South East Technological University

Department of Computing and Mathematics

official record of school work

Department of Computing and Mathematics

CA Work

Programme that you are enrolled on:	Baking Information System
Programme Module	Data Structures and Algorithms 1
Title of Work Submitted (e.g Lab Report 1)	Design and Implementation of a Baking Information System in Java
Lecturers	Guoqing Lu
ID Number	202283890020 202283890029 202283890027 202283890036
Class Group (number)	17

Filename: e.g.GroupNumber_DS&A1_Project_1.pdf	17_DS&A1_project2.pdf
Acknowledgements (if you collaborated as a team then acknowledge colleagues)	Thanks to everyone on the team, including Ziheng WangKaifeng Jin，Weichen Guo，and Jiarui Huang. And our team thanks Mr.Lu for his help.

Declaration

I hereby declare that this is my original work produced without the help of any third party unless where referenced within this report.

Student: Kaifeng Jin，Ziheng Wang，Weichen Guo，Jiarui Huang.

Date and Time of Submission: 26th. Nov. 2023

Table of Contents

Design and Implementation of a Baking Information System in Java

Abstract

The objective of this exercise is to develop a ”Baking Information System” using Java, aimed at enabling users to efficiently add, search, and view various baked goods through a **Graphical User Interface** (GUI). This system includes information such as ingredients, place of origin, images, and calorie content for each baked item. Search functionalities encompass criteria such as name or keywords, providing alphabetical and caloricsorting.

The system also incorporates a degree of interactivity to obtain additional details, enhancing user experience. For instance, a search for baked goods containing a specific ingredient will yield a list of all items featuring that ingredient. Furthermore, clicking on a specific item in the list will reveal detailed information about that baked good, including the ingredients list.

To implement these functionalities, our team has concurrently employed **Hash tables** and **doubly linked lists** as data structures for storage, search, and other functionalities. The doubly linked list's structure facilitates fuzzy searching and the implementation of features listing all contents. The Hash table, on the other hand, is utilized for swift searches based on names, effectively reducing algorithmic time complexity. Additionally, an interactive GUI has been created to ensure user-friendly and efficient system usage.

Within the doubly linked list, we have implemented **comparators** to enable **quick sorting** based on different parameters, such as ascending or descending order, when using the ”description” parameter for fuzzy search results. This ensures that results are presented in descending order based on the match level with the inputted search content.

Finally, algorithmic **complexity analysis** has been conducted, and extensive testing with randomly generated data has been performed to evaluate the program's responsiveness and stability. The feedback received indicates a robust and efficient system.

Key words: **Graphical User Interface**, **Hash tables**, **doubly linked lists**, **comparators**, **complexity analysis**

1 Purpose and Objectives

1.1 Background Introduction

The objective of this exercise is to develop a ”Bakery Information System” using Java, with a stringent emphasis on refraining from utilizing existing Java sorting or searching methods, as well as pre-existing Java collections or data structures. The primary aim of this Bakery Information System is to empower users to seamlessly add, search, and visualize a diverse array of baked goods through a Graphical User Interface (GUI). This encompasses details such as ingredient composition, place of origin, images, and caloric content. Search functionalities may be based on criteria such as name or keywords, with an additional provision for alphabetical and caloric sorting.

The system is also required to facilitate a certain level of interactivity, allowing users to drill down for additional details. For instance, a search for baked goods containing a specific ingredient should yield a comprehensive list of all baked goods featuring that particular ingredient. Subsequently, clicking on an item within the list should provide detailed information about that specific baked good, including an itemized list of ingredients.

The challenge lies in effectively capturing and organizing the intricate relationships within various classes of baked goods, ensuring seamless and efficient management of the bakery.

1.2 Project Goals and Objectives

1.2.1 Efficient Hierarchy and Data Structure

Design a system architecture with custom data structures for streamlined management of baked Good,ingredient,recipe.

1.2.2 User-Friendly Operations

Enable users to perform key operations, including the addition of baked goods,ingredient,recipe, as, ensuring a user-friendly interface.

1.2.3 Inventory Visibility and Retrieval

Implement robust inventory visibility, allowing users to view and navigate through the entire stock. Include a powerful search function for efficient item retrieval.

1.2.4 Comprehensive System Control

Develop facilities for item removal, systematic valuation of inventory, and system reset. Ensure persistence through save and load functions.

1.3 Improving Operational Efficiency

In order to enhance the operational efficiency of the Baking Information System, we have implemented a user-friendly graphical user interface (GUI) interaction feature. This strategic integration facilitates convenient and efficient operations such as adding and removing jewelry, significantly elevating the overall productivity of the jewelry store management system.

2 Design Record

2.1 Overall Structure

Baking Information System primarily aims to achieve functions such as adding, editing, and deleting baked goods, ingredients, and recipes. Overall, the system architecture is clear, with independence between classes. Therefore, we have considered creating multiple classes and simultaneously utilized various data structures such as hash tables and linked lists.

In the implementation of Java code, we have designed several classes, such as ”BakedGood,” ”Ingredient,” and ”Recipe,” to store their corresponding information. For instance, ”BakedGood” includes multiple member variables, such as name (String), origin (String), description (String), and imageUrl (String). ”BakingSystem” serves as the class for the baking goods system, consolidating all information data in the system, as well as methods for searching, editing, and adding. ”BakingSystem” has six member variables, which include hash tables and linked lists storing data for the ”BakedGood,” ”Ingredient,” and ”Recipe” classes. The hash table facilitates quick searches by name, improving operational efficiency, while the list enables traversal and fuzzy search functionality. Through the effective utilization of these four classes and appropriate application of data structures, the baking information system can be implemented efficiently.

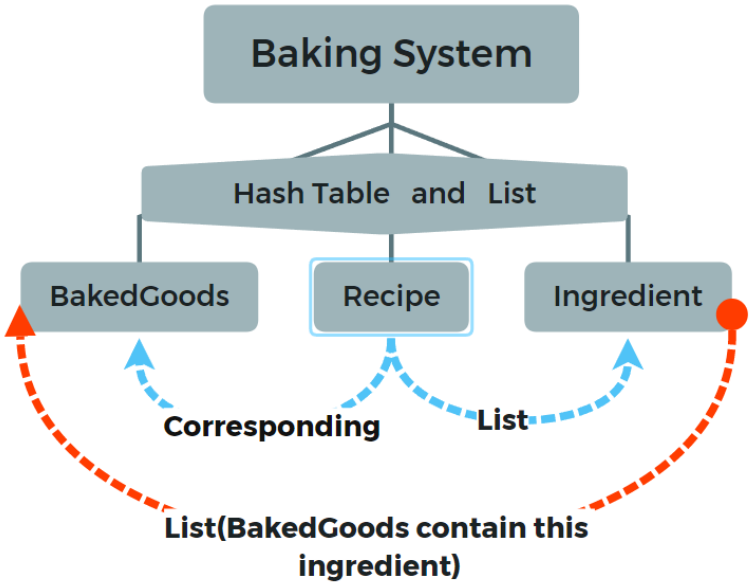


Figure 1: Schematic Diagram of the Overall Structure

As depicted in Figure 1, BakingSystem utilizes lists and hash tables to store data for the three classes. Additionally, there is a one-to-one correspondence between objects of the Recipe class and the BakedGood class. In the Recipe class, a linked list is used to store ingredients and their corresponding qualities. Simultaneously, to identify which baked goods contain a specific ingredient, the Ingredient class includes a linked list to store all baked goods containing that ingredient.

2.2 Implementation of Add/Edit/Delete Functions

2.2.1 The addition functionality

The addition functionality is realized by simultaneously adding elements to both the hash table and the linked list. This involves invoking the put method on the hash table and the addLast method on the linked list. In the hash table, an entry mapping the object's name to the object is added, while in the linked list, the object is appended to the end. Through these steps, data is successfully added.

2.2.2 The edit functionality

The edit functionality initially requires inputting the name of the baked good/ingredient/recipe to be modified. This enables rapid retrieval of the corresponding object from the hash table using the get method. Subsequently, the desired modifications can be input, and the set method is employed to alter the object's parameters. Finally, the put method is used to reintroduce the modified object into the hash table, completing the editing process.

2.2.3 The delete functionality

For the delete functionality, it is first necessary to input the name of the baked good/ingredient/recipe to be deleted. This information is used to retrieve the corresponding objects from both the hash table and the linked list. The delete functionality of the hash table and the linked list is then invoked to execute the deletion operation.

2.3 Implementation of Search Functionality

2.3.1 Implementation of Search for Baked Goods

The system allows searching based on different parameters, such as name, place of origin, etc. When searching by the name of a baked good, it only supports precise matching, meaning the item can be retrieved only when the name is an exact match. Parameters like origin and description support fuzzy matching, allowing the system to list similar items based on the degree of matching, from high to low. Therefore, searching for an origin or description may result in multiple items being listed.

2.3.2 Implementation of Search for Ingredients

The system provides the option to search based on different parameters, such as name, calorie content, etc. When searching by the name of an ingredient, it only supports precise matching. In other words, the item can be retrieved only when the name is an exact match. Parameters like calorie content and description support fuzzy matching, enabling the system to list similar items based on the degree of matching, from high to low. Consequently, searching for calorie content or description may yield multiple items in the result.

2.3.3 Implementation of Search for Recipes

The system allows searching based on various parameters, such as name, total calorie content, etc. When searching by the name of a recipe, it only supports precise matching. The

item can be retrieved only when the name is an exact match. Parameters like total calorie content support fuzzy matching, allowing the system to list similar items based on the degree of matching, from high to low. Consequently, searching for a total calorie value may result in multiple items being listed.

2.4 Implementation of Sorting Based on Different Parameters

In our system, the search results are automatically sorted according to specific criteria. For instance, names can be sorted alphabetically, descriptions can be sorted based on similarity to the search value (in descending order of similarity), and calories can be sorted in descending order. This sorting is achieved through the use of comparators and the quicksort algorithm applied to lists. The following provides a detailed explanation of the quicksort implementation and the comparator.

2.4.1 QuickSort Implementation

• Base Case Check:

If the linked list is empty or contains only one node, it is considered sorted, and the function returns the original list.

• Get Middle Node:

The middle node of the linked list is obtained using the slow and fast pointer technique.

• Split into Two sublists:

The linked list is divided into two sublists—one from the head to the middle and the other from the middle's next node to the end.

• Recursively sort the two sublists:

The quick sort algorithm is recursively applied to both the left and right sublists.

• Merge the Sorted sublists:

The merge operation from the merge sort algorithm is used to combine the two sorted sublists into a single sorted list.

• Get Middle Node:

This method uses the slow and fast pointer technique to find the middle node of a linked list.

• Merge Two Sorted Lists:

This method merges two sorted linked lists into a single sorted list.

Through the use of quick sort, we can rapidly sort all items based on a specific Attribute.

2.4.2 Comparator Implementation

• Comparator for Alphabetical Sorting:

For sorting names alphabetically, a comparator is employed to compare the names of baked goods, ingredients, or recipes. The quicksort algorithm, utilizing this comparator, efficiently rearranges the elements in alphabetical order.

• Comparator for Description Similarity Sorting:

To sort based on the similarity between descriptions and the search value, a comparator assesses the similarity levels and orders the elements accordingly. The quicksort algorithm, guided by this comparator, organizes the items from the highest to the lowest similarity.

• Comparator for Caloric Value Sorting:

For sorting based on caloric values, a comparator is used to compare the calorie content of baked goods, ingredients, or recipes. The quicksort algorithm, driven by this comparator, arranges the elements in descending order of caloric value.

2.5 other methods design

2.5.1 listAllBakedGoodsByName Method

Returns a string representation of all BakedGoods sorted by name. Utilizes the sortBakedGoodsByName method and iterates through the bakedGoodLists linked list.

2.5.2 listAllRecipesByName Method

Returns a string representation of all Recipes sorted by name. Utilizes the sortRecipesByName method and iterates through the recipeList linked list.

2.5.3 listAllRIngredientByName Method

Returns a string representation of all Ingredients sorted by name. Utilizes the sortIngredientsByName method and iterates through the ingredientList linked list.

2.5.4 listAllRecipesByCalories Method

Returns a string representation of all Recipes sorted by total calories. Utilizes the sortRecipesByTotalCalories method and iterates through the recipeList linked list.

2.5.5 listAllRIngredientByCalories Method

Returns a string representation of all Ingredients sorted by calories. Utilizes the sortIngredientsByCalories method and iterates through the ingredientList linked list.

2.5.6 listBakedGoodsContainThisIngredient Method

Returns a string representation of BakedGoods containing a specific Ingredient. Utilizes the getBakedGoodsContainThisIngredient method of the Ingredient class.

2.5.7 computeSimilarity Method

Description: Computes the similarity between two strings basedon a similarity metric. Splits the input strings into arrays of words and calculates the ratio of common words to the total number of words.

2.5.8 CommonWords Method

Determines the number of common words between two arrays of words. Used as a helper method for computing similarity.

2.5.9 abs Method

Returns the absolute value of a given double. Used for absolute value calculations in methods involving numerical comparisons.

2.6 HashTable Class

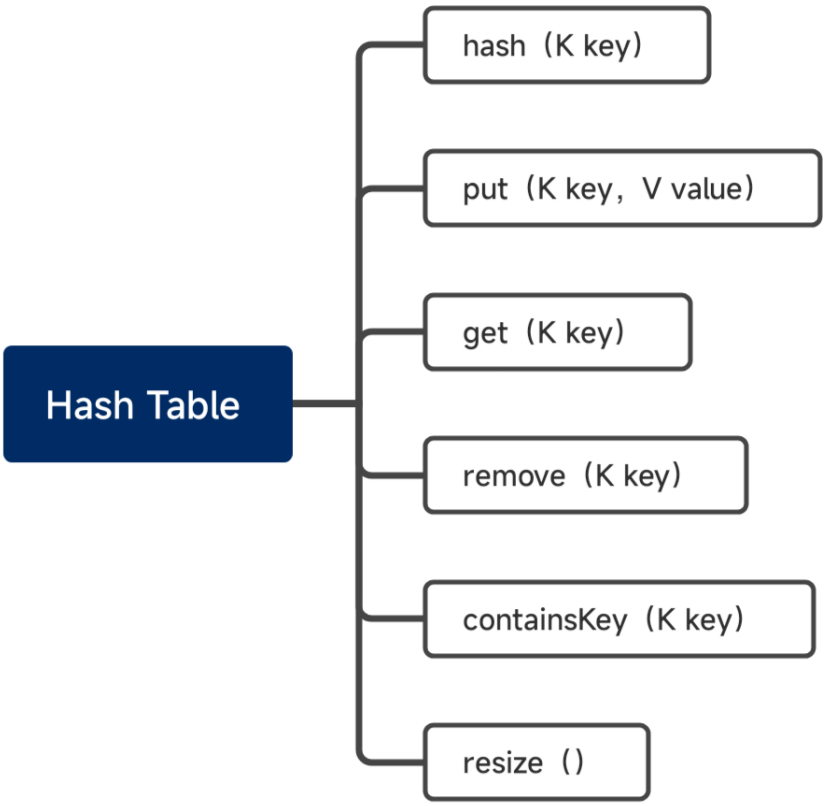


Figure 2: Schematic Diagram of the Hash table

2.6.1 Attributes

- table: An array of Entry objects, representing the buckets in the hash table.
- size: An integer representing the number of key-value pairs stored in the hash table.

2.6.2 Methods

- hash(K key):
Computes the hash value for a given key.
- put(K key, V value):
Inserts a key-value pair into the hash table.
- get(K key):
Retrieves the value associated with a given key.
- remove(K key):

Removes a key-value pair from the hash table.
• containsKey(K key):
Checks if a key is present in the hash table.
• resize():
Resizes the hash table and rehashes all existing key-value pairs.

