write tests prior to write the production code. It is a simple and short cycled mechanism.

Set of rules:

- Write a specification, in code and in the form of a unit test. The test verifies a functional unit of your code.
- Demonstrate test failure.
- Write code to meet the specification.
- Demonstrate test success.
- Refactor the code, to ensure that the system still has an optimally clean code base. Run all tests against the entire system at all time.

6.2.5 Junit Testing

We took all classes in Service layer as JUnit testing objects.



```java
/**
 * @className: AvailableServiceTest
 * @author: Group 17
 * @date: 2020/5/22
 * @description: The type Available service test.
 */
public class AvailableServiceTest {

    private AvailableService availableService = new AvailableServiceImpl();

    /**
     * Test get available msg.
     *
     * @throws IOException the io exception
     */
    @Test
    public void testGetAvailableMsg() throws IOException {
        Available myAvailable = availableService.getAvailableMsg();
        System.out.println(myAvailable.getJudgeCode());
    }

    /**
     * Test change state.
     *
     * @throws IOException the io exception
     */
    @Test
    public void testChangeState() throws IOException {
        Available myAvailable = new Available();
        myAvailable.setNori("nori");
        availableService.changeState(myAvailable, name: "nori", state: "false");
        System.out.println(myAvailable.getJudgeCode());
    }
```

7. Conclusion

Our project lasted two months and finally we were able to meet all the requests of the basic documents and our observations. During the development process, we first convened several meetings to discuss the needs. Then we completed the list of products to buy and continued to update it. We then started writing Java code for basic system functions and GUI (UI) design. We've divided all codes into entity classes, control classes, and boundary classes. The control class includes Dao and Service, both of which are implemented by interfaces. Boundary class mainly refers to the graphical visual operation interface of the system Finally, we managed the test classes and checked if there were any issues to improve our system.

References

1. https://www.pmi.org/about/learn-about-pmi/what-is-project-management

2. Harold Kerzner(2006). *Project Management: A System Approach to Planining, Schedululing, and Controlling*, doi:10.1111/j.1460-9568.2005.04381.x

3. SEBok authors,(2013).*A guide to the project management body of knowledge*, doi:10.1002/pmj.20125

4. Alistair Cockburn,(2000). *Agile Software Development*, doi: 10.1007/978-3-319-05008-9_1

5. Agile Manifesto,(2009).*Manifesto for Agile Software Development*, doi:10.1007/978-3-540-27777-4_53

6. An introduction to the SOLID principles of OO design ***http://www.davesquared.net/2009/01/introduction-to-solid-principles-of-oo.html***

7. LSP: The Liskov Substitution Principle by Robert C. Martin "Uncle Bob" ***http://www.objectmentor.com/resources/articles/lsp.pdf***

8. Lectures from QM EBU6304-Software Engineering–Analysis and Design ***https://qmplus.qmul.ac.uk/pluginfile.php/1320468/mod_resource/content/5/EBU6304_05_Analysis_Design.pdf***

9. Lectures from QM EBU6304-Software Engineering – TDD ***https://qmplus.qmul.ac.uk/pluginfile.php/1320470/mod_resource/content/5/EBU6304_08_TDD.pdf***

10. Lectures from QM EBU6304-Software Engineering –Design Principles ***https://qmplus.qmul.ac.uk/pluginfile.php/1577617/mod_resource/content/21/EBU6304_2019.20_DesignPrinciples.pdf***

Appendix

Appendix A: Class Design

AvailableDaoImpl		Description
Operation	getAvailableMsg()	Get the available message from available.txt
	setAvailableMsg(Available available)	Set the available message to available.txt
	changeState(Available available, String name, String state)	Change the available state of one choice
	searchState(Available available, String name)	Search the available state of one choice
Association	AvailableServiceImpl Class	

BillDaoImpl		Description
Operation	saveBillMsg(Bill bill)	Set the bill message to bill.txt
	getStatistic()	Get the statistic message from bill.txt
Association	BillServiceImpl Class	

CustomerDaoImpl		Description
Operation	setCustomerMsg(Customer customer)	Set the basic message to a customer object
	saveCustomer(Customer customer	Set the customer message to customer.txt
	getCustMsg(String uid)	Get the customer message to customer.txt
	chkExist(Customer customer)	Check whether this customer has already existed
Association	CustomerServiceImpl Class	

PriceDaoImpl		Description
Operation	setPrice(Price price)	Set the price message to price.txt
	getPrice()	Get the available message from price.txt
	searchPrice(Price price, String name)	Search the price of one choice
	changePrice(Price price, String name, double change)	Change the price of one choice
Association	PriceServiceImpl Class	

AvailableServiceImpl		**Description**
Attribute	AvailableDao availableDao	Get the method in AvailableDao so that we could only use AvailableServiceImpl
Operation	getAvailableMsg()	Get the available message from available.txt
	setAvailableMsg(Available available)	Set the available message to available.txt
	changeState(Available available, String name, String state)	Change the available state of one choice
	searchState(Available available, String name)	Search the available state of one choice
Association	GUI Class	

BillServiceImpl		**Description**
Attribute	BillDao billDao	Get the method in BillDao so that we could only use BillServiceImpl
Operation	chkBillMsg(Bill bill)	Check whether the bill is set correctly
	setBillMsg(Bill bill)	Set the message in a bill object
	getBillMsg(Bill bill)	Generate a string which contains bill msg
	getSelfMost(Statistics statistics, String string)	Get the most popular meal
	saveBillMsg(Bill bill)	Set the bill message to bill.txt
	getStatistic()	Get the statistic message from bill.txt
Association	GUI Class	

CustomerServiceImpl		**Description**
Attribute	CustomerDao customerDao	Get the method in CustomerDao so that we could only use CustomerServiceImpl
Operation	chkCustomer(Customer customer)	Check whether the message in a customer object is correct
	setCustomerMsg(Customer customer)	Set the basic message to a customer object
	saveCustomer(Customer customer	Set the customer message to customer.txt
	getCustMsg(String uid)	Get the customer message to customer.txt
	chkExist(Customer customer)	Check whether this customer has already existed
Association	GUI Class	

PriceServiceImpl		**Description**
Attribute	PriceDao priceDao	Get the method in PriceDao so that we could only use PriceServiceImpl

Operation	chgPriceData(Price price)	Change the price data format
	chkPriceData(String price)	Check whether the message in a price object is correct
	setPrice(Price price)	Set the price message to price.txt
	getPrice()	Get the available message from price.txt
	searchPrice(Price price, String name)	Search the price of one choice
	changePrice(Price price, String name, double change)	Change the price of one choice
Association	GUI Class	

