

Master Thesis

Design and Implementation of a Database for Recipes

November 2004

LinLin Wang (s020953)



Supervisor: Paul Fischer

**Informatics and Mathematical Modelling
Technical University of Denmark
Kgs Lyngby, Denmark**

IMM – THESIS – 2004 - 82

Abstract

With the rapid growth of the Internet technology, the information boom has occurred in the human world with the irresistible trend. When the accumulation speed of the information is faster than the digestion speed of that, the database came into being to store the information. How to search and extract the useful content from the Internet and store them into what kind of databases are the most interesting topics for the database programming. According to the evolution trend of the Internet, the intelligent analysis engine is a powerful and efficient tool to establish a structural semantic-oriented web.

This project takes the recipe storage system as an example for looking into the design of the semantic-oriented extraction. The objective of this project is to establish a database for recipes and to fill it with some data. During the project, I designed and implemented two solutions to achieve the above objective. One is based on the traditional thought and relies on some artificial marks to extract the target content; the other one is introduced into some thoughts of the semantic-oriented extraction, and able to extract the recipe files more intelligently and accurately. The test results prove that the semantic-oriented extraction is more effective.

Acknowledgements

First I would like to thank my supervisor--- Paul Fischer, the professor from Informatics and Mathematical Modelling at the Technical University of Denmark. He helped me to set up the definition of my project, and gave me the illuminative guidance throughout the project. And then I want to thank Jens Thyge Kristensen, who is also a professor from IMM, DTU. He gave me the kind help on the area of java programming and object-oriented design. I also appreciate one of my special friends who provide me many useful advice and kind help during the project.

In addition, I want to thank my parents for their support and encouragement during my study in Denmark, and all my friends who have been sharing the joys and tears with me!

The last but not least, I also want to thank my boyfriend, Bo Jiang, thank for his considerate care and generous support!

Terminology

Recipe Data	The content of the recipe, such as the recipe tile, ingredients and direction.
Ingredient Description Line	A piece of ingredient description, which normally consists of quantity, unit and ingredient description.
Quantity	The numerical description of the ingredient, e.g. '5'
Unit	The unit used for measuring the ingredient, e.g. 'cup'
Ingredient Description	The text used for describing the ingredient in an ingredient description line
Entire Paragraph	A paragraph text without any empty lines in between
Word	An English word or an Arab number or a text string which doesn't contain any spaces, e.g. one, 3, 1/4, and 1.5
Signature	the words that indicate the ingredient or direction of the recipe, e.g. 'Ingredient' and 'Direction'
Material	The name of the specific ingredient, e.g. 'milk'

Table of Contents

1. Introduction.....	9
1.1 Project Statement	9
1.2 Problem Analysis	9
1.3 Report Structure	10
2. Solution 1	11
2.1 Requirement Analysis	11
2.1.1 User Requirement Analysis	11
2.1.2 Data Requirement Analysis	11
2.2 System Analysis and Specification	19
2.2.1 Import Functionality	19
2.2.2 Database System	24
2.2.3 Graphics User Interface	35
2.3 System Implementation	37
2.3.1 System Architecture.....	37
2.3.2 Microsoft Access Databases Design and Implementation.....	39
2.3.2 Model Implementation.....	42
2.3.3 View Implementation.....	61
2.3.4 Controller Implementation.....	62
2.4 System Test and Results	70
2.5 Summary	84
3. Solution 2	85
3.1 Analysis.....	85
3.1.1 HTML Document Analysis.....	85
3.1.2 External Recipe Files Analysis	86
3.2 Design and Specification	90
3.2.1 Parsing the HTML Document.....	90
3.2.2 Extraction.....	91
3.2.3 Inserting the Recipe into the Database.....	93
3.3 Implementation	94
3.3.1 The Overview of the Implementation	94
3.3.2 The Implementation of Parsing the HTML Document.....	96
3.3.3 Extraction.....	100
3.3.4 Inserting the Recipe into the Database.....	104
3.4 Results and Test	106
3.4.1 Import the Invalid Recipe File	106
3.4.2 Import the Valid Recipe File.....	107
3.5 Summary	113
4. Conclusion	115
4.1 Future Work	115
4.2 Personal Conclusion.....	115
Reference	117
List of Figures.....	119
List of Tables	121
Appendix I Installation Guide	122
Appendix II Configuration of Source Code	125
Appendix II Test Results.....	127

1. Introduction

1.1 Project Statement

The objective of this project is to design and implement a recipe system. The core of the system is a database which is used to store the recipe information; the front-end of the system is a set of GUI (Graphical User Interface) applications which act as a bridge in between the end users and the system.