2.7 Double linked list

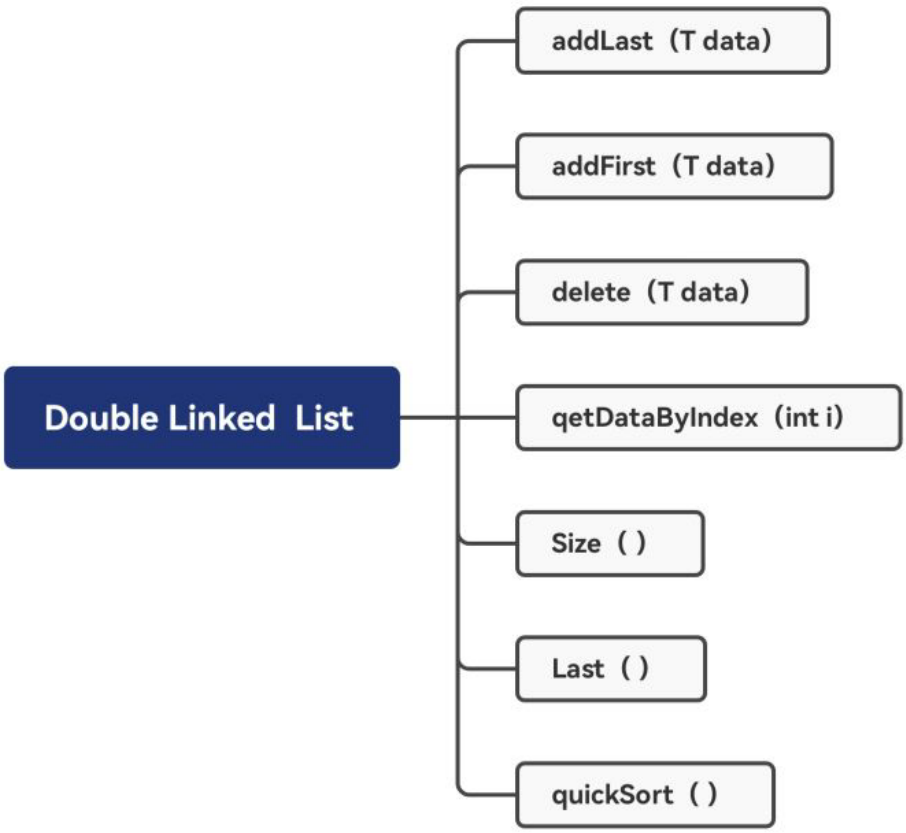


Figure 3: Schematic Diagram of the Double linked list

The figure 3 illustrates the methods associated with the bidirectional linked list we designed and implemented.

• The addLast method is used to add elements to the end of the linked list.
• The addFirst method is used to add elements to the beginning of the linked list.
• The delete method is employed to remove the provided element.
• GetDataByIndex can retrieve the index of the given element.
• Size returns the capacity of the linked list.
• Last returns the last element in the linked list.
• Quicksort performs a quick sort on the elements of the linked list using the Comparable interface.

2.8 save and load

2.8.1 save

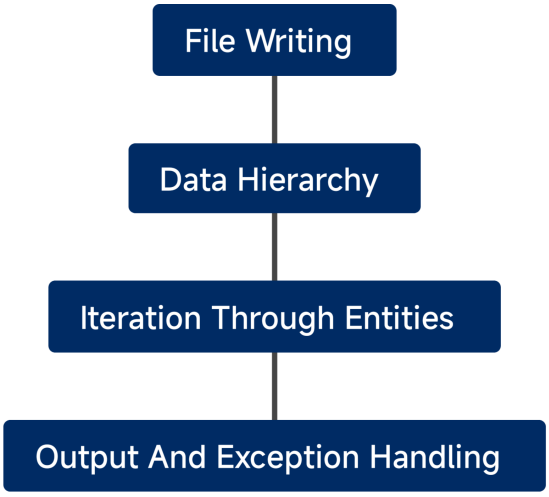


Figure 4: Save flowchart

• File Writing:

The code uses a FileWriter to write data to a file specified by the variable fileName. It writes information about baked goods, ingredients, and recipes to this file. The try-with-resources block ensures that the FileWriter is properly closed after writing the data.

• Data Hierarchy:

The data is organized hierarchically, starting with BakedGood, Ingredient, and Recipe. Information about each entity is written to the file in a specific format, with fields separated by commas.

• Iteration Through Entities:

Nested loops are used to traverse through each level of the data hierarchy. For baked goods, ingredients, and recipes, the relevant information is extracted and written to the file.

• Output and Exception Handling:

Upon successful data saving, a message is printed to the console indicating success. In the event of an IOException during the file writing process, the exception is caught, and the stack trace is printed.

2.8.2 load

• File Reading:

The code uses a BufferedReader to read data from the specified file path. The BufferedReader is encapsulated in a try-with-resources block for proper resource management.

• Entity Initialization:

The method iterates through each line of the file, splitting the line into parts using commas as separators. A switch statement is employed to identify the type of entity (BAKEDGOOD, INGREDIENT, RECIPE). Based on the entity type, corresponding objects are created and initialized.

For ”BAKEDGOOD” and ”INGREDIENT” lines, objects of the respective types (BakedGood and Ingredient) are created and initialized using the information from the file.

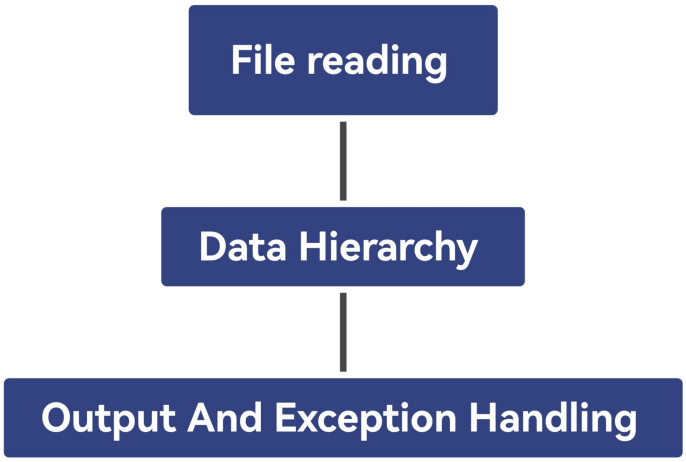


Figure 5: Load flowchart

For ”RECIPE” lines, a new Recipe is created, and a loop is used to read the ingredient information until an empty line is encountered. For each ingredient, a corresponding Ingredient object is retrieved, and the Recipe is populated with the ingredients and quantities.

• Output and Exception Handling:

Upon successful data loading, a message is printed to the console. If an IOException occurs during the file reading process, the exception is caught, and the stack trace is printed.

Upon utilization of the ”Save” functionality, the pertinent data will be automatically stored in a predefined format within a text file located in the current directory, as illustrated in the figure below. The text file is amenable to direct modification and other operations.

```
BAKEDGOOD,Fisherman's toast,Mondstatd,Toast covered with onions. It is a popular staple food among
fishermen. They will stuff their bags with this toast. And squat by the river for the whole day.
Formulation materials ,C:\Users\1.jpg
BAKEDGOOD,Mond potato cake,Mondstatd,Potato fritters. A few pinecones provide a varied taste. Can be
eaten with jam, popular with all ages,C:\Users\1.jpg
INGREDIENT,meat,null,100.0
INGREDIENT,potato,111,40.0
INGREDIENT,raspberry,null,20.0
INGREDIENT,onion,null,20.0
INGREDIENT,milk,null,30.0
INGREDIENT,tomato,null,10.0
INGREDIENT,shrimp,null,50.0
INGREDIENT,flour,null,10.0
INGREDIENT,egg,null,40.0
```

Figure 6: data file Schematic Diagram

2.9 Graphical User Interface

The GUI interface, is a type of user interface that allows users to interact with computer programs through graphical elements such as buttons, menus, icons, and more. The design of GUI interfaces should adhere to some fundamental principles, including usability, consistency, feedback, aesthetics, and others. In this CA assignment, we utilized Swing to implement the functionalities of the GUI.

The primary architecture of the GUI system is depicted in the diagram below.

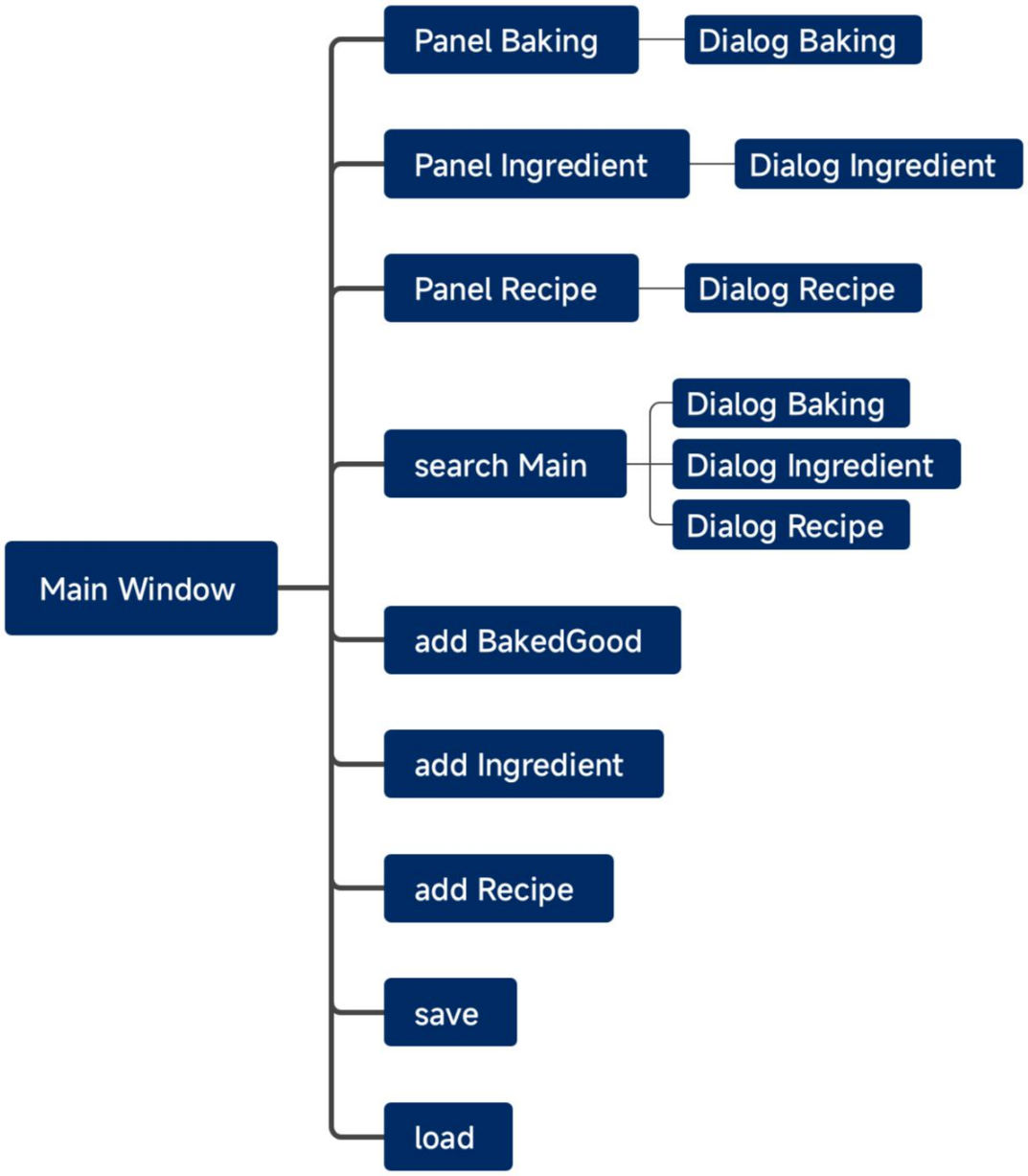


Figure 7: GUI structure

2.9.1 Main Window

Create a JFrame mainWindow, create a new JPanel panelButton, and add panels how to mainWindow, create a new JPanel panelShow, add nine buttons to panelButton, and add panelShow to mainWindow, set the panelShow layout is

cardLayout.

2.9.2 Panel Baking

Using a for loop to create buttons, connect with mainWindow by panelButton.

2.9.3 Panel Ingredient

Using a for loop to create buttons, connect with mainWindow by panelButton.

2.9.4 Panel Recipe

Using a for loop to create buttons, connect with mainWindow by panelButton.

2.9.5 Search Main

Have two functions, one is set for the different type of search, another is set for different kind of search.

2.9.6 Add BakedGood

Using a Brid layout to add three label and three text fired. And using a window chooser to choose picture from computer. Add the picture address to the BakedGood.

2.9.7 Add Ingredient

Using a Bird Layout to add three labels and three text fired.

2.9.8 Add Recipe

Using two ComboBox to add name and ingredient.

2.9.9 Dialog Baking

Created a new dialog to show the information about the select BakedGood. Using Frow Layout to add three panel, panel of picture, panel of information, panel of buttons.

2.9.10 Dialog Ingredient

Created a new dialog to show the information about the select ingredient. Using Frow Layout to add two panels, panel of information, panel of buttons.

2.9.11 Dialog Recipe

Created a new dialog to show the information about the select ingredient. Using Frow Layout to add two panels, panel of information, panel of buttons.

2.10 JUnit test

The "Test" class includes a test source file, which contains various types of JUnit test methods for various methods, and each method can be implemented and run

independently, and also uses assert to test whether the method succeeds.You can also observe the success of getting the passed arguments by modifying the passed arguments.

JUnit not only reduces the main method redundancy of normal test files, but also reduces the memory required to run the files. JUnit can also perform debug tests to test your code errors, which can easily modify the code, quickly adjust the structure of the code, and so on, reducing the time spent by the programmer on debugging.JUnit can automatically run and produce its own results and provide timely feedback, eliminating the process of manually combing test results.