ReceiptDaoImpl		**Description**
Attribute	ReceiptDao receiptDao	Get the method in ReceiptDao so that we could only use ReceiptServiceImpl
Operation	setReceipt(Bill bill, Customer customer)	Set the receipt message to receipt.txt
Association	GUI Class	

Available		**Description**
Attribute	String soupTonkotsu	The availability of Tonkotsu
	String soupShoyu	The availability of Shoyu
	String soupShio	The availability of Shio
	String noodleSoft	The availability of Soft noodle
	String noodleMedium	The availability of Medium noodle
	String noodleFirm	The availability of Firm noodle
	String onion	The availability of Onion
	String nori	The availability of Nori
	String chashu	The availability of Chashu
	String boiledEgg	The availability of BoiledEgg
	String spiciness	The availability of Spiciness
	String bambooShoot	The availability of BambooShoot
	String ExBoiledEgg	The availability of Extra Boiled Egg
	String ExChashu	The availability of Extra Chashu
	String ExNori	The availability of Extra Nori
	String judgeCode	The string to judge the message
Operation	getter()	The accessor to get the message
	Setter(Object object)	The mutator to set the message
Association	Service Class, Dao Class	

ReceiptDaoImpl		Description
Operation	setReceipt(Bill bill, Customer customer)	Set the receipt message to receipt.txt
Association	ReceiptServiceImpl Class	

Bill		Description
Attribute	String uid	Object Bill's user id
	String soup	The user's choice of Soup
	String nood	The user's choice of Noodle
	String onion	The user's choice of Onion
	String nori	The user's choice of Nori
	String chashu	The user's choice of Chashu
	String boiledEgg	The user's choice of BoiledEgg
	String spiciness	The user's choice of Spiciness
	String bambooShoot	The user's choice of BambooShoot
	String extraEgg	The user's choice of Extra Boiled Egg
	String extraChashu	The user's choice of Extra Chashu
	String extraNori	The user's choice of Extra Nori
	String ifFree	Whether the bill is free
	String price	The total price of the bill
	String payoption	The pay option of the bill
	String date	The date of the bill
	String packet	The packet of the bill
Operation	getter()	The accessor to get the message
	Setter(Object object)	The mutator to set the message
Association	Service Class, Dao Class	

Customer		Description
Attribute	String uid	The uid of the customer
	String firname	The first name of the customer
	String surname	The last name of the customer
	String email	The email of the customer
	String mobile	The mobile of the customer
	Int stamp	The stamp of customer
	String msgChk	Check the message of customer
	String judgeCode	Judge the correctness of customer
Operation	getter()	The accessor to get the message
	Setter(Object object)	The mutator to set the message
Association	Service Class, Dao Class	

MostPopMeal		Description
Attribute	int value	The choice number of the popular meal
	String name	The name of the most popular meal

Operation	getter()	The accessor to get the message
	Setter(Object object)	The mutator to set the message
Association	Service Class, Dao Class	

Price		**Description**
Attribute	Double ramen	The price of Ramen
	Double exNori	The price of extra Nori
	Double exEgg	The price of extra Egg
	Double exBamboo	The price of extra BambooShoot
	Double exChashu	The price of extra Chashu
	String judgeCode	Judge the price is correct
	Int chkDiscount	Check the discount of the price
Operation	getter()	The accessor to get the message
	Setter(Object object)	The mutator to set the message
Association	Service Class, Dao Class	

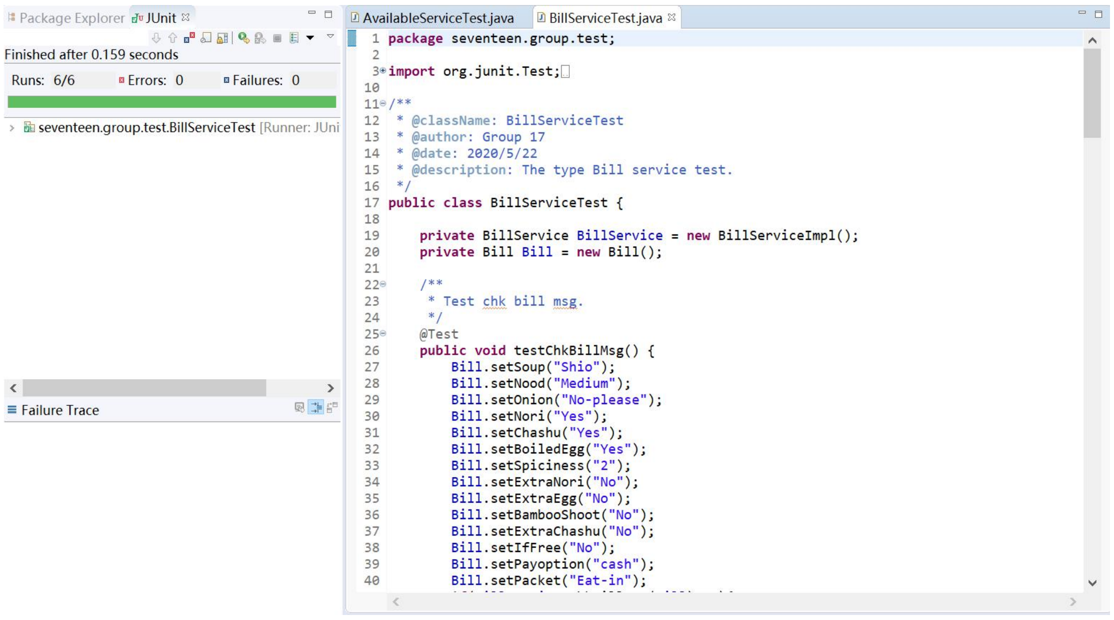
Appendix B: Testing Cases

Test Case 1: Member Register	Test Description	Test Cases	Expected Result	Actural Result	Bugs	Comments
N/A	In the member registration, users need to input information of firstname, lastname, mobile phone number and email.	setup				
1.1	Input of firstname/lastname should not contain number and punctuation.	Jun2han	Input invalid	Input invalid	0	Correct behaviour observed
		Jun.han	Input invalid	Input invalid	0	Correct behaviour observed
1.2	The length of phone number should not longer than 15	123456789012345	Input valid	Input valid	0	Edge test passed
		1234567890123456	Input invalid	Input invalid	0	Correct behaviour observed
		123	Input valid	Input valid	0	Correct behaviour observed
1.3	The format of mail input should like xxx@xx.xx. Only one @ is allowed. Except one full stop, no other punctuation is allowed after @.	Han@qq.com	Input valid	Input valid	0	Correct behaviour observed
		Han@han@qq.com	Input invalid	Input invalid	0	Correct behaviour observed
		Han@qq.com!com	Input invalid	Input invalid	0	Correct behaviour observed
1.4	The email and mobile phone Numbers entered cannot be the same as those already registered. And firstname /lastname has no such restriction	Input same phone	User exist	User exist	0	Correct behaviour observed
		Input same e-mail	User exist	User exist	0	Correct behaviour observed
		Input same firstname/last name	No notification	No notification	0	Correct behaviour observed