Generally speaking, the recipe database should be able to store the recipe information, which normally includes the title, the category, the ingredients and the direction of the recipes. The database should also support the following functions:

- Insert a new recipe record manually
- Modify the items of the recipe record manually
- Delete the recipe record manually
- Import the external recipe files automatically
- Search the recipe by the category manually
- Search the recipe by the ingredients manually
- Search the recipe by the title manually

In addition, as the recipe system supports two kinds of users (the general user and the super user) and offers them different access rights, the database should be able to store the user information (the user name and password) and provide the functions like:

- Modification of the password of the super user

In the front-end client side, the system should offer the “win-form” based GUIs, which allow the users easily fill in the input, select and choose the functions of interest, and read the results. For the general user, the GUI application should provide all the search functions and display the search results. For the super user (administrator), the GUI application should provide the authentication window which is used to authenticate and authorise the super user. Moreover, the super user should be able to manage an update the recipe database and have the access for the full functions of the system, such as inserting, modifying and deleting the recipe.

1.2 Problem Analysis

From the project statement, we can see the most interesting topics for this project are how to search and extract the recipe content from the external files and store them into what kind of databases. As the external files can be in various formats and the layout of the recipe content can be quite different, it is necessary to design a general extraction system, which can handle as more as possible recipe files.

The general extraction method, in essence, is based on the principle of the semantic-oriented analysis. Here the semantic-oriented analysis means the system can understand the ‘organic’ structure of the recipe files and know what the recipe files are describing about. Thus, the system can easily handle all kinds of recipe files in the right way.

In order to achieve the semantic-oriented analysis, the system should first be able to ‘read’ and ‘recognize’ the recipes. In other word, the system should know the recipes normally consist of the following parts: the title, ingredients and direction. In the further

step, the system should understand which words are describing the ingredients or direction and what the ingredient or direction description really means.

Therefore, in this project, I put my main effort on the work of solving how to design and implement an intelligent extraction system, which is able to read and understand the recipe files like the human being.

1.3 Report Structure

The outline of the rest content is:

Chapter 2 – Solution 1 mainly describes how this recipe database system is specified, designed and implemented. It includes requirement analysis, design and specification, implementation, results and test, and summary.

Chapter 3 – Solution 2 mainly describes how the import function is improved and optimized. It includes analysis, design and specification, implementation, results and test, and summary.

Chapter 4 – Conclusion summarizes the report and project work from the general perspective and gives a view about the future work.

2. Solution 1

2.1 Requirement Analysis

The objectives of this requirement analysis are:

- What is this recipe database used for?
- What kind of functions does the database offer?
- How to specify the access right for the database?
- Browse plenty of recipe web sites, and analyze the recipe file structures, distinctions, contents etc.
- What kind of affects can be reached to the *Import* function?
- What kind of Graphic User Interface can be supplied?

2.1.1 User Requirement Analysis

Generally speaking, there are two user groups that will use this recipe database system: general user and administrator.

2.1.1.1 General User

General User here points to all the people who will use this database system. It should include all the internal staffs if the system is used for some local area and it should include all the internal and external persons if the system is used in public. The general user will search corresponding recipes depending on their interest by inputting all kinds of criteria, such as: recipe title, recipe category, and some ingredients.

2.1.1.2 Administrator

Administrator, i.e. the super-user, includes those peoples who are authorized to this database system and are permitted to modify this database. The administrator can manage and manipulate this database freely. The administrator can do the following operations to this database: insert recipe, edit recipe, delete recipe, import recipe and modify user's settings.

2.1.2 Data Requirement Analysis

In this recipe database system, an import function should be implemented. Below, let's discuss the problem of it.

2.1.2.1 Import Functionality Analysis

Import function is one of the most important functions of this system. It should include the information extraction technology, which is researched for developing and implementing human languages extraction. The import function should enable administrator to import recipes into database from external recipe files automatically. So in this section, I focus on analyzing external recipe files, including its structure, contents, and other attributes.

Generally, recipe files might be obtained from many places, such as floppy disk, CD, and internet; it may be stored in different formats (such as Doc, Html, txt and etc.). In solution 1, we just assume all the recipe files are searched and downloaded from the web site, and then should be saved as *.txt format into the local hard disk.