3 Implementation (including codes)

3.1 Baking information system

3.1.1 Baked goods

3.1.1.1 Attribute

```
public class BakedGood {
    private String name;
    private String origin;
    private String description;
    private String imageUrl;

    @Override
    public String toString() {
        return "BakedGood{" +
                "name='" + name + '\'' +
                ", origin='" + origin + '\'' +
                ", description='" + description + '\'' +
                ", imageUrl='" + imageUrl + '\'' +
                '}';
    }
```

3.1.1.2 Get and set

```
public BakedGood(String name, String origin, String description, String imageUrl)
{
    this.name = name;
    this.origin = origin;
    this.description = description;
    this.imageUrl = imageUrl;
}

public String getName() {
    return name;
}
```

```
public void setName(String name) {
    this.name = name;
}

public String getOrigin() {
    return origin;
}

public void setOrigin(String origin) {
    this.origin = origin;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public String getImageUrl() {
    return imageUrl;
}

public void setImageUrl(String imageUrl) {
    this.imageUrl = imageUrl;
}
```

3.1.1.3 Sort the baked food by name

```
public static void sortBakedGoodsByName(List<BakedGood> bakedGoods) {
        bakedGoods.quickSort(new Comparator<BakedGood>() {
            @Override
            public int compare(BakedGood o1, BakedGood o2) {
                return o1.getName().compareTo(o2.getName());
            }
        });
    }

    public static void sortBakedGoodsByOrigin(List<BakedGood> bakedGoods) {
        bakedGoods.quickSort(new Comparator<BakedGood>() {
            @Override
            public int compare(BakedGood o1, BakedGood o2) {
                return o1.getOrigin().compareTo(o2.getOrigin());
            }
        });
```

```
    }

    public static void sortBakedGoodsByDescription(List<BakedGood> bakedGoods)
{
        bakedGoods.quickSort(new Comparator<BakedGood>() {
            @Override
            public int compare(BakedGood o1, BakedGood o2) {
                return o1.getDescription().compareTo(o2.getDescription());
            }
        });
    }
}
```

3.1.2 Baking system

```
public class BakingSystem {
    public static HashTable<String, BakedGood> bakedGoodsTable = new
HashTable<>();
    public static HashTable<String, Ingredient> ingredientsTable = new
HashTable<>();
    public static HashTable<String, Recipe> recipesTable = new HashTable<>();

    public static List<BakedGood> bakedGoodLists = new List<>();
    public static List<Ingredient> ingredientList = new List<>();
    public static List<Recipe> recipeList = new List<>();

    public static void addBakedGood(String name, String origin, String
description, String imageUrl) {
        BakedGood bakedGood = new BakedGood(name, origin, description, imageUrl);
        bakedGoodsTable.put(bakedGood.getName(), bakedGood);
        bakedGoodLists.addLast(bakedGood);
    }

    public static void addIngredient(String name, String description, double
calories) {
        Ingredient ingredient = new Ingredient(name, description, calories);
        ingredientsTable.put(ingredient.getName(), ingredient);
        ingredientList.addLast(ingredient);
    }

    public static void addRecipe(BakedGood bakedGood, List<Ingredient>
ingredients, List<Double> qualities) {
        Recipe recipe = new Recipe(bakedGood, ingredients, qualities);
        recipesTable.put(bakedGood.getName(), recipe);
        recipeList.addLast(recipe);
    }
```

3.1.2.1 Edit a bakedGood which has been put into the hashTable

```
public static void editBakedGood(String preName, String newName, String origin,
String description, String imageUrl) {
    bakedGoodsTable.get(preName).setName(newName);
    bakedGoodsTable.get(preName).setOrigin(origin);
    bakedGoodsTable.get(preName).setDescription(description);
    bakedGoodsTable.get(preName).setImageUrl(imageUrl);

    bakedGoodsTable.put(newName, bakedGoodsTable.get(preName));
    bakedGoodsTable.remove(preName);
}
```

3.1.2.2 Edit an ingredient which has been put into the hash table

```
public static void editIngredient(String preName, String newName, String
description, double calories) {
    ingredientsTable.get(preName).setName(newName);
    ingredientsTable.get(preName).setDescription(description);
    ingredientsTable.get(preName).setCalories(calories);

    ingredientsTable.put(newName, ingredientsTable.get(preName));
    ingredientsTable.remove(preName);
}
```

3.1.2.3 Edit a recipe which has been put into the hash table

```
public static void editRecipe(String name, List<Ingredient> ingredients,
List<Double> qualities) {
    recipesTable.get(name).setIngredients(ingredients);
    recipesTable.get(name).setQualities(qualities);
}

public static BakedGood getBakedGoodByName(String name) {
    return bakedGoodsTable.get(name);
}

public static Ingredient getIngredientByName(String name) {
    return ingredientsTable.get(name);
}

public static Recipe getRecipeByName(String name) {
    return recipesTable.get(name);
}
```

3.1.2.4 Delete methods

```
public static void deleteBakedGood(String name) {
    bakedGoodLists.delete(BakingSystem.getBakedGoodByName(name));
    bakedGoodsTable.remove(name);
```

```
}

public static void deleteIngredient(String name) {
    ingredientList.delete(BakingSystem.getIngredientByName(name));
    ingredientsTable.remove(name);
}

public static void deleteRecipe(String name) {
    recipeList.delete(BakingSystem.getRecipeByName(name));
    recipesTable.remove(name);
}
```

3.1.2.5 List all baked goods by name

```
public static String listAllBakedGoodsByName() {
    BakedGood.sortBakedGoodsByName(bakedGoodLists);
    StringBuilder content = new StringBuilder();
    for (int i = 0; i < bakedGoodLists.size(); i++) {
        content.append(bakedGoodLists.getDataByIndex(i).toString());
        content.append("\n");
    }
    return content.toString();
}
```

3.1.2.6 List all recipes by name

```
public static String listAllRecipesByName() {
    Recipe.sortRecipesByName(recipeList);
    StringBuilder content = new StringBuilder();
    for (int i = 0; i < recipeList.size(); i++) {
        content.append(recipeList.getDataByIndex(i).toString());
        content.append("\n");
    }
    return content.toString();
}
```

3.1.2.7 List all recipes by calories

```
public static String listAllRecipesByCalories() {
    Recipe.sortRecipesByTotalCalories(recipeList);
    StringBuilder content = new StringBuilder();
    for (int i = 0; i < recipeList.size(); i++) {
        content.append(recipeList.getDataByIndex(i).toString());
        content.append("\n");
    }
    return content.toString();
}
```

3.1.2.8 List all ingredient by name

```
public static String listAllIngredientByName() {
    Ingredient.sortIngredientsByName(ingredientList);
```

```
    StringBuilder content = new StringBuilder();
    for (int i = 0; i < ingredientList.size(); i++) {
        content.append(ingredientList.getDataByIndex(i).toString());
        content.append("\n");
    }
    return content.toString();
}
```

3.1.2.9 List all ingredient by calories

```
public static String listAllIngredientByCalories() {
    Ingredient.sortIngredientsByCalories(ingredientList);
    StringBuilder content = new StringBuilder();
    for (int i = 0; i < ingredientList.size(); i++) {
        content.append(ingredientList.getDataByIndex(i).toString());
        content.append("\n");
    }
    return content.toString();
}
```

3.1.2.10 List baked goods contain this Ingredient

```
public static String listBakedGoodsContainThisIngredient(Ingredient ingredient)
{

BakedGood.sortBakedGoodsByName(ingredient.getBakedGoodsContainThisIngredient(
));
    return ingredient.getBakedGoodsContainThisIngredient().toString();
}
```

3.1.2.11 Search baked goods

```
public static List<BakedGood> searchBakedGood(String content, String parameter)
{
    switch (parameter) {
        case "name" -> {
            List<BakedGood> list = new List<>();
            list.addLast(bakedGoodsTable.get(content));
            return list;
        }
        case "description" -> {
            List<BakedGood> list = new List<>();
            for (int i = 0; i < bakedGoodLists.size(); i++) {
                if
(computeSimilarity(bakedGoodLists.getDataByIndex(i).getDescription(),
content) >= 0.6) {
                    list.addLast(bakedGoodLists.getDataByIndex(i));
                }
            }
            list.quickSort((o1,                    o2)                    ->
```

```
(Double.compare(((computeSimilarity(o2.getDescription(),content))),computeSim
ilarity(o1.getDescription(),content))));
            return list;
        }
        case "origin" -> {
            List<BakedGood> list = new List<>();
            for (int i = 0; i < bakedGoodLists.size(); i++) {
                if
(Objects.equals(bakedGoodLists.getDataByIndex(i).getOrigin(), content)) {
                    list.addLast(bakedGoodLists.getDataByIndex(i));
                }
            }
            return list;
        }
    }
    return new List<>();
}
```

3.1.2.12 Search Ingredient

```
public static List<Ingredient> searchIngredient(String content, String
parameter) {
    switch (parameter) {
        case "name" -> {
            List<Ingredient> list = new List<>();
            list.addLast(ingredientsTable.get(content));
            return list;
        }
        case "description" -> {
            List<Ingredient> list = new List<>();
            for (int i = 0; i < ingredientList.size(); i++) {
                if
(computeSimilarity(ingredientList.getDataByIndex(i).getDescription(),
content) >= 0.6) {
                    list.addLast(ingredientList.getDataByIndex(i));
                }
            }
            list.quickSort((o1, o2) ->
(Double.compare(((computeSimilarity(o2.getDescription(),content))),computeSim
ilarity(o1.getDescription(),content))));
            return list;
        }
        case "calories" -> {
            List<Ingredient> list = new List<>();
            for (int i = 0; i < ingredientList.size(); i++) {
                if
```

```
((abs((ingredientList.getDataByIndex(i).getCalories()-Double.parseDouble(cont
ent))/Double.parseDouble(content)))<0.1){
                    list.addLast(ingredientList.getDataByIndex(i));
                }
            }
          list.quickSort(new Comparator<Ingredient>() {
              @Override
              public int compare(Ingredient o1, Ingredient o2) {
                  return
Double.compare((abs((o1.getCalories()-Double.parseDouble(content))/Double.par
seDouble(content))),(abs((o2.getCalories()-Double.parseDouble(content))/Doubl
e.parseDouble(content))));
              }
          });
            return list;
        }
    }
    return new List<>();
}
```

3.1.2.13 Search recipe

```
public static List<Recipe> searchRecipe(String content, String parameter) {
    switch (parameter) {
        case "name" -> {
            List<Recipe> list = new List<>();
            list.addLast(recipesTable.get(content));
            return list;
        }

        case "calories" -> {
            List<Recipe> list = new List<>();
            for (int i = 0; i < recipeList.size(); i++) {
                if
((abs((recipeList.getDataByIndex(i).getTotalCalories()-Double.parseDouble(con
tent))/Double.parseDouble(content)))<0.3){
                    list.addLast(recipeList.getDataByIndex(i));
                }
            }
            list.quickSort(new Comparator<Recipe>() {
                @Override
                public int compare(Recipe o1, Recipe o2) {
                    return
Double.compare((abs((o1.getTotalCalories()-Double.parseDouble(content))/Doubl
e.parseDouble(content))),(abs((o2.getTotalCalories()-Double.parseDouble(conte
nt))/Double.parseDouble(content))));
```

```
                }
            });
            return list;
        }
    }
    return new List<>();
}
```

3.1.2.14 Absolute value

```
public static double abs(double number) {
    if (number > 0) {
        return number;
    } else return -number;
}
```

3.1.2.15 Compute similarity

```
public static double computeSimilarity(String content1, String content2) {
    String[] words1 = content1.split("\\s");
    String[] words2 = content2.split("\\s");

    int commonWords = CommonWords(words1, words2);

    int totalWords = words1.length + words2.length;

   return (double) commonWords / totalWords;
}
```

3.1.2.16 Find common words

```
private  static int CommonWords(String[] words1, String[] words2) {
        int i = 0;
        for (String word1 : words1) {
            for (String word2 : words2) {
                if (word1.equals(word2)) {
                    i++;
                    break;
                }
            }
        }
        return i;
    }
}
```

3.1.3 Ingredient

```
public class Ingredient {
    private String name;
    private String description;
    private double calories;
```

```
    @Override
    public String toString() {
        return "Ingredient{" +
                "name='" + name + '\'' +
                ", description='" + description + '\'' +
                ", calories=" + calories +
                ",bakedGoodsContainThisIngredient=\n\n"                                    +
bakedGoodsContainThisIngredient +
                '}';
    }

    private  List<BakedGood> bakedGoodsContainThisIngredient=new List<>();

    public List<BakedGood> getBakedGoodsContainThisIngredient() {
        return bakedGoodsContainThisIngredient;
    }

    public Ingredient(String name, String description, double calories) {
        this.name = name;
        this.description = description;
        this.calories = calories;
    }
```

3.1.3.1 Get information about ingredient

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public double getCalories() {
    return calories;
}

public void setCalories(double calories) {
```

```
    this.calories = calories;
}
```

3.1.3.2 Conduct sorting ingredient by name

```
public static void sortIngredientsByName(List<Ingredient> ingredients) {
    ingredients.quickSort(new Comparator<Ingredient>() {
        @Override
        public int compare(Ingredient o1, Ingredient o2) {
            return o1.getName().compareTo(o2.getName());
        }
    });
}

public static void sortIngredientsByDescription(List<Ingredient> ingredients) {
    ingredients.quickSort(new Comparator<Ingredient>() {
        @Override
        public int compare(Ingredient o1, Ingredient o2) {
            return o1.getDescription().compareTo(o2.getDescription());
        }
    });
}
```

3.1.3.3 Conduct sorting ingredient by calories

```
public static void sortIngredientsByCalories(List<Ingredient> ingredients) {
        ingredients.quickSort(new Comparator<Ingredient>() {
            @Override
            public int compare(Ingredient o1, Ingredient o2) {
                return Double.compare(o1.getCalories(),o2.getCalories());
            }
        });
    }
}
```

3.1.4 Recipe

3.1.4.1 Get information about recipes

```
public class Recipe {
    private BakedGood bakedGood;
    private List<Ingredient> ingredients;
    private List<Double> qualities;

    @Override
    public String toString() {
        StringBuilder content= new StringBuilder("Recipe{" +
                "bakedGood="          +          bakedGood.getName()          +"
calories="+this.getTotalCalories()+
                "}\n");
```

```
        for (int i=0;i<ingredients.size();i++){
            content.append(ingredients.getDataByIndex(i).getName()).append("
").append(qualities.getDataByIndex(i));
            content.append("\n");
        }
        return content.toString();
    }

    public double getTotalCalories() {
        int totalCalories = 0;
        for (int i = 0; i < ingredients.size(); i++) {
            totalCalories   +=   ingredients.getDataByIndex(i).getCalories()   *
qualities.getDataByIndex(i);
        }
        return totalCalories;
    }

    public List<Double> getQualities() {
        return qualities;
    }

    public void setQualities(List<Double> qualities) {
        this.qualities = qualities;
    }

    public BakedGood getBakedGood() {
        return bakedGood;
    }

    public void setBakedGood(BakedGood bakedGood) {
        this.bakedGood = bakedGood;
    }

    public List<Ingredient> getIngredients() {
        return ingredients;
    }
```

3.1.4.2 Set relevant ingredients

```
public void setIngredients(List<Ingredient> ingredients) {
    this.ingredients = ingredients;
}
```

3.1.4.3 Get and sort recipes

```
public Recipe(BakedGood bakedGood, List<Ingredient> ingredients, List<Double>
qualities) {
        this.bakedGood = bakedGood;
```

```
        this.ingredients = ingredients;
        this.qualities = qualities;

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

ingredients.getDataByIndex(i).getBakedGoodsContainThisIngredient().addLast(ba
kedGood);
        }
    }

    public static void sortRecipesByName(List<Recipe> recipes) {
        recipes.quickSort(new Comparator<Recipe>() {
            @Override
            public int compare(Recipe o1, Recipe o2) {
                return
o1.getBakedGood().getName().compareTo(o2.getBakedGood().getName());
            }
        });
    }


    public static void sortRecipesByTotalCalories(List<Recipe> recipes) {
        recipes.quickSort(new Comparator<Recipe>() {
            @Override
            public int compare(Recipe o1, Recipe o2) {
                return                    Double.compare(o1.getTotalCalories(),
o2.getTotalCalories());
            }
        });
    }
}
```

3.2 Date structure

3.2.1 Hash table

```
public class load {
        public static void loadSystemData() {
            String projectDir = System.getProperty("user.dir");
            String fileName = projectDir + "/out"+"/data.dat";
            try    (BufferedReader    reader    =    new    BufferedReader(new
FileReader(fileName))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    String[] tokens = line.split(",");
                    String type = tokens[0];
```

```
                switch (type) {
                    case "BAKEDGOOD" -> {
                        String name = tokens[1];
                        String origin = tokens[2];
                        String description = tokens[3];
                        String imageUrl = tokens[4];
                        BakingSystem.addBakedGood(name, origin, description,
imageUrl);
                    }
                    case "INGREDIENT" -> {
                        String ingredientName = tokens[1];
                        String ingredientDescription = tokens[2];
                        double calories = Double.parseDouble(tokens[3]);
                        BakingSystem.addIngredient(ingredientName,
ingredientDescription, calories);
                    }
                    case "RECIPE" -> {
                        String recipeName = tokens[1];
                        BakedGood bakedGood =
BakingSystem.getBakedGoodByName(recipeName);
                        if (bakedGood != null) {
                            List<Ingredient> ingredients = new List<>();
                            List<Double> quantities = new List<>();