Test Case 2: Member Login	Test Description	Test Cases	Expected Result	Actural Result	Bugs	Comments
N/A	Users need to enter a registered user ID when they log in	setup				
1.1	The input user ID should be registered.	User ID has registered	Can log in	Can log in	0	Correct behaviour observed
		User ID hasn't registered	Can't log in	Can't log in	0	Correct behaviour observed
1.2	The User id case should be insensitive.(In this test, the existed user ID should be AB123)	Ab123	Can log in	Can't log in	**1**	**Already modified in later iteration**
		ab123	Can log in	Can't log in	**1**	**Already modified in later iteration**
		AB123	Can log in	Can log in	0	Correct behaviour observed

Test Case 3: Check and modify menu	Test Description	Test Cases	Expected Result	Actural Result	Bugs	Comments
N/A	Manager can check the status of menu and modify it.	setup				
1.1	The data that manager sees should as same as the actual menu	No input	Data correct	Data correct	0	Correct behaviour observed
1.2	If the price and status are changed by the manager, the menu will be changed accordingly.	Change status	Menu changed	Menu changed	0	Correct behaviour observed
		Change price	Menu changed	Menu changed	0	Correct behaviour observed
1.3	Can only enter integers or decimals when modifying the price. Multiple dots are not allowed	10.0	Input valid	Input valid	0	Correct behaviour observed
		10@!	Input invalid	Input invalid	0	Correct behaviour observed
		10.0.0	Input valid	Input valid	0	Correct behaviour observed
1.4	The price of food should be halve on Thursday. (It's just a Eggshell for our software!)	On Thursday	50% discount	50% discount	0	Correct behaviour observed
		Not Thursday	Normal	Normal	0	Correct behaviour observed

Appendix C: JUnit Testing



Package Explorer JUnit

Finished after 0.159 seconds

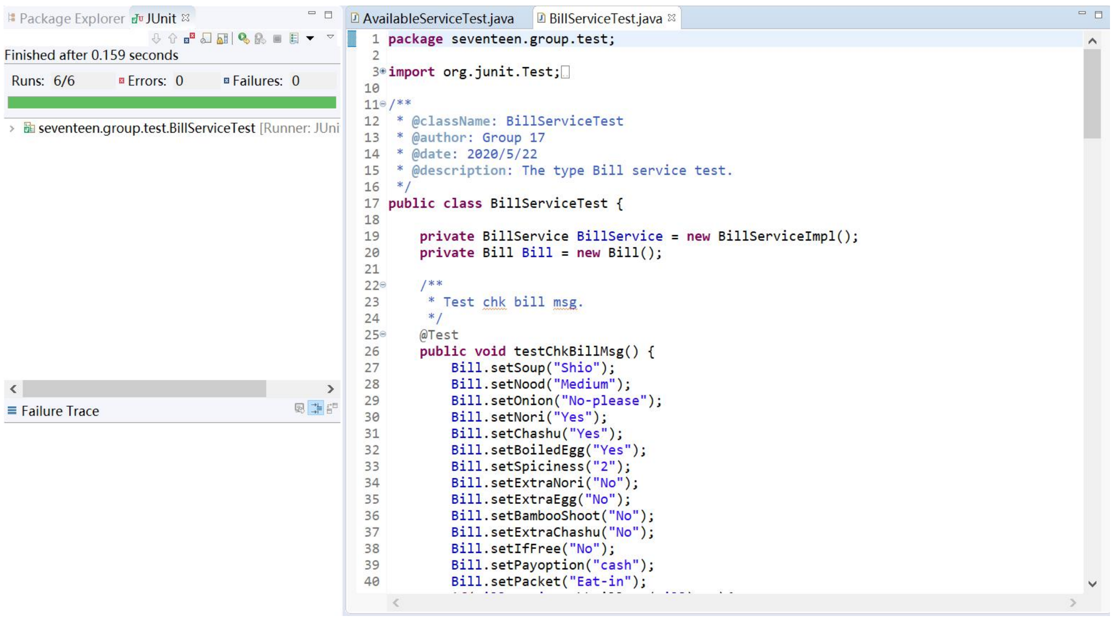
Runs: 6/6 Errors: 0 Failures: 0

seventeen.group.test.BillServiceTest [Runner: JUni

Failure Trace

AvailableServiceTest.java BillServiceTest.java

```java
package seventeen.group.test;

import org.junit.Test;

/**
 * @className: BillServiceTest
 * @author: Group 17
 * @date: 2020/5/22
 * @description: The type Bill service test.
 */
public class BillServiceTest {

    private BillService BillService = new BillServiceImpl();
    private Bill Bill = new Bill();

    /**
     * Test chk bill msg.
     */
    @Test
    public void testChkBillMsg() {
        Bill.setSoup("Shio");
        Bill.setNood("Medium");
        Bill.setOnion("No-please");
        Bill.setNori("Yes");
        Bill.setChashu("Yes");
        Bill.setBoiledEgg("Yes");
        Bill.setSpiciness("2");
        Bill.setExtraNori("No");
        Bill.setExtraEgg("No");
        Bill.setBambooShoot("No");
        Bill.setExtraChashu("No");
        Bill.setIfFree("No");
        Bill.setPayoption("cash");
        Bill.setPacket("Eat-in");
```