Because the recipe files were obtained from the web sites, plenty of text information might exist, not only the recipe description, but also the information like advertisements and some other links. For the recipe database, the recipe files consist of both useful and useless information (of course sometimes just useless information exist). What the system has to do is to recognize the useful information and extract them.

Before extracting information, we have to analyze the external recipe source files. After analyzing plenty of recipe web pages on the internet, some general rules of recipe files were concluded as below (all the recipe files have been saved as *.txt into the local place):

▪ **Content:**

1. Almost all the recipe files consist of four main parts: the title, the category, the ingredient, and the direction. A few recipe files include some comments.
2. The recipe title appears at the random place but not some fixed place.
3. Recipe category is partitioned in all kinds of ways, such as: depending on the recipe region or the recipe main ingredient etc. The recipe category might be pointed out in some files or might be not in others.
4. Several recipe files include special signatures to indicate ingredient and direction paragraphs¹, the special words might be 'ingredient' or 'direction' or 'instruction' or 'procedure'.

Refer to

5. Recipe ingredient description consists of 3 parts: quantity, unit and the ingredient description.

¹ Refer to <http://cake.allrecipes.com>
<http://search.yummy.com/recipe.htm?ID=8632>
http://cookbook.rin.ru/cookbook_e/recipes/0838985.html
<http://www.recipecenter.com/Recipe.asp?Code=27>
<http://www.ichef.com/recipe.cfm/>

6. Most of the direction parts are displayed in one paragraph²; a few exceptions exist as well.

▪ **Structure:**

1. In the recipe files, the recipe description is always displayed in this order:

The recipe title

The recipe category (some recipe file doesn't offer the recipe category)

The recipe ingredient description

The recipe direction description

Some recipe files also include some comments somewhere.

2. Almost all the recipe ingredient part is displayed in one paragraph; this means there are no empty lines in between the descriptions.

3. Almost all the recipe ingredient description is written in following order:
quantity, unit, some ingredient descriptions.

4. Almost all the first words are numerical in each line of the recipe ingredient description. For example:

1 1/3 cups flour

1/2 tsp salt

1 1/3 tsp baking powder

1 1/3 tsp baking soda

Some exceptions also exist, for example:

Dash each salt and black pepper

Thickly sliced homemade-style white bread

² Refer to <http://www.recipesource.com/>
<http://www.allrecipes.com/>
<http://www.recipelink.com/>
<http://www.recipecenter.com/>

The normal text recipe file is shown below:

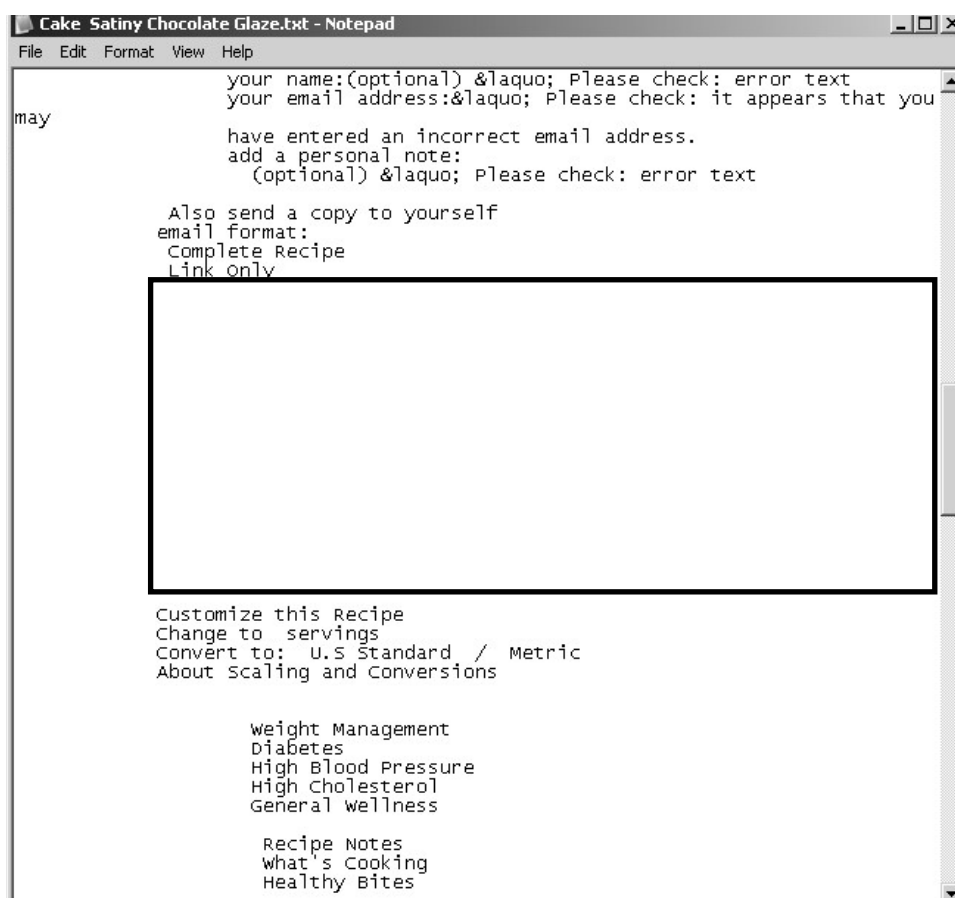


Figure 1 Normal Text Recipe File

The further analysis on the elements of recipe files is shown as below:

- **Title**

Every recipe must have one title. Generally, titles can represent the recipe main distinction. However the structure of the title is irregular and it just can be recognized by a human, not by the computers, when it appears at the random place in the file. Because all recipe files are the Html pages obtained from the web sites and the Html pages use tags to markup the content, it seems that we can search the “title” through the special tag pair “<Title></Title>” in the file. However, as the web pages are saved as *.txt files, all the tag information will lose and the title can’t be recognized by any keywords.

Therefore, in solution 1, the recipe file’s name will be extracted as recipe title. It is reasonable that the recipe files can be renamed to recipe titles when somebody saves the recipe file.

- **Category**

Recipe category can vary a lot, and it can be partitioned in many ways. Recipe category maybe pointed out in some recipe files and maybe not in others.

In this recipe database, the recipe category should be indicated depending on the recipe main ingredients. For example: beef, pork, chicken, seafood etc.

- **Ingredient**

Through browsing many web sites, some general rules and principles for the recipe ingredients description can be obtained. The basic structure of the ingredient's descriptive sentences is:

1 cup water

These three words can be treated as quantity description, unit description, and ingredient description. This means one general ingredient's descriptive sentence consist of three main parts, namely Quantity; Unit; Ingredient.

The reason to subdivide the recipe ingredient description is to decrease the system query time when the user search recipes through the ingredients.

In the following, we analyse these three parts in details.

- *Quantity*

All the Quantity is represented by numbers. It maybe consists of one numerical word (such as 1, 2, 3...) or two numerical words (such as 2 1/4, 5 1/8...). So we can make sure that the Quantity words are always made up of one or two words which include numbers. For example:

1 cup water

1 1/2 cup water

- *Unit*

The Unit word means one kind of measure unit, such as 'cup', 'tablespoon', 'ml' etc. In our real life, the general unit words are finite and standard. So if the system establishes a Unit database in advance which includes all the unit words, then the program can query the Unit database, match and recognize which words are Unit.

Another situation we have to note that sometimes there are one or more adjuncts in front of the unit word. For example:

1 (12 ounce) can corn and 1 glass cup water

System should extract '(12 ounce) can' and 'glass cup' as one entire dataset and put them together into the database.

The last situation also should be considered is that the there are no unit words existing in a piece of ingredients. For example:

2 eggs

The system should return a *null* value when the unit words can't be found out.

- *Ingredient*

Ingredient here means recipe materials description. The materials can be some kinds of seasoning such as a spice, herb, salt, or pepper and some kinds of human food such as beef, eel, or spinach.

Obviously, it is infeasible to subdivide the ingredient description further. The best way to extract materials part is to treat all of the rest words as materials dataset after extracting Quantity, Unit. For example:

2 tablespoons softened butter, hot water

The program should extract out '2', 'tablespoons', 'softened butter, hot water' these three groups of datasets as Quantity, Unit, and Ingredient.

- **Direction**

Most recipe direction part is contained by one whole paragraph. It is not necessary to subdivide the recipe direction though we always treat the recipe direction as one whole part.

2.1.2.2 Database System Requirement Analysis

The aim of creating recipe database system is to query recipe data conveniently and quickly for the general users. This database system also can be managed and controlled through doing some operations by administrators. About this Recipes Database System, one special **import** function should be highlighted.