                            while ((line = reader.readLine()) != null
&& !line.isEmpty()) {
                                String[] ingredientTokens = line.split(",");
                                String ingredientNameInRecipe =
ingredientTokens[0];
                                double quantity =
Double.parseDouble(ingredientTokens[1]);

                                Ingredient ingredientInRecipe =
BakingSystem.getIngredientByName(ingredientNameInRecipe);
                                if (ingredientInRecipe != null) {
                                    ingredients.addLast(ingredientInRecipe);
                                    quantities.addLast(quantity);
                                }
                            }

                            BakingSystem.addRecipe(bakedGood, ingredients,
quantities);
                        }
```

```
                    }
                    default ->
                            System.err.println("Unknown type: " + type);
                }
            }

            System.out.println("System data loaded successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3.2.2 List

3.2.2.1 Define the parent class double linked list

```
public class List<T> {

    private static class Node<U> {
        public U data;

        Node<U> prev;
        Node<U> next;

        public Node(U data) {
            this.data = data;
            this.prev = null;
            this.next = null;
        }

        public U getData() {
            return data;
        }

        public void setData(U data) {
            this.data = data;
        }
    }

    public Node<T> head;

    public List() {
        this.head = null;
    }
```

3.2.2.2 Add a node to the end of the list

```
public void addLast(T data) {
```

```
    Node<T> i = new Node<>(data);
    if (head == null) {
        head = i;
    } else {
        Node<T> p = head;
        while (p.next != null) {
            p = p.next;
        }
        p.next = i;
        i.prev = p;
    }
}
```

3.2.2.3 Insert a node at the head of the list. After the node is found and deleted, the function ends

```
public void addFirst(T data) {
    Node<T> i = new Node<>(data);
    i.next = head;
    if (head != null) {
        head.prev = i;
    }
    head = i;
}
public void delete(T data) {
    Node<T> p = head;

    while (p != null) {
        if (p.data.equals(data)) {
            if (p.prev != null) {
                p.prev.next = p.next;
            } else {
                head = p.next;
            }

            if (p.next != null) {
                p.next.prev = p.prev;
            }

            return;         }
        p = p.next;
    }
}
```

3.2.2.4 Find the node for the specified data.If not found, null is returned

```
public Node<T> find(T data) {
    Node<T> p = head;
```

```
    while (p != null) {
        if (p.data.equals(data)) {
            return p;
        }

        p = p.next;
    }

    return null;
```

3.2.2.5 Print the list

```
public void print() {
    Node<T> p = head;

    while (p != null) {
        System.out.print(p.data + " ");
        p = p.next;
    }

    System.out.println();
```

3.2.2.6 Get the node value based on the node location

```
public T getDataByIndex(int index) {
    if (index < 0) {
        throw new IndexOutOfBoundsException("Index should be positive: " + index);
    }

    Node<T> p = head;
    int i = 0;

    while (p != null) {
        if (i == index) {
            return p.data;
        }
        p = p.next;
        i++;
    }

    return null;
}
```

3.2.2.7 Get the number of linked list nodes

```
public int size() {
    Node<T> p = head;
    int i = 0;
```

```
    while (p != null) {
        i++;
        p = p.next;
    }
    return i;
}

public T last() {
    if (head == null) {
        return null;
    }

    Node<T> p = head;
    while (p.next != null) {
        p = p.next;
    }

    return p.data;
}
```

3.2.2.8 Clear list

```
public void clear() {
    head = null;
}

public int getIndexByData(T data) {
    Node<T> p = head;
    int i = 0;

    while (p != null) {
        if (p.data == data) {
            return i;
        }
        p = p.next;
        i++;
    }

    return -1;
}
```

3.2.2.9 Quick sort of linked list

```
public void quickSort(Comparator<T> comparator) {
    head = quickSort(head, comparator);
}
private Node<T> quickSort(Node<T> head, Comparator<T> comparator) {
```

```
    if (head == null || head.next == null) {
        return head;
    }

    Node<T> middle = getMiddle(head);
    Node<T> left = head;
    Node<T> right = middle.next;
    middle.next = null;
    left = quickSort(left, comparator);
    right = quickSort(right, comparator);
    return merge(left, right, comparator);
}

private Node<T> getMiddle(Node<T> head) {
    if (head == null || head.next == null) {
        return head;
    }

    Node<T> slow = head;
    Node<T> fast = head.next;

    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    return slow;
}
```

```
private Node<T> merge(Node<T> left, Node<T> right, Comparator<T> comparator) {
        Node<T> dummy = new Node<>(null);
        Node<T> current = dummy;

        while (left != null && right != null) {
            if (comparator.compare(left.data, right.data) < 0) {
                current.next = left;
                left.prev = current;
                left = left.next;
            } else {
                current.next = right;
                right.prev = current;
                right = right.next;
            }
```

```
            current = current.next;
        }

        if (left != null) {
            current.next = left;
            left.prev = current;
        }

        if (right != null) {
            current.next = right;
            right.prev = current;
        }

        return dummy.next;
    }

    public String toString() {
        StringBuilder content = new StringBuilder();
        for (int i = 0; i < this.size(); i++) {
            content.append(this.getDataByIndex(i).toString());
            content.append("\n");
        }
        return content.toString();
    }
}
```

3.3 Save and load

3.3.1 Save

```
public class save {
    public static void saveSystemData() {
        String projectDir = System.getProperty("user.dir");
        String fileName = projectDir + "/out"+"/data.dat";
        try (FileWriter writer = new FileWriter(fileName)) {

            List<BakedGood> bakedGoods = BakingSystem.bakedGoodLists;
            for (int i = 0; i < bakedGoods.size(); i++) {
                BakedGood bakedGood = bakedGoods.getDataByIndex(i);
                writer.write("BAKEDGOOD," + bakedGood.getName() + "," +
bakedGood.getOrigin() + "," +
                        bakedGood.getDescription() + "," +
bakedGood.getImageUrl() + "\n");
            }
```

```
            List<Ingredient> ingredients = BakingSystem.ingredientList;
            for (int i = 0; i < ingredients.size(); i++) {
                Ingredient ingredient = ingredients.getDataByIndex(i);
                writer.write("INGREDIENT," + ingredient.getName() + "," +
ingredient.getDescription() + "," +
                        ingredient.getCalories() + "\n");
            }

            List<Recipe> recipes = BakingSystem.recipeList;
            for (int i = 0; i < recipes.size(); i++) {
                Recipe recipe = recipes.getDataByIndex(i);
                writer.write("RECIPE," + recipe.getBakedGood().getName() + "," +
                        recipe.getTotalCalories() + "\n");


                List<Ingredient> ingredientsInRecipe = recipe.getIngredients();
                List<Double> quantities = recipe.getQualities();
                for (int j = 0; j < ingredientsInRecipe.size(); j++) {
                    Ingredient ingredientInRecipe =
ingredientsInRecipe.getDataByIndex(j);
                    double quantity = quantities.getDataByIndex(j);

                    writer.write(ingredientInRecipe.getName() + "," + quantity +
"\n");
                }

                writer.write("\n");
            }

            System.out.println("System data saved successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3.3.2 Load

```
public class load {
        public static void loadSystemData() {
            String projectDir = System.getProperty("user.dir");
            String fileName = projectDir + "/out"+"/data.dat";
            try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
                String line;
                while ((line = reader.readLine()) != null) {
```

```
                String[] tokens = line.split(",");
                String type = tokens[0];

                switch (type) {
                    case "BAKEDGOOD" -> {
                        String name = tokens[1];
                        String origin = tokens[2];
                        String description = tokens[3];
                        String imageUrl = tokens[4];
                        BakingSystem.addBakedGood(name, origin, description,
imageUrl);
                    }
                    case "INGREDIENT" -> {
                        String ingredientName = tokens[1];
                        String ingredientDescription = tokens[2];
                        double calories = Double.parseDouble(tokens[3]);
                        BakingSystem.addIngredient(ingredientName,
ingredientDescription, calories);
                    }
                    case "RECIPE" -> {
                        String recipeName = tokens[1];
                        BakedGood bakedGood =
BakingSystem.getBakedGoodByName(recipeName);
                        if (bakedGood != null) {
                            List<Ingredient> ingredients = new List<>();
                            List<Double> quantities = new List<>();

                            while ((line = reader.readLine()) != null
&& !line.isEmpty()) {
                                String[] ingredientTokens = line.split(",");
                                String ingredientNameInRecipe =
ingredientTokens[0];
                                double quantity =
Double.parseDouble(ingredientTokens[1]);

                                Ingredient ingredientInRecipe =
BakingSystem.getIngredientByName(ingredientNameInRecipe);
                                if (ingredientInRecipe != null) {
                                    ingredients.addLast(ingredientInRecipe);
                                    quantities.addLast(quantity);
                                }
                            }

                            BakingSystem.addRecipe(bakedGood, ingredients,
```

```
quantities);
                        }
                    }
                    default ->
                            System.err.println("Unknown type: " + type);
                }
            }

            System.out.println("System data loaded successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4 Results and Conclusions (Results analysis and discussion)

4.1Results

This program has successfully completed various functions required by the homework, and will be demonstrated below：

In MainWindow, you can perform all operations, such as viewing the BakedGoodPanel, IngredientPane and other interfaces, adding BakedGood, Ingredient and Recipe, as well as save, load and search functions.

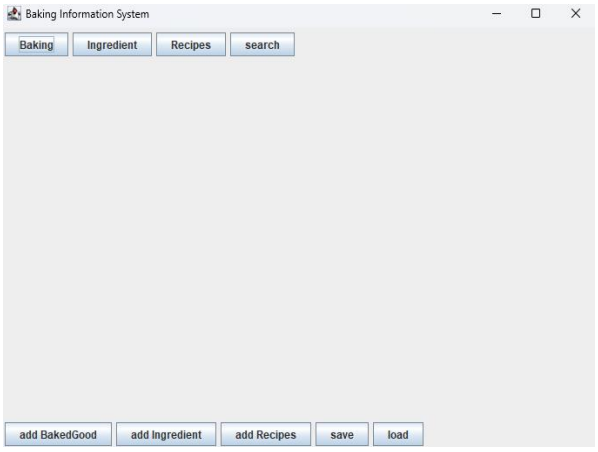


Figure 8: MainWindow show

4.1.1 BakingPanel

This button is on the above of the MainWindow and it is named as “Baking”.

Figure 9: Baking Button show

Click this button, BakedGoodPanel will be showed in MainWindow.

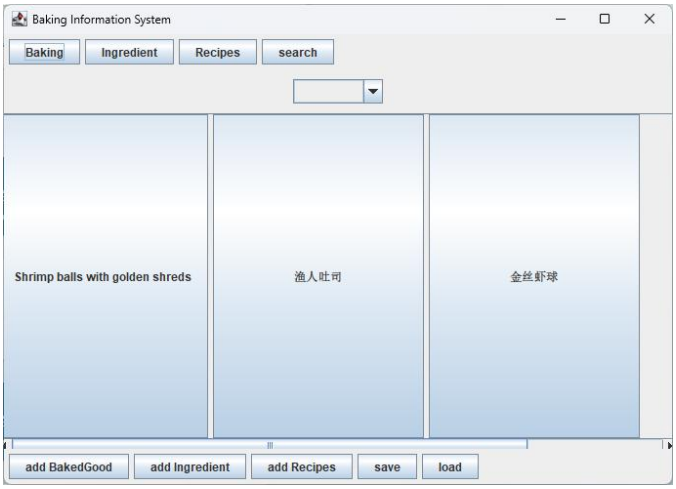


Figure 10: BakedGoodPanel show

The BakedGood added before can be seen.

4.1.2 IngredientPanel

This button is on the above of the MainWindow and it is named as “Ingredient”.

Figure 11: ingredient Button show

Click this button, IngredientPanel will be showed in MainWindow.

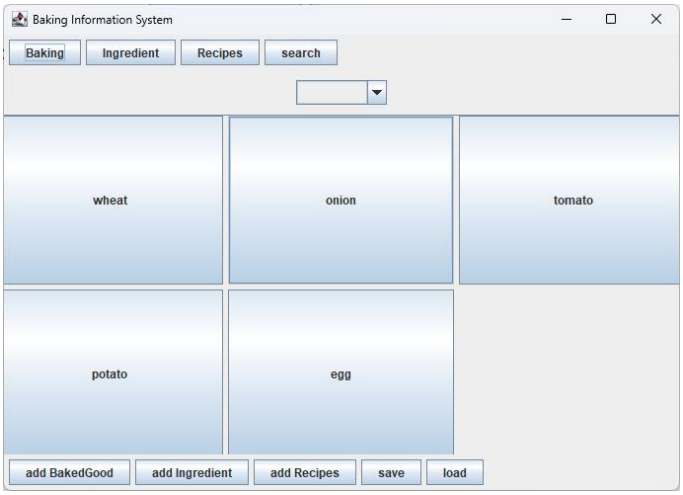


Figure 12: IngredientPanel show

The ingredient added before can be seen.

4.1.3 RecipePanel

This button is on the above of the MainWindow and it is named as “Recipe”.

Figure13: recipe Button show

Click this button, IngredientPanel will be showed in MainWindow.

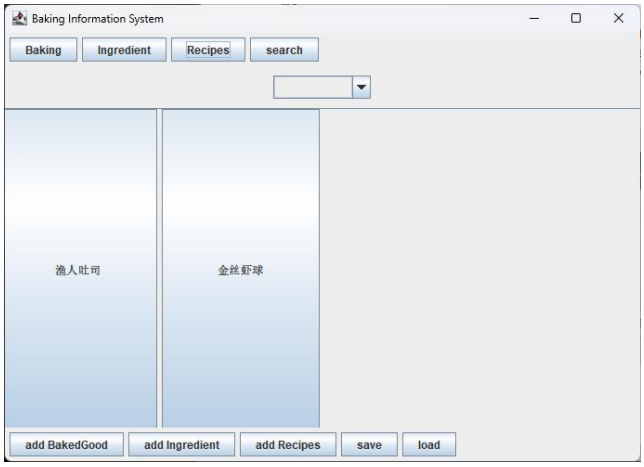


Figure 14: RecipePenal show

The recipe added before can be seen.

4.1.4 Search

This button is on the top of the MainWindow and it is named as “search”.

Figure 15: search button show

Click this button, searchPanel will be showed in MainWindow.

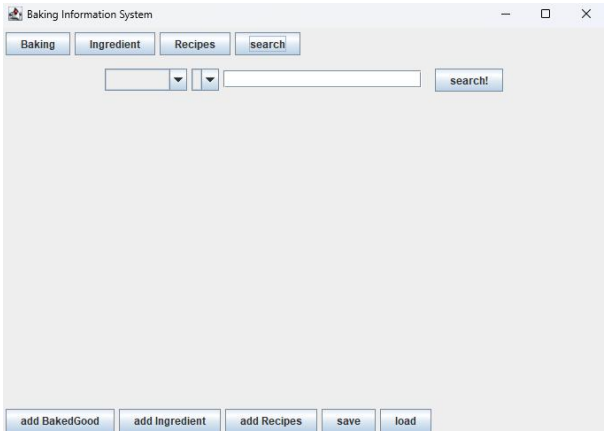


Figure 16: searchPanel show

In this panel, two ComboBox can be seen, the first one is used to choose search type from

A. “BakedGood”

B. “Ingredient”

C. “Recipe”

Figure 17: first ComboBox show

The second ComboBox only can be operated after you choose Item from the first ComboBox, otherwise the second ComboBox is empty.