Package Explorer JUnit

Finished after 0.159 seconds

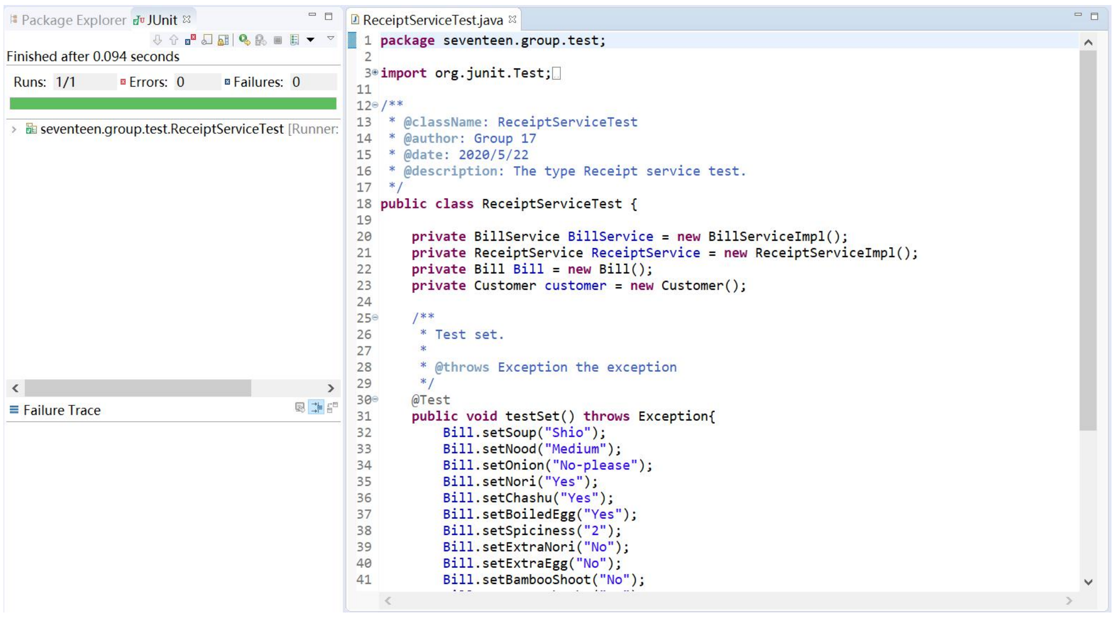
Runs: 6/6 Errors: 0 Failures: 0

seventeen.group.test.BillServiceTest [Runner: JUni

Failure Trace

AvailableServiceTest.java BillServiceTest.java

```java
package seventeen.group.test;

import org.junit.Test;

/**
 * @className: BillServiceTest
 * @author: Group 17
 * @date: 2020/5/22
 * @description: The type Bill service test.
 */
public class BillServiceTest {

    private BillService BillService = new BillServiceImpl();
    private Bill Bill = new Bill();

    /**
     * Test chk bill msg.
     */
    @Test
    public void testChkBillMsg() {
        Bill.setSoup("Shio");
        Bill.setNood("Medium");
        Bill.setOnion("No-please");
        Bill.setNori("Yes");
        Bill.setChashu("Yes");
        Bill.setBoiledEgg("Yes");
        Bill.setSpiciness("2");
        Bill.setExtraNori("No");
        Bill.setExtraEgg("No");
        Bill.setBambooShoot("No");
        Bill.setExtraChashu("No");
        Bill.setIfFree("No");
        Bill.setPayoption("cash");
        Bill.setPacket("Eat-in");
```



Package Explorer JUnit

Finished after 0.094 seconds

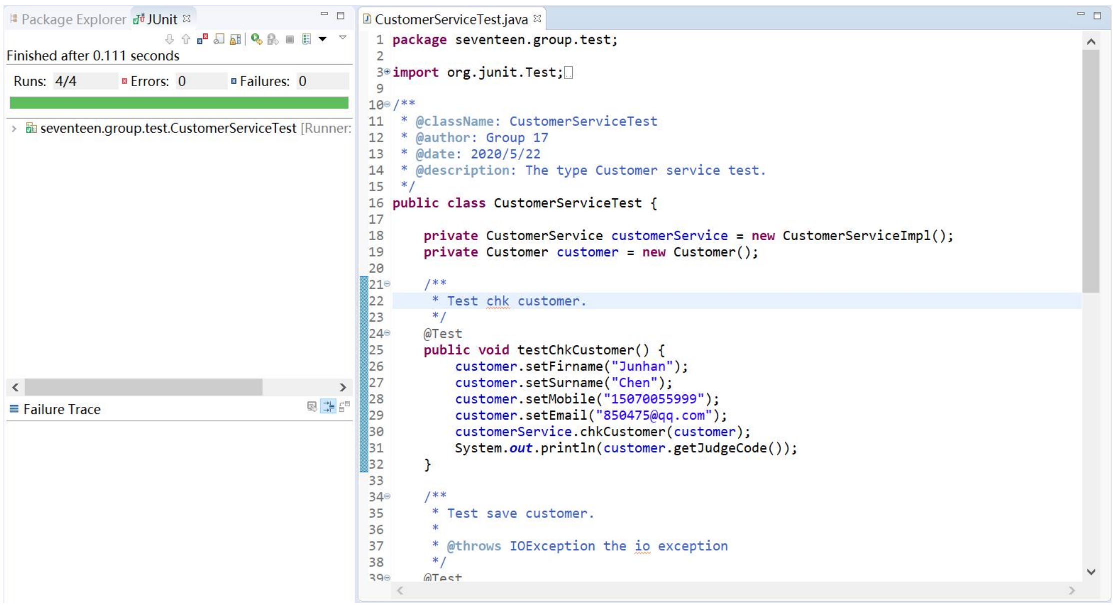
Runs: 1/1 Errors: 0 Failures: 0

seventeen.group.test.ReceiptServiceTest [Runner:

Failure Trace

ReceiptServiceTest.java

```java
package seventeen.group.test;

import org.junit.Test;

/**
 * @className: ReceiptServiceTest
 * @author: Group 17
 * @date: 2020/5/22
 * @description: The type Receipt service test.
 */
public class ReceiptServiceTest {

    private BillService BillService = new BillServiceImpl();
    private ReceiptService ReceiptService = new ReceiptServiceImpl();
    private Bill Bill = new Bill();
    private Customer customer = new Customer();

    /**
     * Test set.
     *
     * @throws Exception the exception
     */
    @Test
    public void testSet() throws Exception{
        Bill.setSoup("Shio");
        Bill.setNood("Medium");
        Bill.setOnion("No-please");
        Bill.setNori("Yes");
        Bill.setChashu("Yes");
        Bill.setBoiledEgg("Yes");
        Bill.setSpiciness("2");
        Bill.setExtraNori("No");
        Bill.setExtraEgg("No");
        Bill.setBambooShoot("No");
```