This Recipe Database System should meet following requirements:

1. This database system should be able to store the recipe data, such as: recipe title, recipe category, recipe ingredient, and recipe direction.
2. Specify different access right for general users and administrators.
3. General users can search recipes by inputting various conditions, for example: title, category, and some ingredients etc.
4. Administrators can do the basic operations such as insert, edit, or delete to update the data to the database.
5. In this recipe database system, an import function will be implemented.
6. As the standard ingredient description line consists of three parts: quantity, unit and the ingredient description, the storage of the ingredient should be detail to those parts level, i.e. store the quantity, unit and the ingredient description respectively.
7. One supplementary administrator record database should exist. It can be used for managing and checking out administrator's information. It should include administrator's name and password.

2.1.2.3 Graphic User Interface Requirement Analysis

Graphic User Interface should meet the following requirements:

1. General users and administrators have different access right. So two different interfaces should be offered, that are: the user interface and the administrator interface.
2. One main interface should exist as recipe database system's entrance. Users have different access right can log in this recipe database respectively from this main interface.
3. The user interface provides general users a query interface. Users can get the corresponding recipe information through inputting the keywords. These keywords can be recipe title, recipe category, and some recipe ingredients.
4. One recipe display window is needed to display those recipes which the user is looking for. Since there is probably more than one recipe were found out, this window should include the recipe detail information: recipe ID, title, category,

ingredient, direction and one recipe name list which can link to other recipes information.

5. Administrator interface provide one database update interface. Administrator can do INSERT, EDIT, DELETE, IMPORT and PERSONAL SETTING operations to the database system.

Insert -- manually insert recipe data like: title, category, ingredient and direction.

Edit -- modify the recipe information

Delete -- delete the recipe from the database

Import -- import recipe information into database from the external recipe files

Personal Setting -- change administrator's password.

Note: In Insert and Edit interfaces, the recipe category should be selected from one category list by users, then one category table should be needed in the database in advance.

2.2 System Analysis and Specification

2.2.1 Import Functionality

The recipe files normally are the Html pages obtained from the web sites. Before performing the import function, the program should remove the Html formatting and tags, and save the recipe files as .txt format. There may be one or more recipes in these files.

We all know that a general, normal recipe basically consists of four main parts: title, category, ingredients and direction. The main task of implementing the import function is to extract those four parts from the recipe files and then put them into the database.

2.2.1.1 Recipe Title Extraction

First of all, the program should extract the recipe title. The recipe title maybe appear at any random places in the file, and its name is irregular. It is infeasible that let the program recognize which string is recipe title from the recipe file.

My idea is:

Extract the recipe file name as recipe name. The precondition for this is that the recipe file was renamed as the recipe title when it was saved into the hard disk. The system can import the recipe title successfully in this way.

2.2.1.2 Paragraph Extraction

Because the external recipe source files are downloaded from web site, probably some redundant, useless information exist in the files. As most of the recipe ingredient description part and recipe direction description part are included in two separate paragraphs, in solution 1, my idea for extracting ingredient and direction is to follow the next two steps:

- First, extract the two useful paragraphs: ingredient paragraph and direction paragraph.
- Then, extract the detail information from these two paragraphs, e.g. extract the quantity, unit and descriptive sentences of the ingredients.

The way I used to recognize the ingredient and direction paragraph can be named: signature way or keyword way.

First, we can assume that the two key words such as 'ingredient' and 'direction' exist in front of the ingredient paragraph and direction paragraph respectively.

About these two special key words, there are many situations to be discussed:

1. The ingredient description part is always indicated by the string 'ingredient', and the direction description part maybe indicated by many strings 'direction' or 'instruction' or 'procedure',
2. Some plural format maybe appears like: ingredients, procedures etc.
3. Sometimes these key words don't appear alone, e.g. appear as part of one sentence, for example:

--- Amount Measure **Ingredient** -- Preparation Method -----

4. Before the right key words shown up, there maybe many such key words exist somewhere in the file, so one judgement should be needed for recognizing which keywords are used for the extraction.

Clearly, the extraction key word “ingredient” is the last occurrence string to appear before the other extraction key words ‘direction’. So after the string ‘ingredient’ appears, the program should continue to search. If another string ‘ingredient’ appears, then the previous ‘ingredient’ will be treated as invalid and then it should be ignored. Until the string ‘direction’ appears, the previous ‘ingredient’ will be treated as the right key word for extraction. At the same time, the string ‘direction’ will be treated as the key word for extraction as well.