If the first ComboBox select Item is BakedGood, the second ComboBox will appeal three Item to choose:

A.name

B.origin

C.description

If you choose “name”, enter the name of the BakedGood you want the TextField, then click the “Search” button, a dialog of the BakedGood you want will appear.

Figure 18: search BakedGood by name show

If you choose “origin”, enter the origin of the BakedGood you want the TextField, then click the “Search” button, a dialog of the BakedGood you want will appear.
If there are multiple BakedGood containing this origin, then all BakedGood containing it will appear.

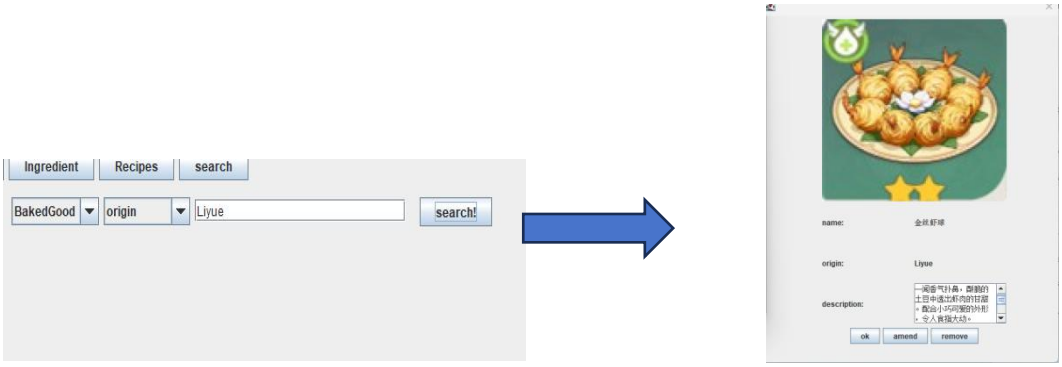


Figure 19: search BakedGood by origin show

If you choose “description”, enter the description of the BakedGood you want the TextField, then click the “Search” button, a dialog of the BakedGood you want will appear.
If there are multiple BakedGood containing this description, then all BakedGood containing it will appear.

Figure 20: search BakedGood by description show

If the first ComboBox select item is Ingredient, the second ComboBox will appeal three Item to choose:
A.name
B.description
C.calories
If you choose “name”, enter the description of the ingredient you want the TextField, then click the “Search” button, a dialog of the ingredient you want will appear.

Figure 21: search ingredient by name show

If you choose “description”, enter the description of the ingredient you want the TextField, then click the “Search” button, a dialog of the ingredient you want will appear.
If there are multiple ingredients containing this description, then all ingredient containing it will appear.

Figure 22: search ingredient by description show

If you choose “calories”, enter the description of the ingredient you want the TextField, then click the “Search” button, a dialog of the ingredient you want will appear.
If there are multiple ingredients containing this description, then all ingredient containing it will appear.

Figure 23: search ingredient by calories show

If the first ComboBox select Item is Ingredient, the second ComboBox will appeal two Item to choose:
A.name
B.calories

If you choose “name”, enter the name of the recipe you want the TextField, then click the “Search” button, a dialog of the recipe you want will appear.

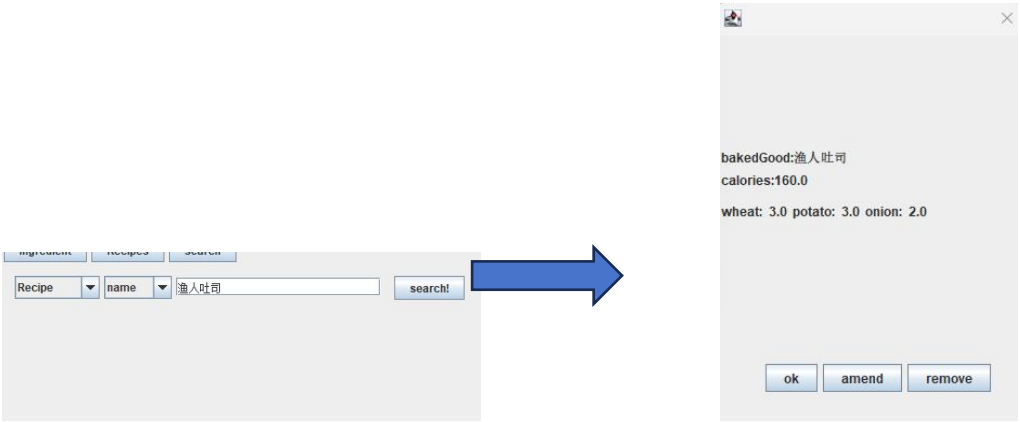


Figure 24: search recipe by name show

If you choose “calories”, enter the calories of the recipe you want the TextField, then click the “Search” button, a dialog of the recipe you want will appear.
If there are multiple recipes containing this “calories”, then all recipe containing it will appear.

Figure 25: search recipe by calories show

4.1.5 ComboBox for different type of sort

This ComboBox provides different types to sort the Item contain in the panel.

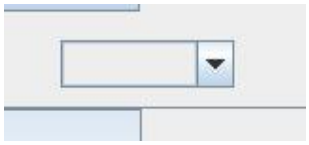


Figure 26: sort ComboBox show

If you are in BakingPanel, the item showed in the ComboBox is
A.name
B.origin
C.description

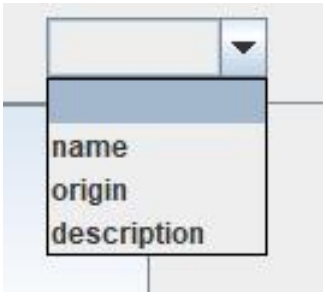


Figure 27: sort ComboBox in BakedGoodPanel show

If you choose “name”. The item showed in BakingPanel will be sorted by name.

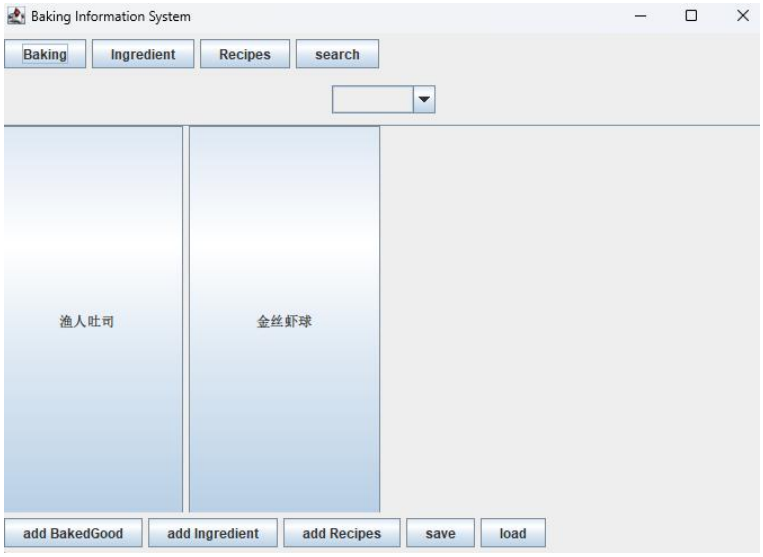


Figure 28: BakedGood after click sort ComboBox “name”

If you choose “origin”. The item showed in BakingPanel will be sorted by origin

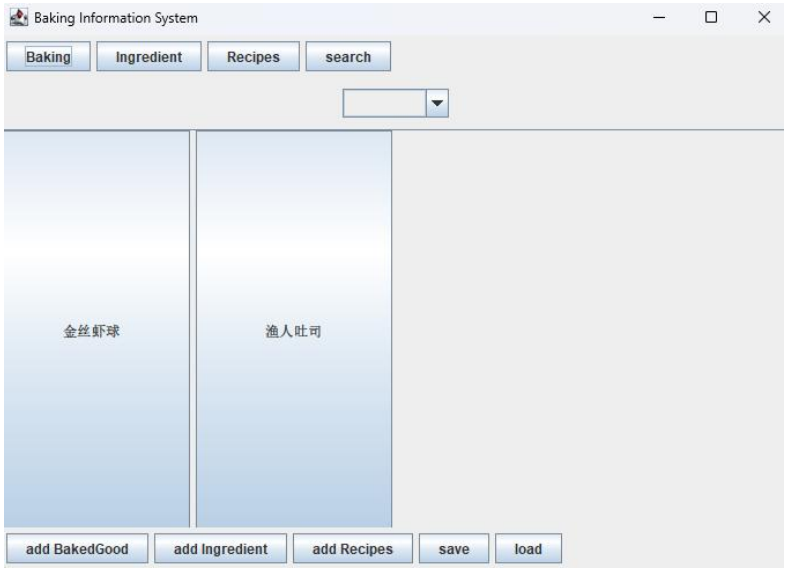


Figure 29: BakedGood after click sort ComboBox “origin”

If you choose “description”. The item showed in BakingPanel will be sorted by description.

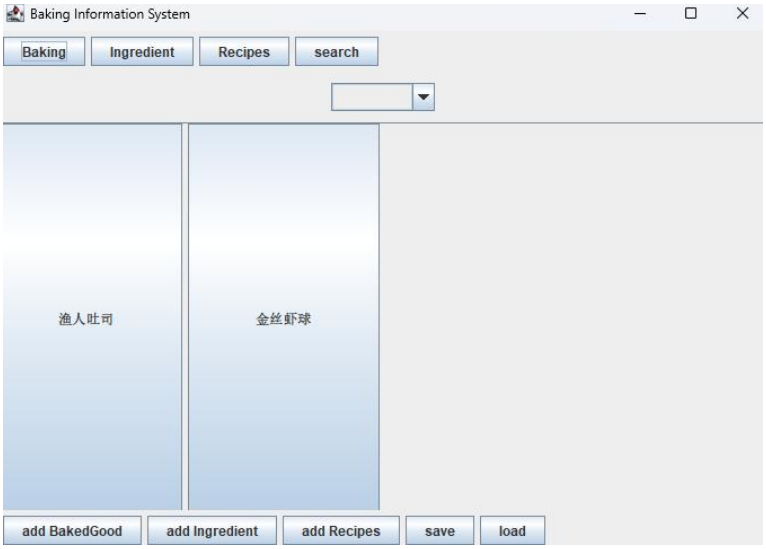


Figure 30: BakedGood after click sort ComboBox “description”

If you are in IngredientPanel, the item showed in the ComboBox is:
A.name
B.description
C.calories

Figure 31: sort ComboBox in IngredientPanel show

If you choose “name”. The item showed in IngredientPanel will be sorted by name.

If you choose “Total calories”. The item showed in RecipePanel will be sorted by total calories.

4.1.6Add BakedGood Function

Click the button at the bottom, the addBaked panel will be showed in MainWindow.

Figure 32: add BakedGood Panel show

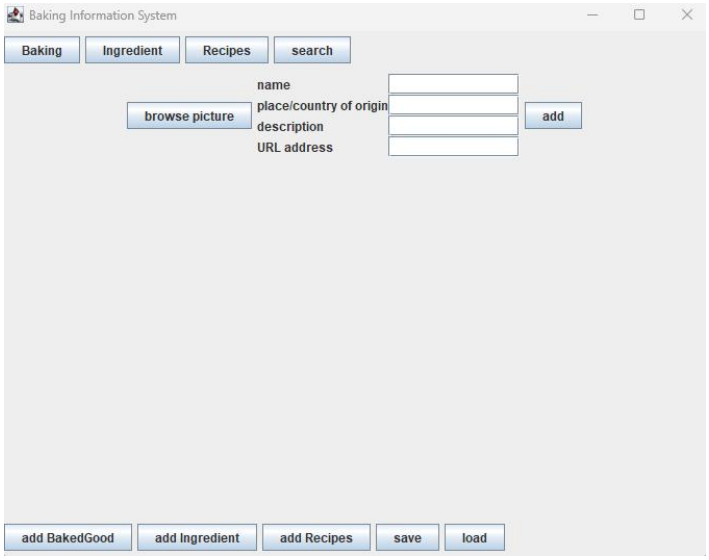


Figure 33: add BakedGood Panel show

In this panel, you can enter name, place/origin, or description.

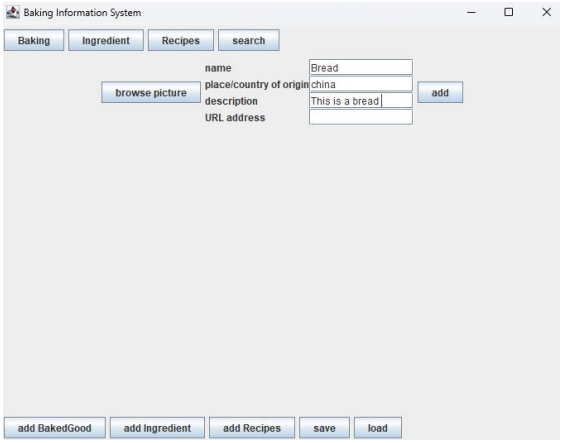


Figure 34: add new BakedGood show

Then click the browse picture button to browse URL from your computer.

Figure 35: browse button show

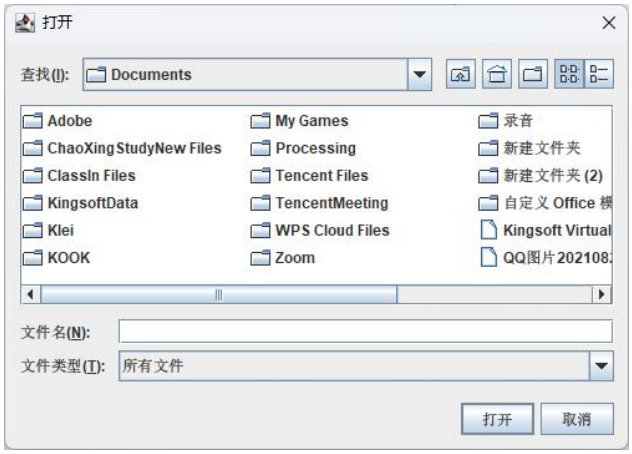


Figure 36: Window chooser show

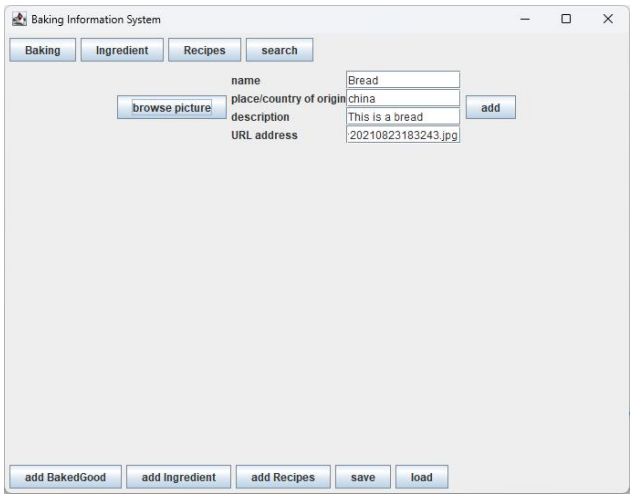


Figure 37: add completely show

Then click add Button.