Package Explorer JUnit

Finished after 0.111 seconds

Runs: 4/4 Errors: 0 Failures: 0

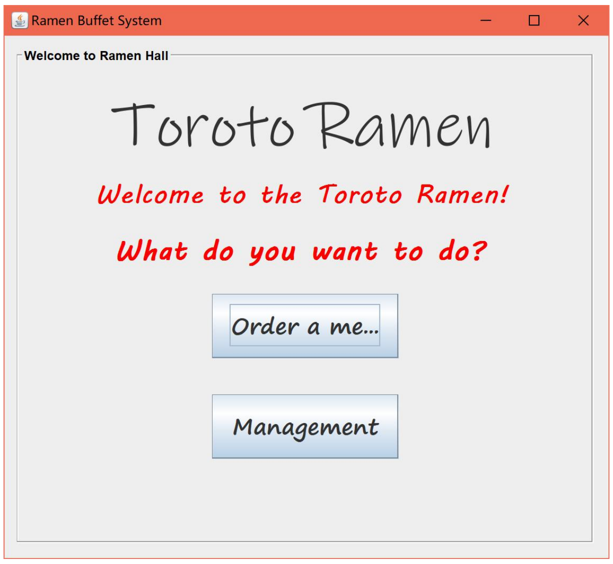
seventeen.group.test.CustomerServiceTest [Runner:

Failure Trace

CustomerServiceTest.java

```java
package seventeen.group.test;

import org.junit.Test;

/**
 * @className: CustomerServiceTest
 * @author: Group 17
 * @date: 2020/5/22
 * @description: The type Customer service test.
 */
public class CustomerServiceTest {

    private CustomerService customerService = new CustomerServiceImpl();
    private Customer customer = new Customer();

    /**
     * Test chk customer.
     */
    @Test
    public void testChkCustomer() {
        customer.setFirname("Junhan");
        customer.setSurname("Chen");
        customer.setMobile("15070055999");
        customer.setEmail("850475@qq.com");
        customerService.chkCustomer(customer);
        System.out.println(customer.getJudgeCode());
    }

    /**
     * Test save customer.
     *
     * @throws IOException the io exception
     */
    @Test
```

Appendix D: Major Screenshots of The System

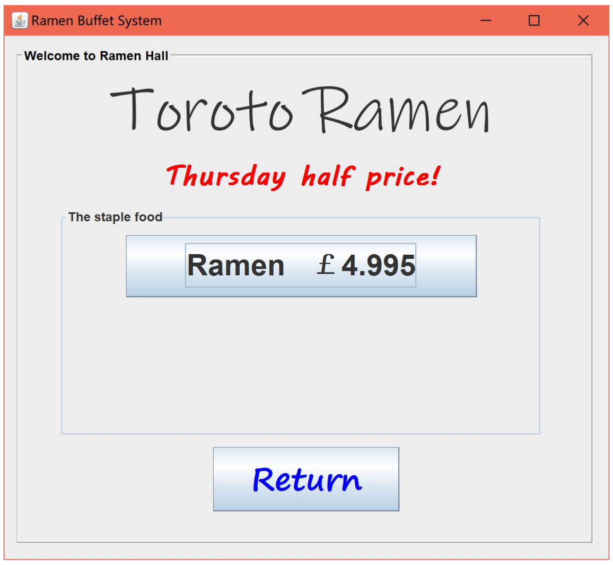

Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

Welcome to the Toroto Ramen!

What do you want to do?

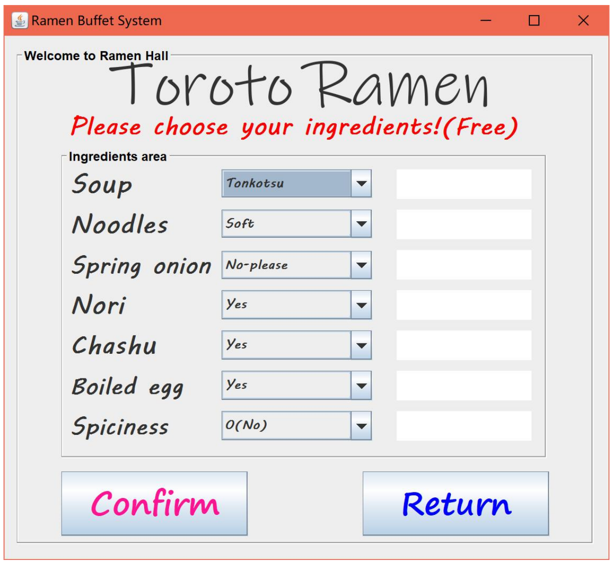

Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

Thursday half price!

The staple food


Ramen Buffet System

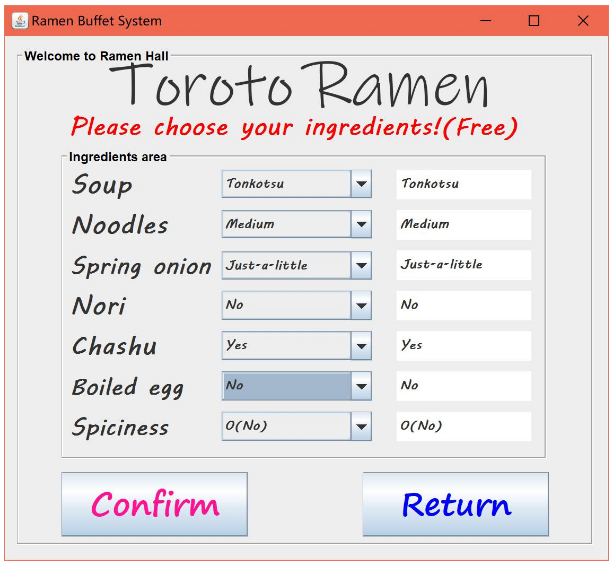
Welcome to Ramen Hall

Toroto Ramen

Please choose your ingredients!(Free)