After finding out the correct extraction key word, the program should treat the paragraph immediately after it as the extraction paragraph.

Through the analysis above, we know all the ingredient part is displayed in one whole paragraph and most direction description is displayed in one whole paragraph. Then in solution 1, we will assume all the ingredient and direction part are included in one entire paragraph. Likewise, the program can assume the ingredient and direction description terminate when the empty line appears.

After recognizing these two valid paragraphs, another text file (we can call it ‘paragraph file’) will be generated for storing these two useful paragraphs. The new text file will be used for doing the detail extraction conveniently in the future.

2.2.1.3 Recipe Category Extraction

After extracting the recipe ingredient and direction paragraph, the program will extract the recipe category. In this recipe database, the recipe category will be partitioned in the most common way, the category can be ‘Beef’, ‘Pork’, ‘Chicken’, ‘Lamb’, ‘Seafood’ etc according to recipe main materials.

One category table should be created in advance, which includes the following categories: *Beef, Bread, Chicken, Duck, Lamb, Pasta & Pizza, Pork, Seafood, Soup, Sweet & Dessert, Vegetable & Fruit, and Others.*

The way to extract the category is:

First, the program defines the elements in the ‘name’ column of the material table as the query keywords. Then, the program searches these keywords in the title and the direction paragraphs got from the last step – the initial extraction. As the elements have been mapped to some specific category respectively, if any of them is found, the program will set the category which the found keyword belongs to as the recipe category (As long as the program found one keyword existed in the searching area, it will stop the query and set the recipe category). If none of the key words is found, the recipe category will be set to ‘Others’.

The possible materials which belong to one of the recipe category are listed as below:

- **Beef:** beef and stake
- **Bread:** Crust, bread, toast and crumb.
- **Chicken:** Chook, drumstick, turkey, and wing.
- **Lamb:** mutton and lamb.
- **Pork:** Pig, hog, pettitoes, griskin.
- **Seafood:** Fish, shark fin, sturgeon, chub, crucian, pomfret.
- **Sweet & Dessert:** Cookie, cake, biscuit, tortoni, chocolate, choc-ice, nougatine, nicy, ice-cream, and coffee.
- **Vegetable & Fruit:** Salad, celery, cucumber, pawpaw, aubergine, tomato, potato, apple, orange, banana, pear, peach, grape, cherry, and strawberry.

In the database, a material table is needed, which is used to store the name of above common ingredients. When the program looks through the material table and finds out the material in the table is just the same as the one contained in the recipe text file, the recipe category can be specified.

2.2.1.4 Ingredients Extraction

The next step that the program should do is to perform the detail ingredient extraction for the new paragraph file.

Now that all the ingredients descriptions exist in one entire paragraph, so the program consider ingredients description terminate when the space line appears.

According to project's statement, the program should extract recipe ingredients description from file, and then convert them to special dataset, at last put them into the recipe database. Now we know, every recipe ingredients description line consists of Quantities, Unit, and Ingredient parts. So the program should extract these 3 parts from each line respectively.

As mentioned before, we assume that the ingredients description always display in this order: quantity, unit, ingredients description. For example:

1 cup water
500 g butter

Namely, the first part always numerical words, the second part always some words describing measure units; the rest part is material description. The program should recognize these 3 parts and extract them from every line.

First part – Quantity extraction

In the normal situation, the quantity of ingredient description is always written in this format:

1, 1.5, 1/2 or 2 1/2

So I can assume that at most two numerical words (two numbers with some blanks in between each other; the fraction number is considered one word) used for describing quantity. Practically never more than two numbers were used for describing quantity contribution like this:

2 2 1/2 cup water

The extraction procedure for the quantity is: first I can assume the first word is numerical, and the program should continue to check the second word. If the second word is also numerical, the second word should be appended to the first word to generate one string as Quantity. Eventually these two words should be put into the database as one string. If the second word isn't numerical, the program just treats the first numerical words as Quantity and put it into the database.

Second part – Unit extraction

Given the words describing unit are finite and standard, one unit table can be created in recipe database in advance. This unit table should contain all the unit words which may be appearing at any ingredients description (include these words' plural and abbreviations format), such as cup, cups, spoon, tb, g, and ml etc.