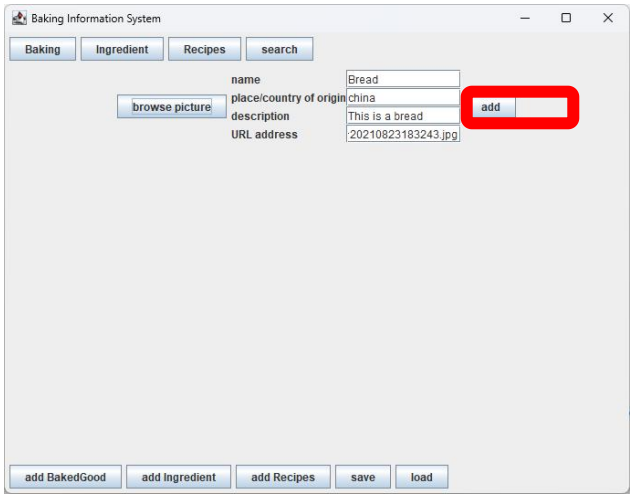


Figure 38: add button show

Click Baking button again, you can see a new item appear in PanelBaking.

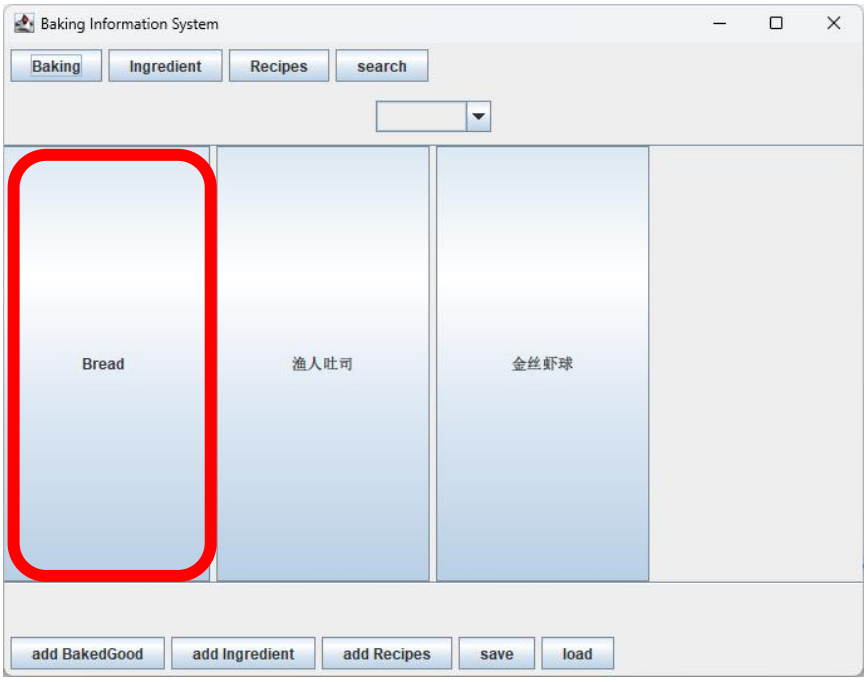


Figure 39: new item show

4.1.7 Add Ingredient Function

Click the button at the bottom, the addIngredient panel will be showed in MainWindow.

Figure 40: add ingredient Panel show

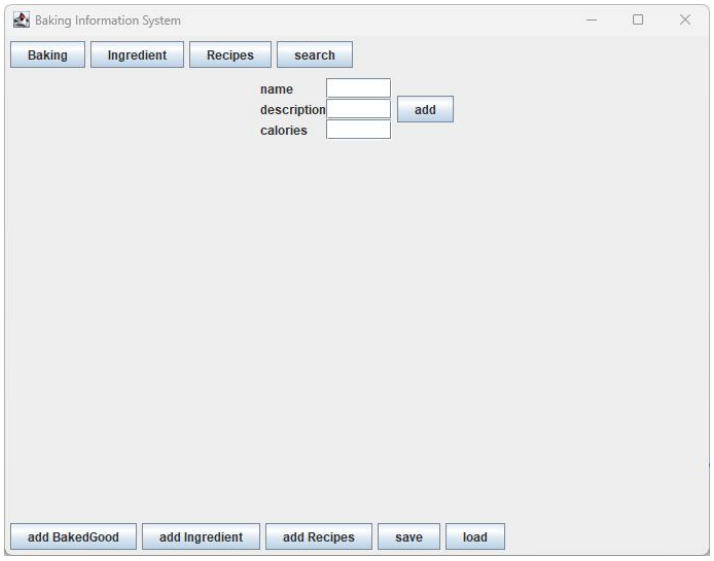


Figure 41: add Ingredient Panel show

In this panel, you should enter name, description and calories

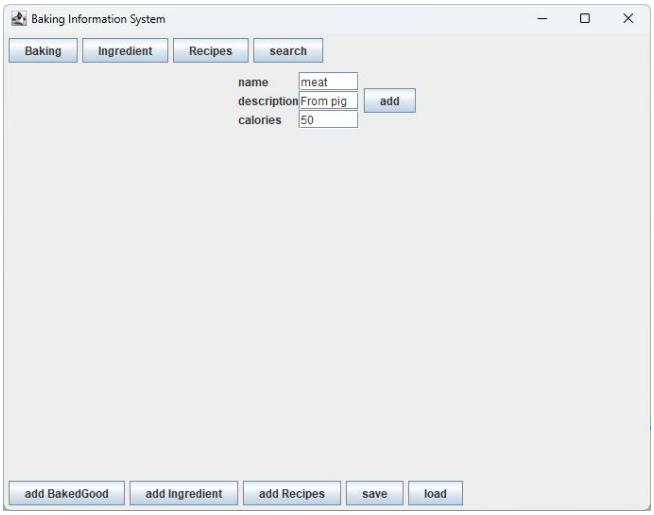


Figure 42: add new ingredient show

Then click add Button.

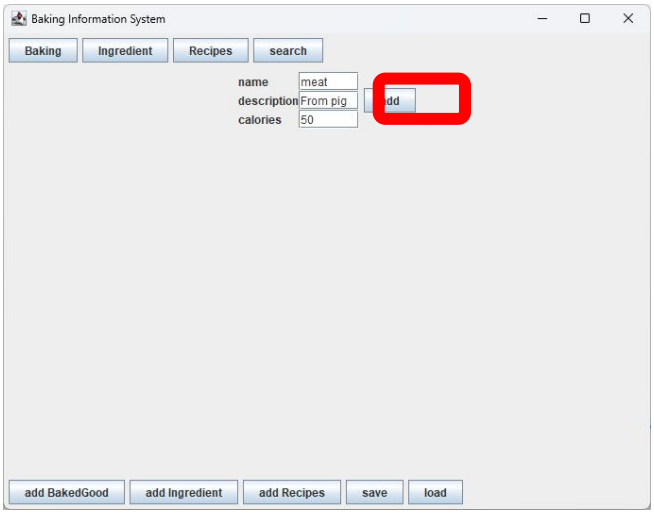


Figure 43: add button click show

Click Ingredient button again, you can see a new item appear in PanelIngredient..

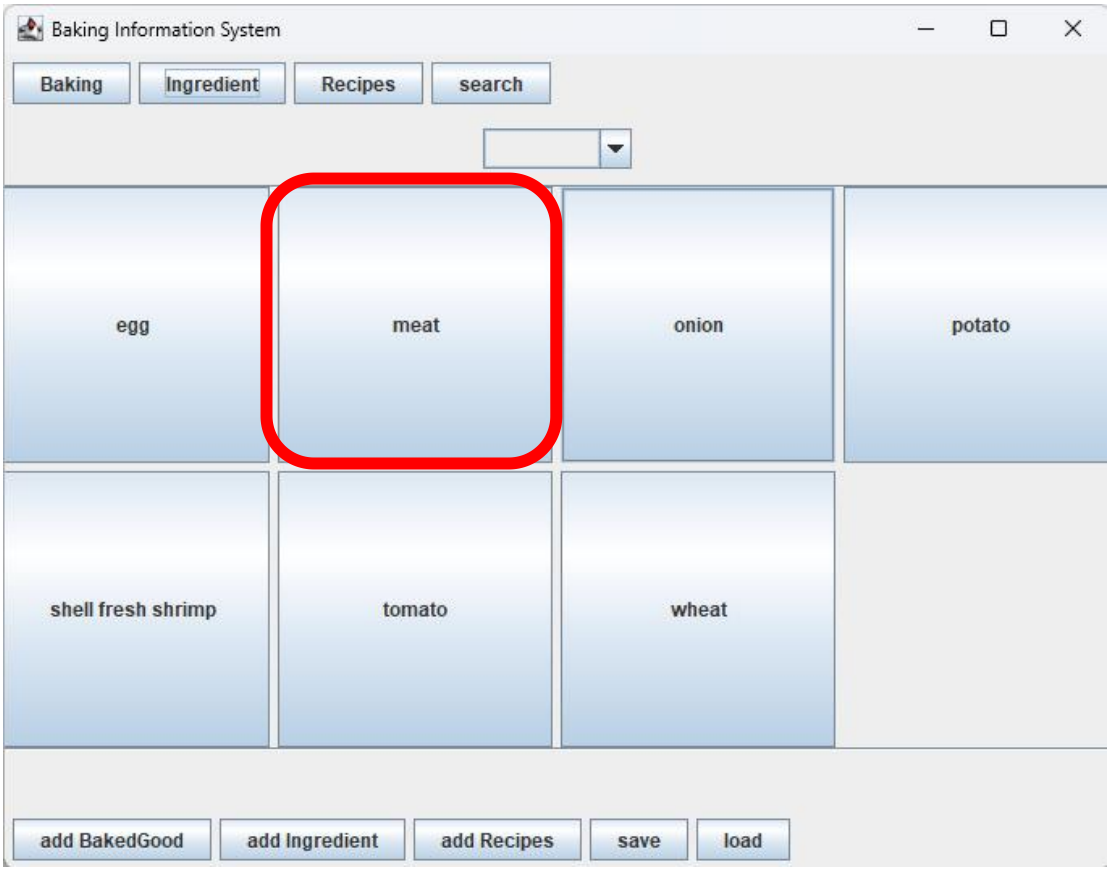


Figure 44: new ingredient show

4.1.8 Add Recipe Function

Click the button at the bottom, the addRecipe panel will be showed in MainWindow.

Figure 45: add recipe Panel show

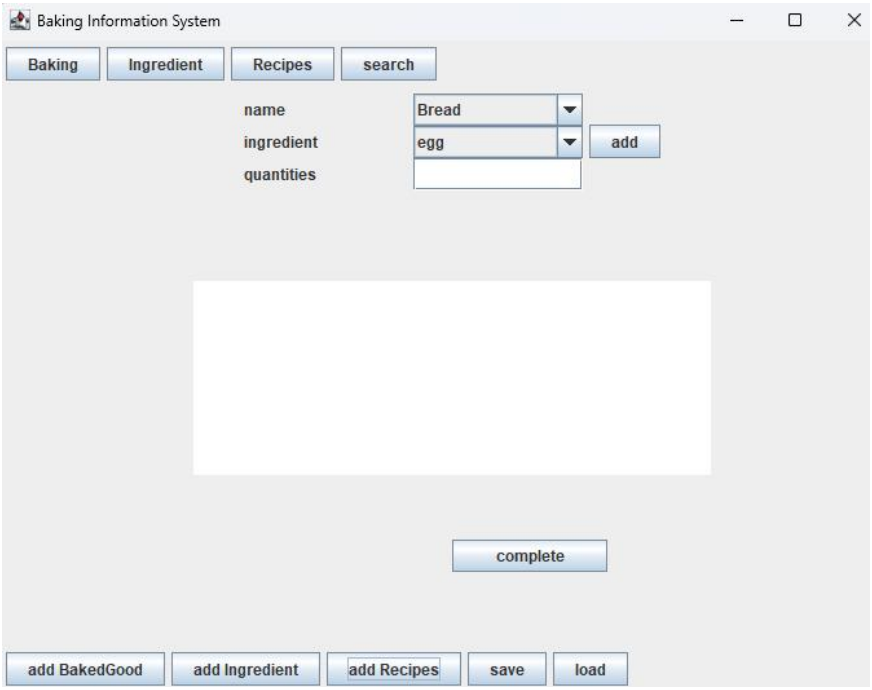


Figure 46: add recipe Panel show

In this panel, you can choose name from the BakedGoods have added.

Figure 47: choose name from BakedGood show

and then choose ingredient from the ingredients have added.

Figure 48: choose ingredient show

Enter the quantity of the ingredient

Figure 49: add quantity show

Click the add button

Figure 50: add show

You can see the ingredient has been added to this recipe

Figure 51: ingredient added show

You can add multiple ingredients to one recipe.

Figure 52: multiple ingredients added show

Click “complete” button.

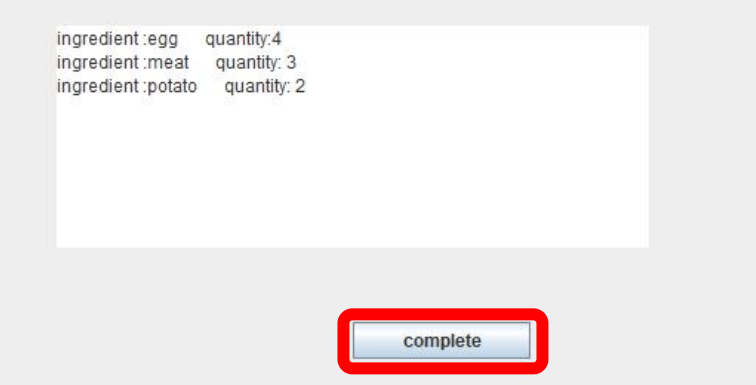


Figure 53: complete button show

Click Recipe button above again, you can see a new item.

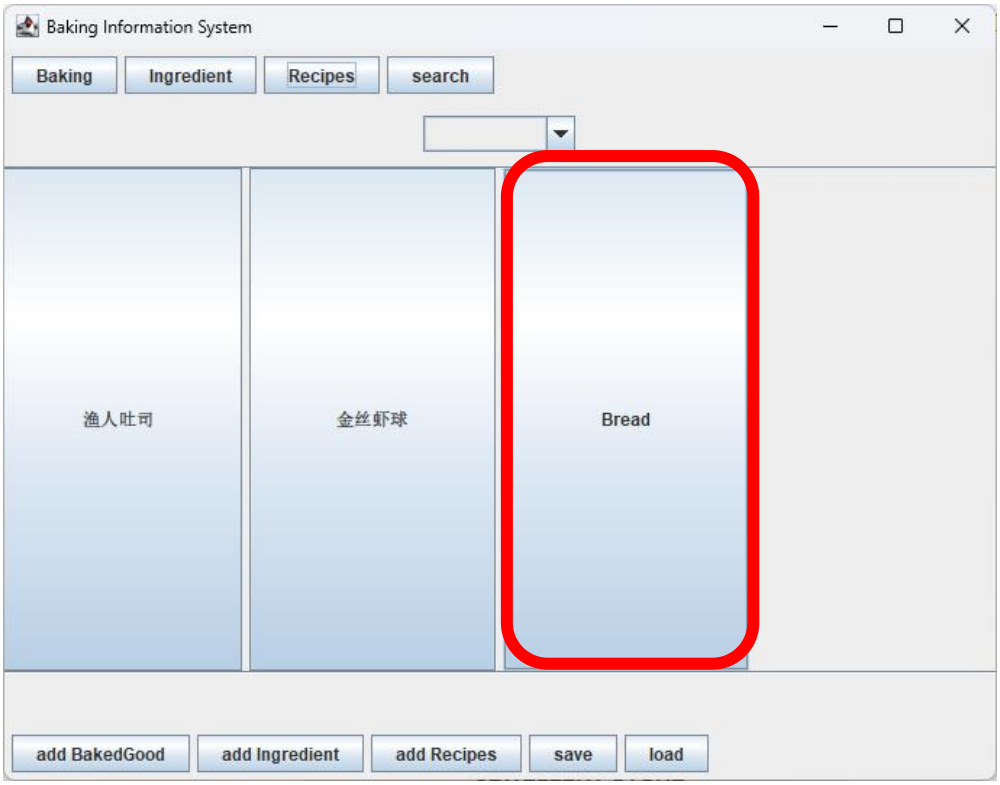


Figure 54: new recipe show

4.1.9 Save Function

Click the save button, the program will save the data of BakedGood, ingredient and recipe.