Ingredients area

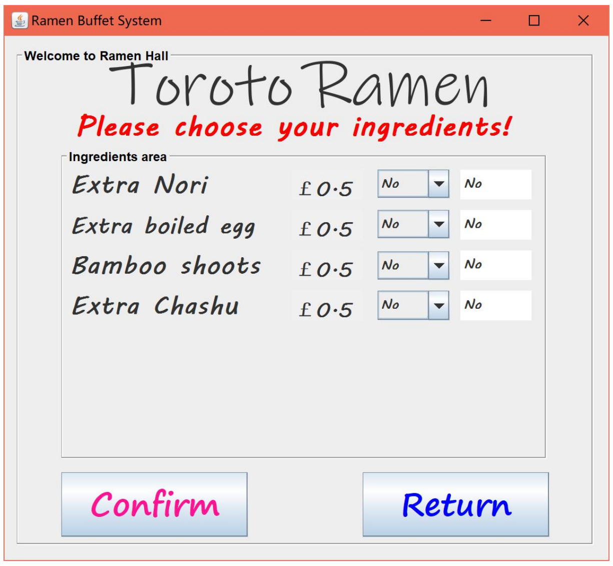
Soup	Tonkotsu
Noodles	Soft
Spring onion	No-please
Nori	Yes
Chashu	Yes
Boiled egg	Yes
Spiciness	0(No)



Please choose your ingredients!(Free)

Ingredients area

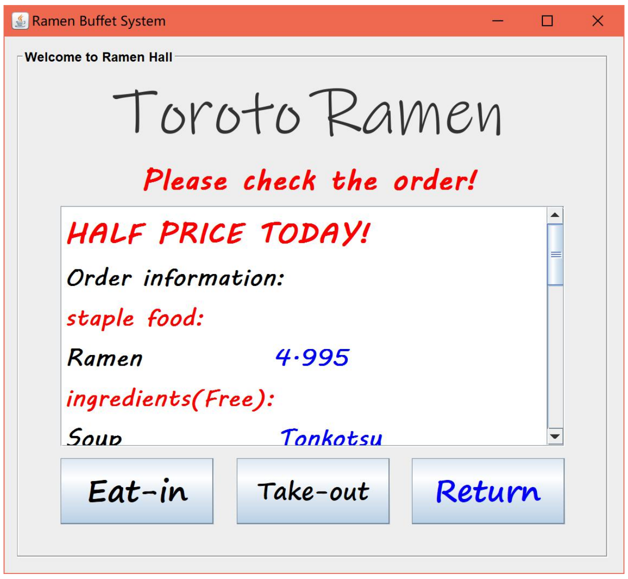
Soup	Tonkotsu	Tonkotsu
Noodles	Medium	Medium
Spring onion	Just-a-little	Just-a-little
Nori	No	No
Chashu	Yes	Yes
Boiled egg	No	No
Spiciness	0(No)	0(No)



Please choose your ingredients!

Ingredients area

Extra Nori	£0·5	No	No
Extra boiled egg	£0·5	No	No
Bamboo shoots	£0·5	No	No
Extra Chashu	£0·5	No	No



Please check the order!

HALF PRICE TODAY!

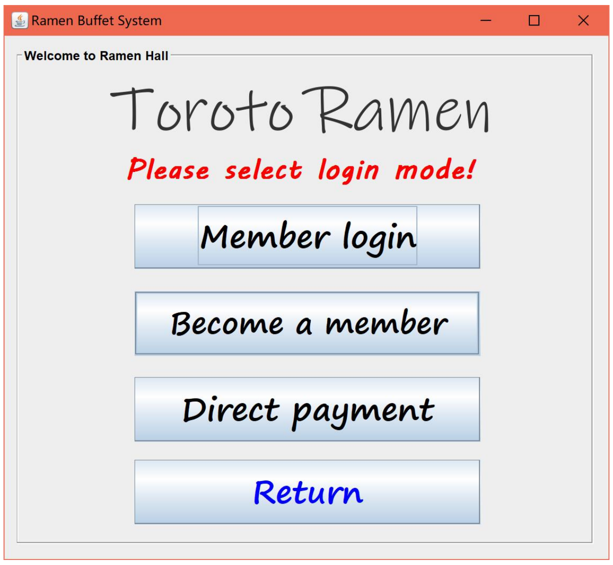
Order information:

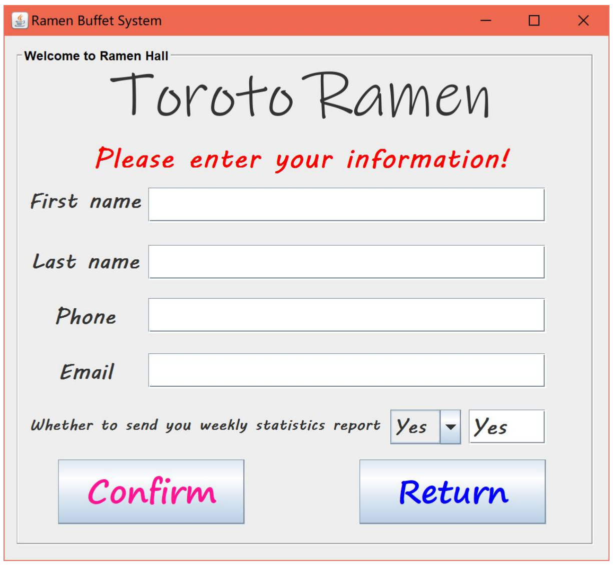
staple food:

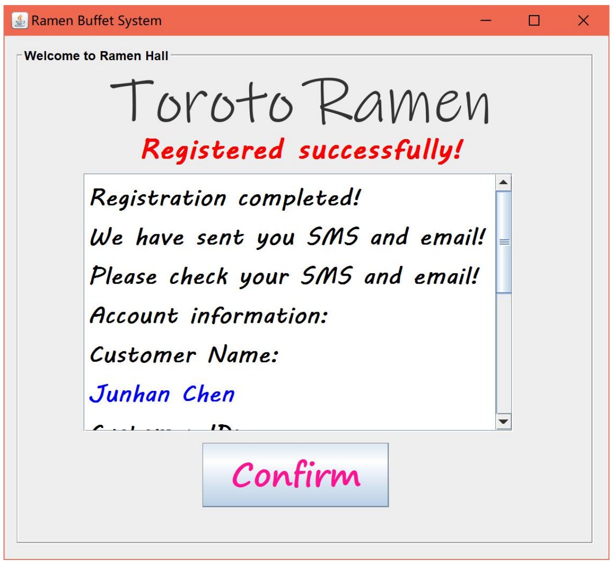
Ramen 4·995

ingredients(Free):