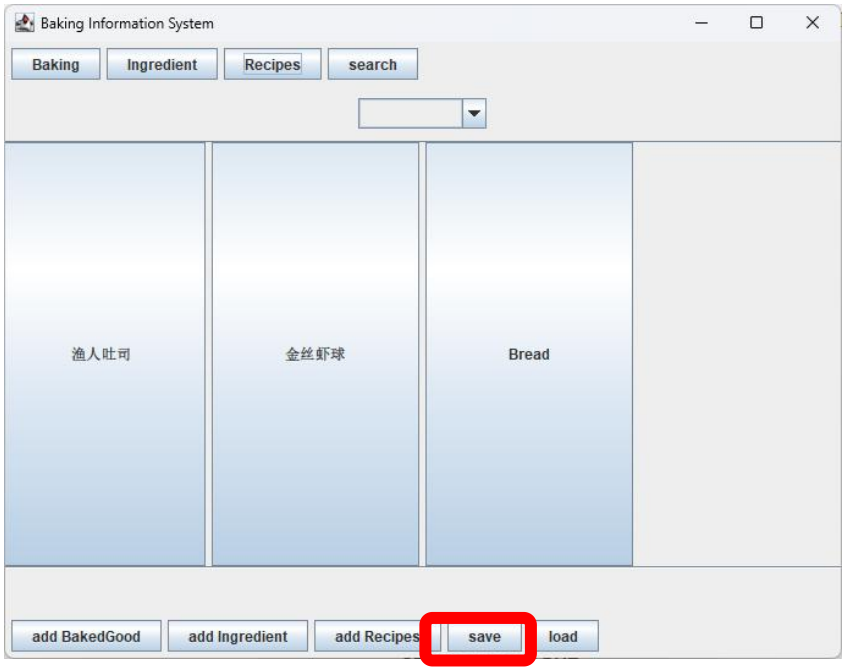


Figure 55: save button show

```
BAKEDGOOD,渔人吐司,mondstatd,null,C:\Users\黄家睿\Pictures\原神食物图片\1.jpg
BAKEDGOOD,金丝虾球,Liyue,油炸而成的虾料理。乍一闻香气扑鼻，酥脆的土豆中透出虾肉的甘甜。配合小巧可爱的外形，令人食指大动。,C:\Users\黄家睿\Pictures\原神食物图片\2.jpg
INGREDIENT,egg,null,50.0
INGREDIENT,onion,null,20.0
INGREDIENT,potato,null,30.0
INGREDIENT,shell fresh shrimp,null,50.0
INGREDIENT,tomato,null,20.0
INGREDIENT,wheat,null,10.0
RECIPE,渔人吐司,160.0
wheat,3.0
potato,3.0
onion,2.0
```

Figure 55: data before save

```
BAKEDGOOD,Bread,china,This is a bread ,C:\Users\黄家睿\Documents\QQ图片20210823183243.jpg
BAKEDGOOD,金丝虾球,Liyue,油炸而成的虾料理。乍一闻香气扑鼻，酥脆的土豆中透出虾肉的甘甜。配合小巧可爱的外形，令人食指大动。,C:\Users\黄家睿\Pictures\原神食物图片\2.jpg
INGREDIENT,egg,null,50.0
INGREDIENT,meat,From pig,50.0
INGREDIENT,potato,null,30.0
INGREDIENT,shell fresh shrimp,null,50.0
INGREDIENT,tomato,null,20.0
INGREDIENT,wheat,wheat,10.0
RECIPE,渔人吐司,160.0
wheat,3.0
potato,3.0
onion,2.0

RECIPE,金丝虾球,320.0
potato,4.0
shell fresh shrimp,4.0

RECIPE,Bread,410.0
potato,2.0
meat,3.0
egg,4.0
```

Figure 56: data after save

4.1.10 Load Function

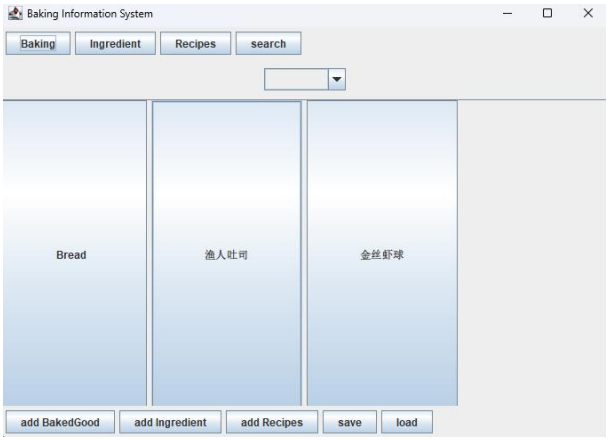


Figure 57: PanelBaking after load show

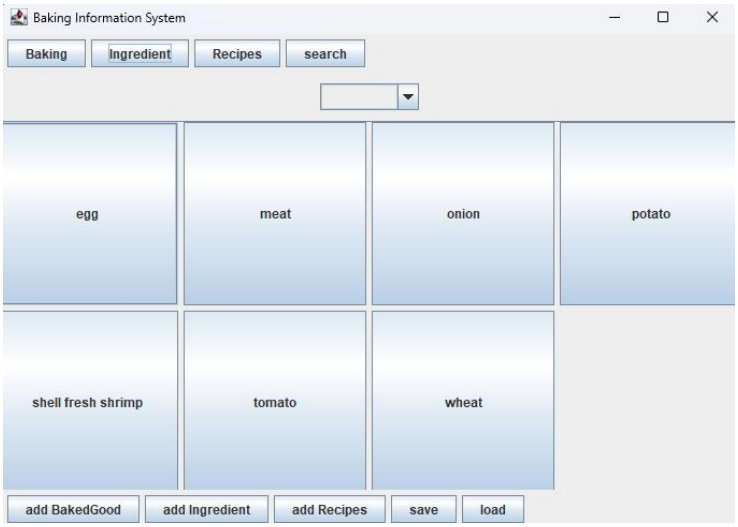


Figure 58: PanelIngredient after load show

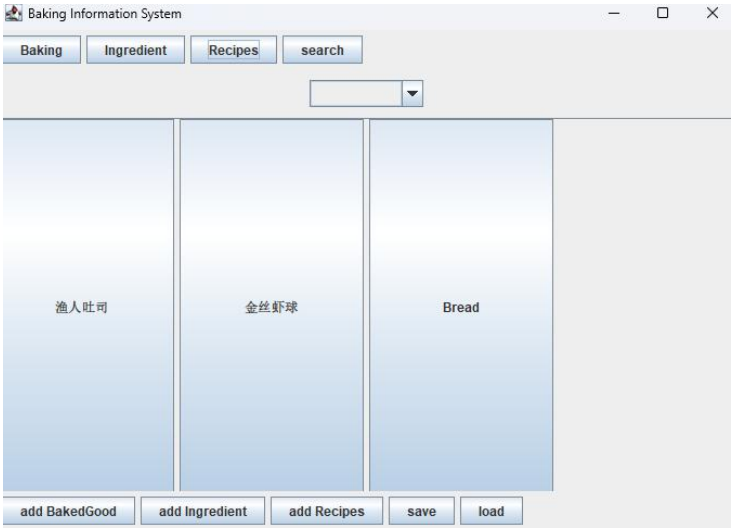


Figure 59: PanelRecipe after load show

4.1.11 BakingGood details

Click the item in Panel Baking, a dialog about this BakedGood detail information will appear.

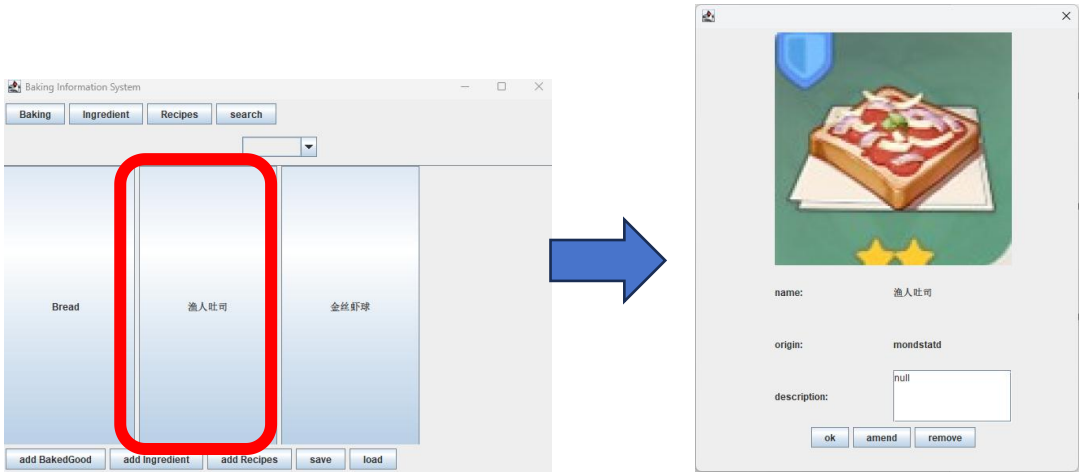


Figure 60: DialogBakedGood show

In the dialog, there are three buttons, “ok” can close this dialog, “amend” can amend information about this BakedGood, “remove” can delete this BakedGood.

4.1.11.1 Amend

Click the “amend’ button, the amend dialog will appear.

Figure 61: amend button show

Figure 62: amend dialog show

You can amend name, origin, description and picture address in this dialog.

Figure 63: amend name show

Then click ok button

Figure 64: amend ok button show

You can see the item name has been changed.

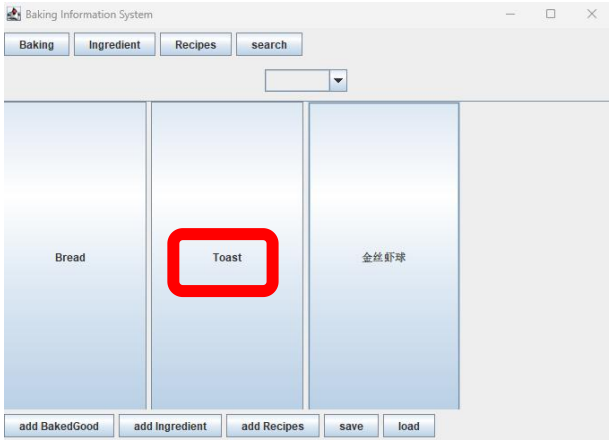


Figure 65: name change show

4.1.11.2 Remove

click this button, you can see this item has been deleted.

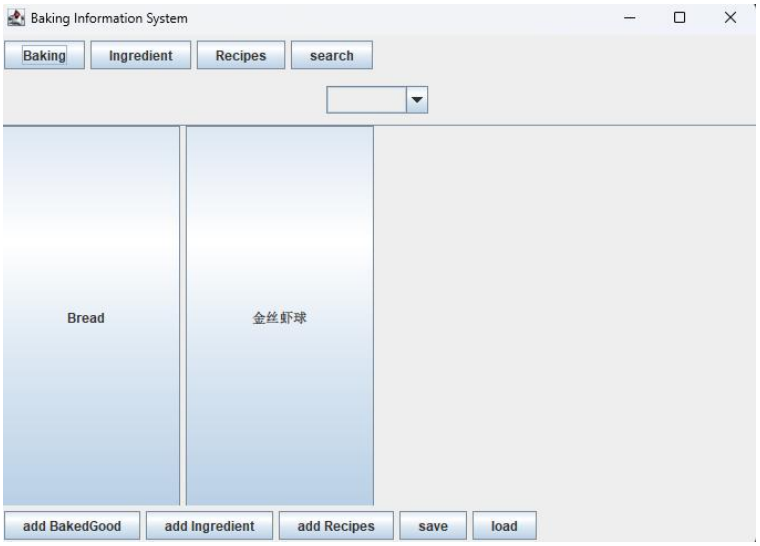


Figure 66: remove show

4.1.12 Ingredient details

Click the item in Panel Baking, a dialog about this BakedGood detail information will appear.

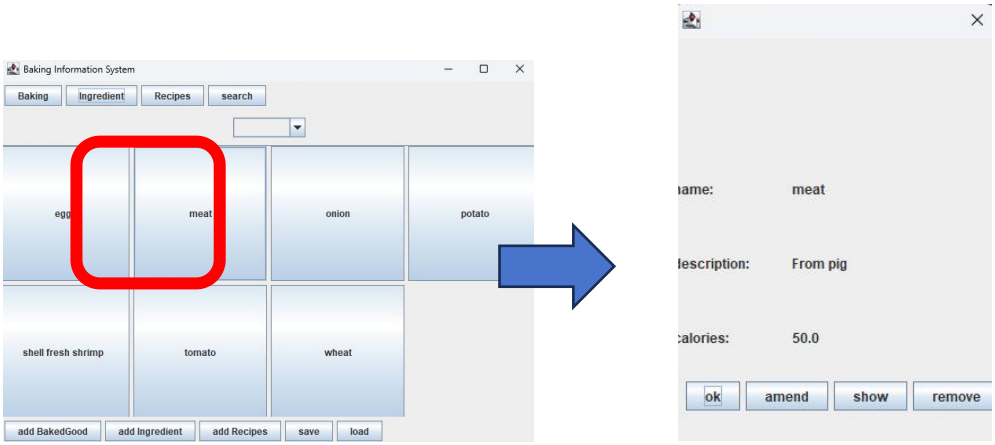


Figure 67: DialogIngredient show

In the dialog, there are four buttons, “ok” can close this dialog, “amend” can amend information about this BakedGood, “show“ button can show all BakedGood contain this ingredient and remove” can delete this BakedGood.

4.1.12.1 Amend

click the “amend’ button, the amend dialog will appear.

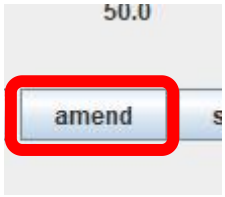


Figure 68: amend button show in ingredient change dialog

Figure 69: amend dialog show

You can amend name, description and calories in this dialog.

Figure 70: amend name show

Then click ok button.

Figure 71: amend ok button show

You can see the item name has been changed.

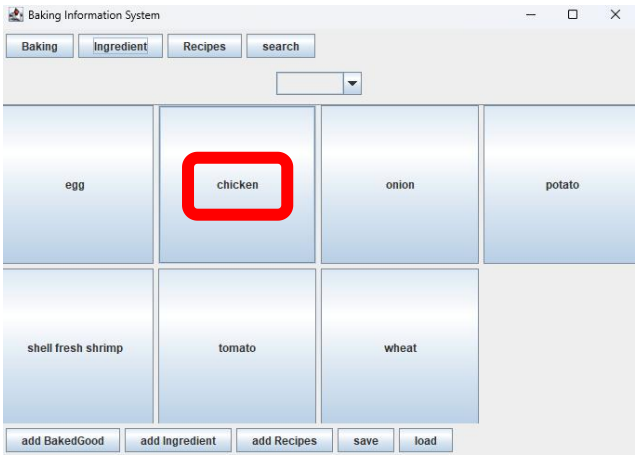


Figure 72: ingredient change show

4.1.12.2 Show all baked goods contain this ingredient

Click “show” button, all BakedGoods contain this ingredient will be displayed.