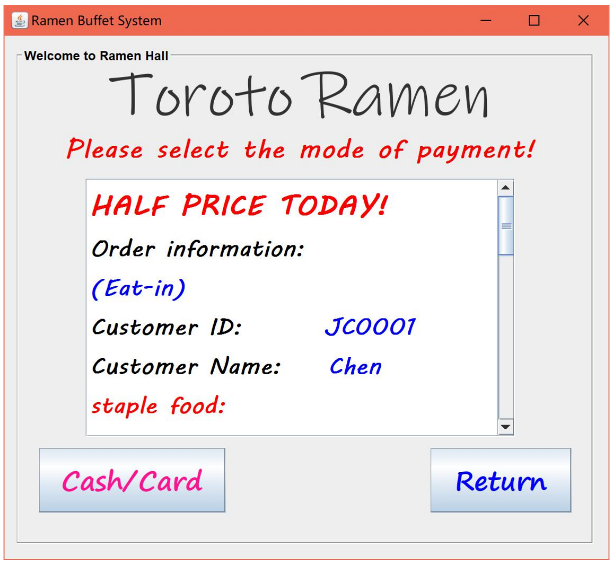
Soup Tonkotsu







Registration completed!
We have sent you SMS and email!
Please check your SMS and email!
Account information:
Customer Name:
Junhan Chen

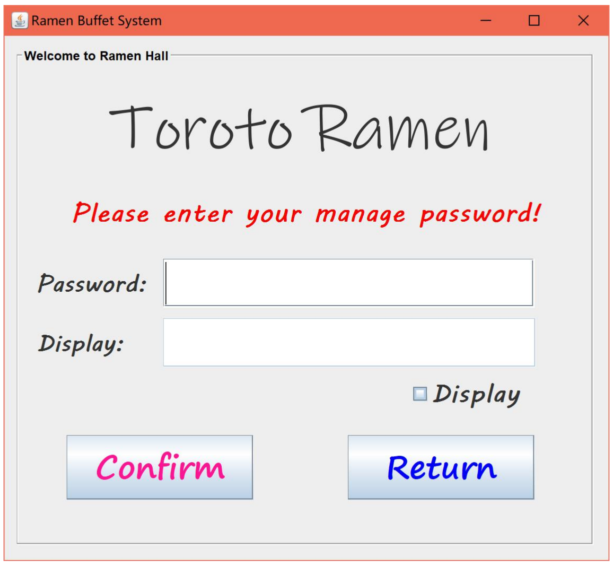

Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

Please select the mode of payment!

HALF PRICE TODAY!
Order information:
(Eat-in)
Customer ID: JC0001
Customer Name: Chen
staple food:

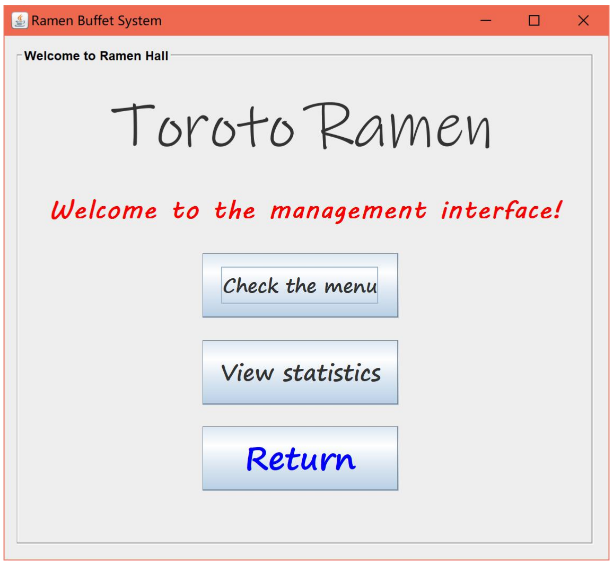

Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

Please enter your manage password!

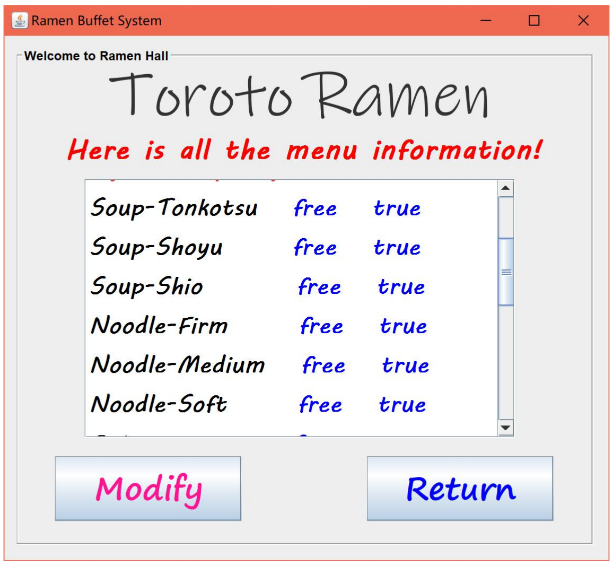
Password:
Display:


Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

Welcome to the management interface!


Ramen Buffet System

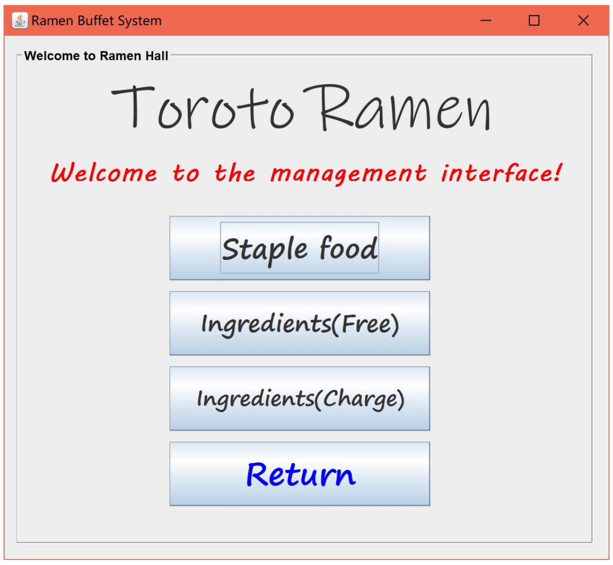
Welcome to Ramen Hall

Toroto Ramen

Here is all the menu information!

Soup-Tonkotsu	free	true
Soup-Shoyu	free	true
Soup-Shio	free	true
Noodle-Firm	free	true
Noodle-Medium	free	true
Noodle-Soft	free	true

Modify Return


Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

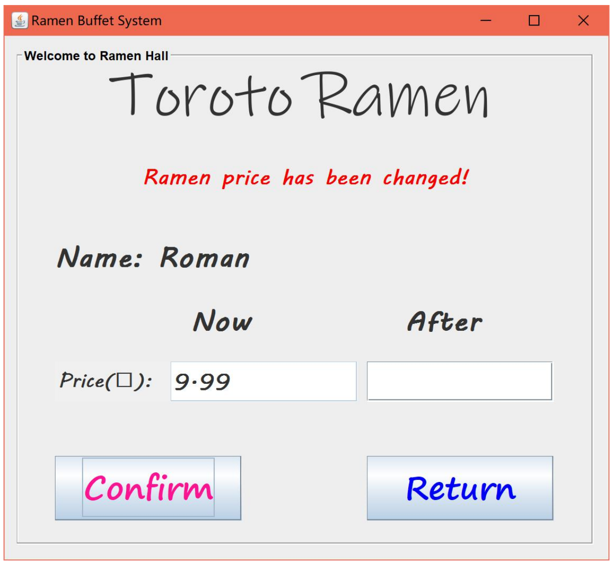
Welcome to the management interface!

Staple food

Ingredients(Free)

Ingredients(Charge)

Return


Ramen Buffet System

Welcome to Ramen Hall

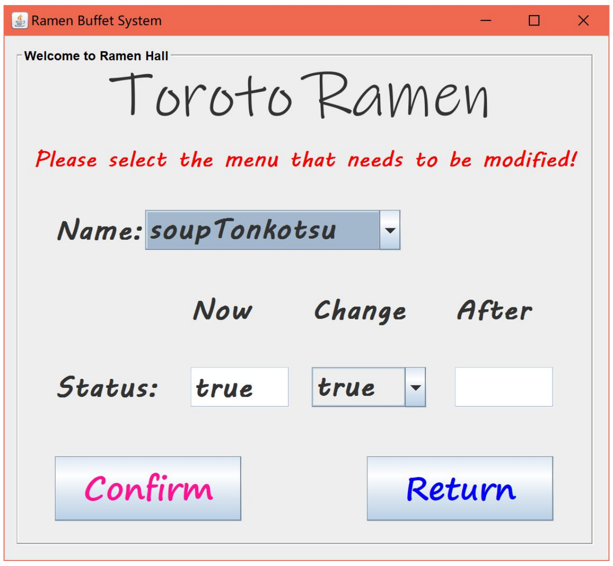
Toroto Ramen

Ramen price has been changed!

Name: Roman

	Now	After
Price(□):	9.99	

Confirm Return


Ramen Buffet System

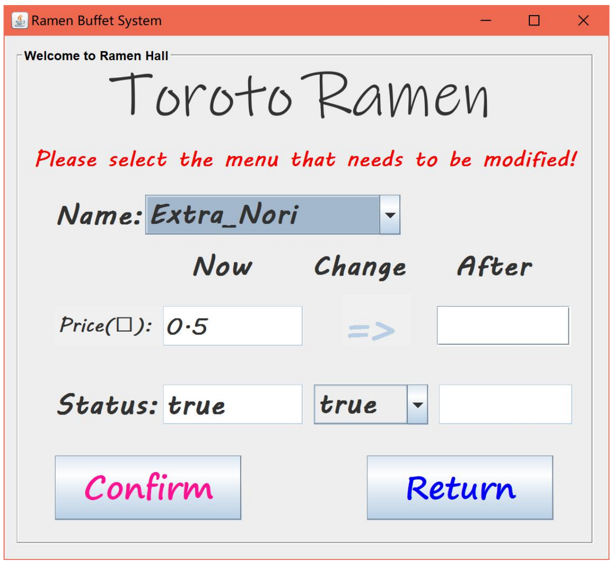
Welcome to Ramen Hall

Toroto Ramen

Please select the menu that needs to be modified!

Name: soupTonkotsu

	Now	Change	After
Status:	true	true	


Ramen Buffet System

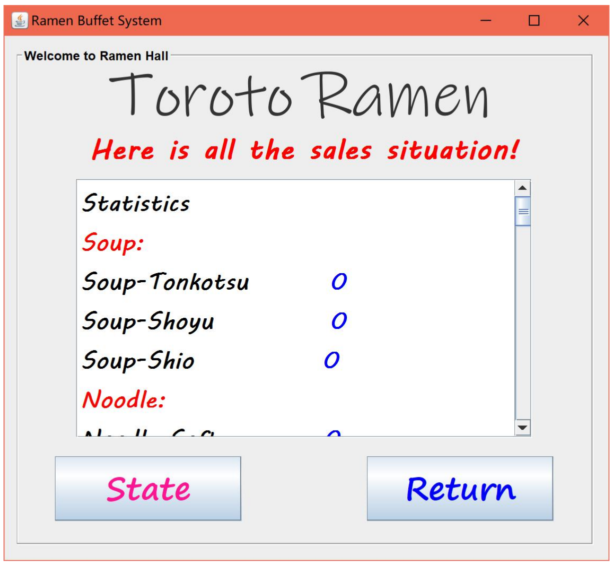
Welcome to Ramen Hall

Toroto Ramen

Please select the menu that needs to be modified!

Name: Extra_Nori

	Now	Change	After
Price(□):	0.5	=>	
Status:	true	true	

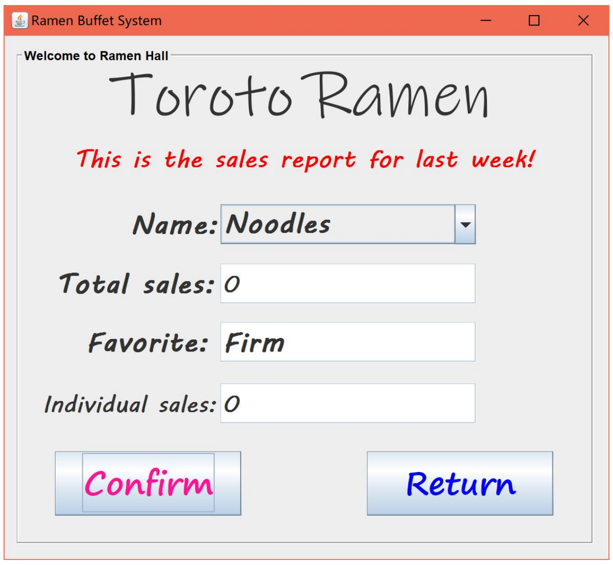

Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

Here is all the sales situation!

```
Statistics
Soup:
Soup-Tonkotsu     0
Soup-Shoyu        0
Soup-Shio         0
Noodle:
```



Ramen Buffet System

Welcome to Ramen Hall

Toroto Ramen

This is the sales report for last week!

Name: Noodles

Total sales: 0

Favorite: Firm

Individual sales: 0