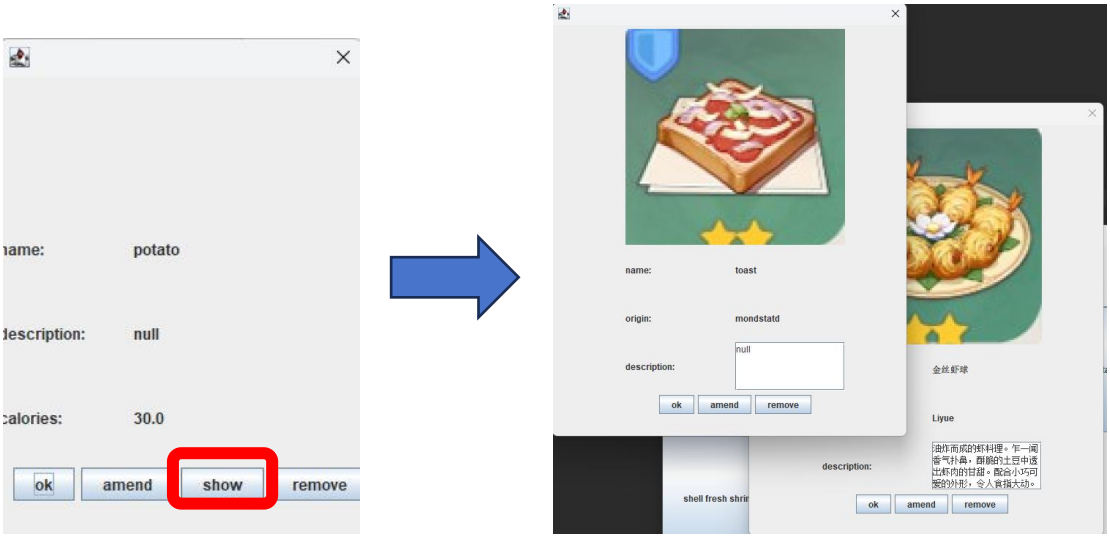


Figure 73: show button show

4.1.12.3 remove

Click this button, you can see this item has been deleted.

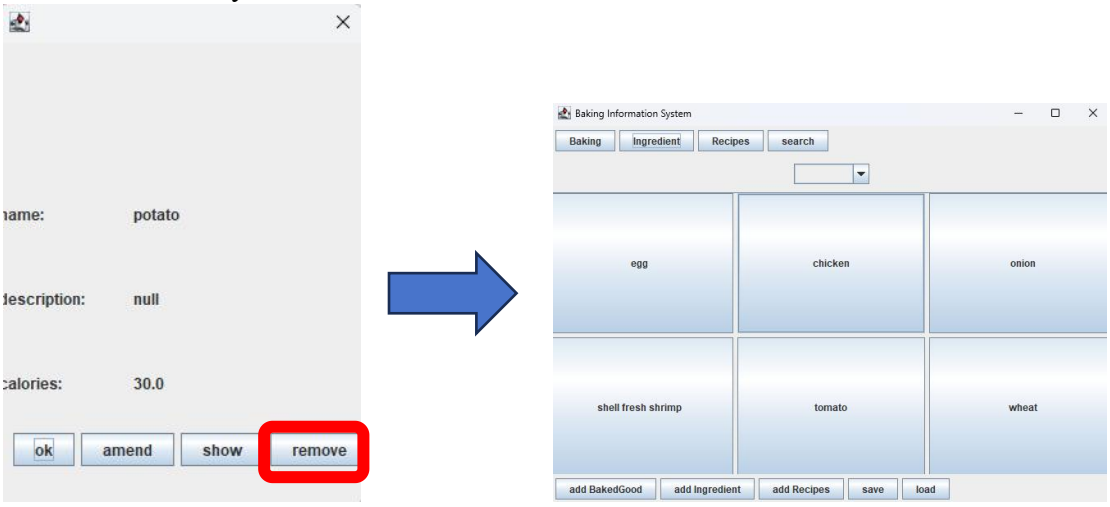


Figure 74: remove button show

4.1.13 Recipe details

Click the item in Panel Baking, a dialog about this BakedGood detail information will appear.

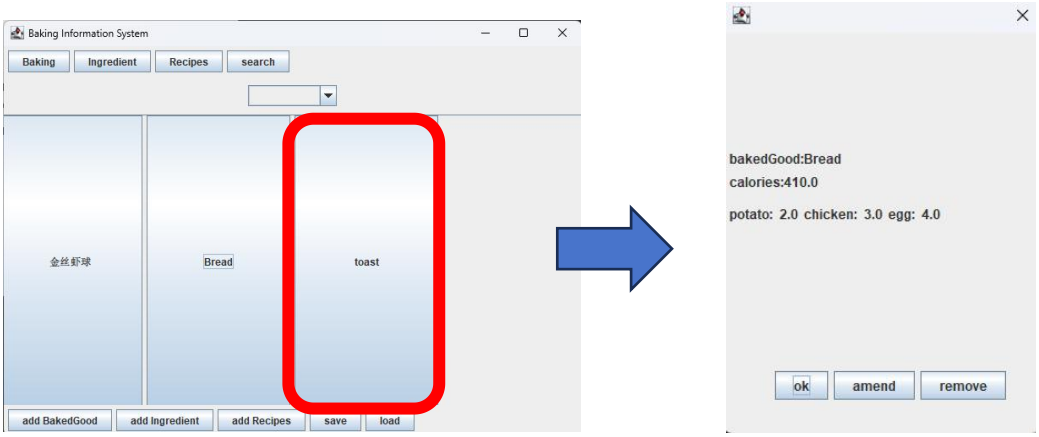


Figure 75: DialogRecipe show

In the dialog, there are three buttons, “ok” can close this dialog, “amend” can amend information about this recipe, “remove” can delete this recipe.

4.1.13.1 Amend
click the “amend” button, the amend dialog will appear.

Figure 76: amend button show

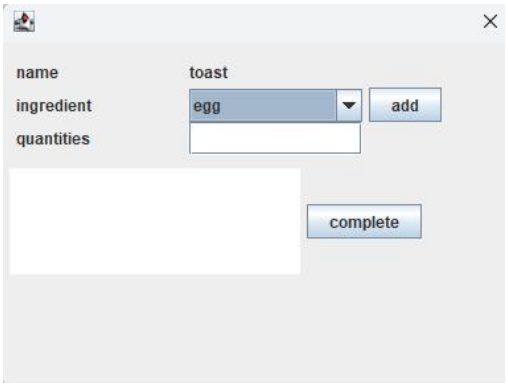


Figure 77: amend dialog show

You can amend name, ingredient and quantity in this dialog.

Figure 78: amend name show

Then click complete button.

Figure 79: amend complete button show

You can see the ingredient and total calories have been changed.

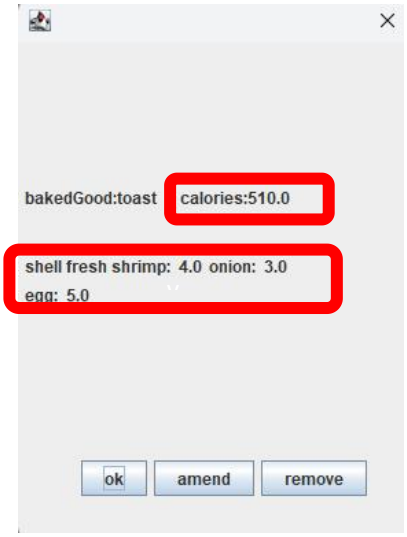


Figure 80: change show

4.1.13.2 Remove
Click this button, you can see this item has been deleted.

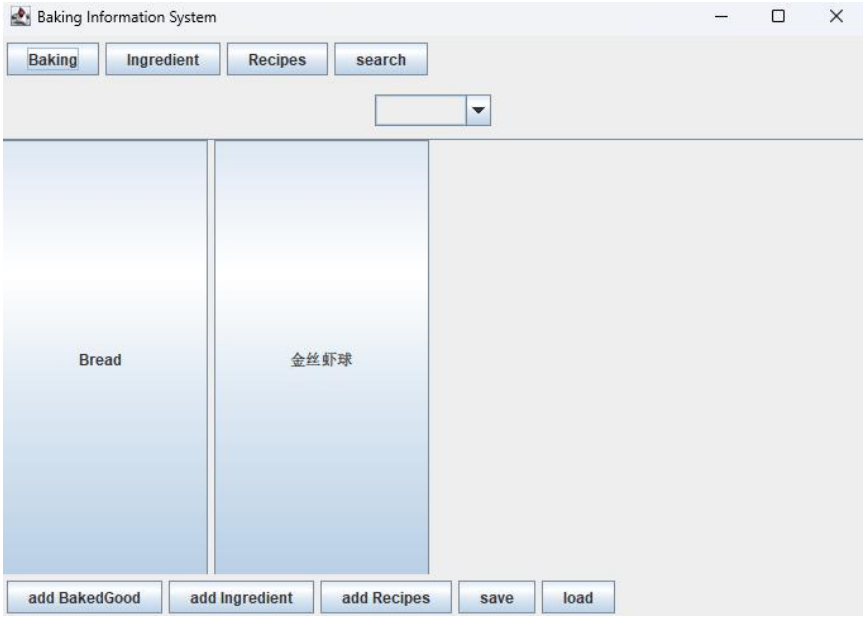


Figure 81: remove show

4.2 JUnit test

In test part, we use Junit to test out method. We define a test source folder in my project, and create the Junit class for each class in our BakingSystem. Because the Junit can efficiently test the method in java packet , so we use it to test method and get the result to make sure all method can run correctly . Also it can cut down the memories occupation which will occur when we use main method to test all method. And we can use Junit to fix our mistake which occur when we test the method .

Here are some example :

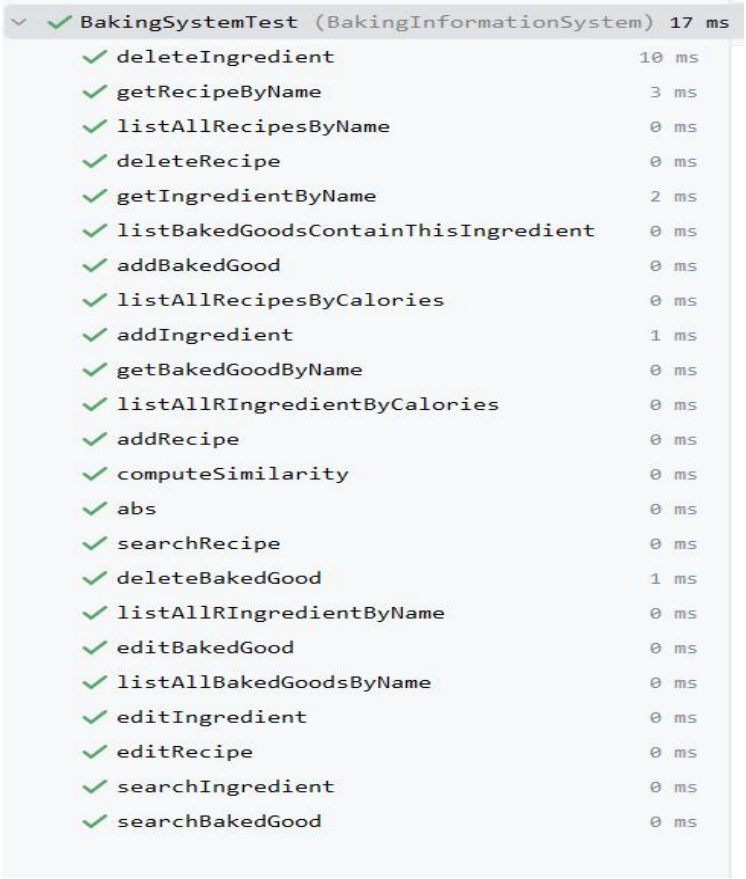
First , we can use the button which on the class left , it show that the operation we can do to the class.

We choose run this test.

We can see this picture , we test the method called “Baked Good “,and in this test class ,we cite all the method in Baked Good class. For example :

```
public void sortBakedGoodsByName() {
    bakedGoodLists.addLast(bakedGood1);
    bakedGoodLists.addLast(bakedGood2);
    BakedGood.sortBakedGoodsByName(bakedGoodLists);
    assertEquals( message: "wrong",bakedGoodLists.getDataByIndex(1).getName(), actual: "bread");
}
```

In this test we add two object to bakedgoodlists, and we invoke the sort method ,and we use assert to test the method , and compare the expect result with the actual result.



According those pictures , we can know our methods all run correctly , and all results are right.

4.3 Analysis of the algorithmic complexity

4.3.1Add Operations (addBakedGood, addIngredient, addRecipe):

Time Complexity: O(1)
Space Complexity: O(1)

These operations rely on the performance of the hash function.The space complexity is constant as well since it involves memory allocation for the new elements.

4.3.2 Edit Operations (editBakedGood, editIngredient, editRecipe):

Time Complexity: O(1)
Space Complexity: O(1)
Similar to add operations, editing involves hash table lookups and modifications. In the average case, these operations are constant time. The space complexity is constant since it involves updating existing elements.

4.3.3 Delete Operations (deleteBakedGood, deleteIngredient, deleteRecipe):

Time Complexity: O(1)
Space Complexity: O(1)

4.3.4 List Display Operations (listAllBakedGoodsByName, listAllRecipesByName, listAllRecipesByCalories, listAllRIngredientByName, listAllIngredientByCalories):

Time Complexity: O(n log n)
Space Complexity: O(1)

4.3.5 Search Operations (searchBakedGood, searchIngredient, searchRecipe):

Time Complexity: O(n)
Space Complexity: O(1)
Searching involve traversing the entire list . In the worst case, it takes linear time. Space complexity is constant as it only requires storage for the search result.

4.3.6 String Similarity Calculation Operation (computeSimilarity):

Time Complexity: O(m + n), where m and n are average lengths of the two strings
Space Complexity: O(1)
The time complexity involves splitting the strings into words and calculating the common words, which is O(m+n). Space complexity is constant as it does not depend on the size of the input.

Conclusions

All in all,we completed the project successfully.In the program,we use the knowledge of data structure,and use some smart and advanced methods to make the whole more reasonable.In the process of completing this project,we cooperated with each other and had a clear division of labor.In the process of doing the project,we also learn additional knowledge such as GUI making.

In this collaborative team task, all members gained profound insights. Firstly, we acquired the art of collaboration and collective learning. Faced with challenges in the program, we assisted each other, collaboratively identified and rectified errors. Secondly, we effectively allocated tasks, ensuring each team member shouldered a reasonable workload, thereby ensuring the team's efficient operation. Furthermore, we refined our team assignment through consultations with teachers and other means.

Within the scope of the assignment, we integrated knowledge seamlessly. This comprehensive approach not only enhanced our problem-solving capabilities but also provided valuable experience in applying theoretical knowledge to practical problems. This collaborative team task is not merely an accumulation of project experience; it is a comprehensive exercise in team collaboration and knowledge integration.

Acknowledgments

We would like to express our sincere gratitude to Dr.Guoqing Lu, SETU, for his invaluable guidance and support throughout the course of this research. His expertise and insightful feedback have significantly contributed to the refinement and depth of this work.

We want to express our heartfelt thanks to our entire team, including ZihengWang, JiaruiHuang, KaifengJin, and WeichengGuo, for their dedication and collaborative efforts. Their constructive criticism and meaningful discussions have significantly enriched the intellectual environment surrounding this work.

Lastly, we want to express our heartfelt thanks to our family and friends for their unwavering encouragement and understanding during the challenging phases of this work .

This work was made possible through the collaborative efforts of all those mentioned above, and we are truly appreciative of their contributions.