One situation should be noted that an adjective might exist before the unit words according to ingredients description rule, for example:

Small cup, middle package etc.

The extraction procedure of the unit is: scan the ingredients line from the left to the right. If a unit words was found, the program should continue to check its previous word. The string before the unit word can be two types: numerical word and descriptive word. For example:

2 cup water

2 small package nuts

If the previous word is numerical word, like *2 cup*, ignore it and just put this individual unit word into the database; if it isn't numerical word, the program will consider this string as one descriptive word like 'small', 'glass' etc, and append it in front of the unit word to generate one string. Eventually, the program will put them together into the database as Unit.

If there isn't any unit words exist, return *null*, namely a *null* value will be put into database as Unit, for example:

2 eggs

Last part – *Ingredients extraction*

The last part is recipe material description; it often consists of some recipe materials and some additional descriptions.

On the surface, the way to recognize recipe materials words can be: first create a materials database, and then extract the material words from the lines. However people will find the way mentioned above is impossible or is not the best to solve this problem after reading my following analysis.

First of all, there are more than 10 thousand kinds of human food. It is impossible and makes no sense to make one statistic on the various human foods for a simple, ordinary recipe database system. Even imagine we have made the perfect statistic for human food and seasonings, please see the following example:

2 cup white sugar

Suppose the program has extracted out ‘2’, ‘cup’ and ‘sugar’ from this sentence. However, how should the program process the rest word ‘white’? let’s see another example:

2 tablespoons softened butter, hot water

Suppose the program can extract out ‘2’, ‘tablespoons’, ‘butter’, ‘water’ from this sentence, and then ‘softened’ and ‘hot’ will be left. Where should they be put into?

Therefore the best way is to treat all the rest strings as one whole string even though it is meaningless to partition the ingredient description into parts. After extracting quantity words and unit words, the program will put all the rest parts of this line into the database as Ingredients.

2.2.1.5 Direction Extraction

This process is very similar to the previous ingredient extraction; the program should extract the direction part from the new paragraph file as well.

According to my experience on the direction structure, generally speaking, most of the directions are written in the consecutive, plain text style (only some minors are formatted into bullets; But these bullets are still context related.). Therefore it makes no sense to separate the direction paragraph into many parts, the recipe direction part can be treated as one whole string. The program should extract this entire paragraph out, and then put them together into the database as Direction.

2.2.1.6 Same Recipe Estimation

Given the possibility of repeat inputting the same recipe, one additional judgment procedure should be needed.

As we all know, people can judge whether the two recipes are the same or not according to the recipe title or direction. Two recipes with the same title maybe have entirely

different directions and two recipes with the same direction maybe have different titles. So we can arrive at the conclusion that the key judgment for two same recipes should depend on the recipe direction. These recipes will be treated as different recipes if their directions are different.

Here, the program will check the directions through the rule of string comparison. The checking algorithm is: first follow the order from the left to the right, from the top to the bottom to check whether each word existed in direction A also exists in direction B. Once a word has been found also existed in direction B, a counter will automatically increase by 1. The program will continue this check until the last word in direction A has been checked. Then, the program will divide the total number of the words in direction A by the number of that counter and get the results AR. After that, the program will do the same operations and calculation on all the words in direction b and check how many percent (BR) of them has also existed in direction A. If both AR and BR exceed 80%, these two recipes will be treated as the same.

2.2.2 Database System

2.2.2.1 Development Environment

In this project, *Microsoft Access 2000* is chosen as the relational database management system. The reason to use the relational database instead of other kinds of databases, such as the XML database, is that the data in the relational database is more structural, and the redundancy of the system can be very low. In addition, the relational database provides much stronger query function and is more extendable. For the XML database (e.g. the Native XML Database³), it stores the whole documents as a unit and may cause some redundancy. Although the XED (XML Enabled Database⁴) can reduce the redundancy by introducing the fine-grained data model, it, in essence, is still based on the relational database.

The data is stored in row and column style in the relational database system. The collection of the rows and columns is called Table, and a group of tables constitutes a database system. In the relational database system, all the data are organized and linked by their relationship. We can present and manipulate the data in the relational database freely.

³ refer to the link: <http://www.xml.com/pub/a/2001/10/31/nativexmlldb.html>

⁴ refer to the link: <http://www.tongyi.net/article/20031012/200310123737.shtml>

2.2.2.2 E-R Model
E-R Diagram

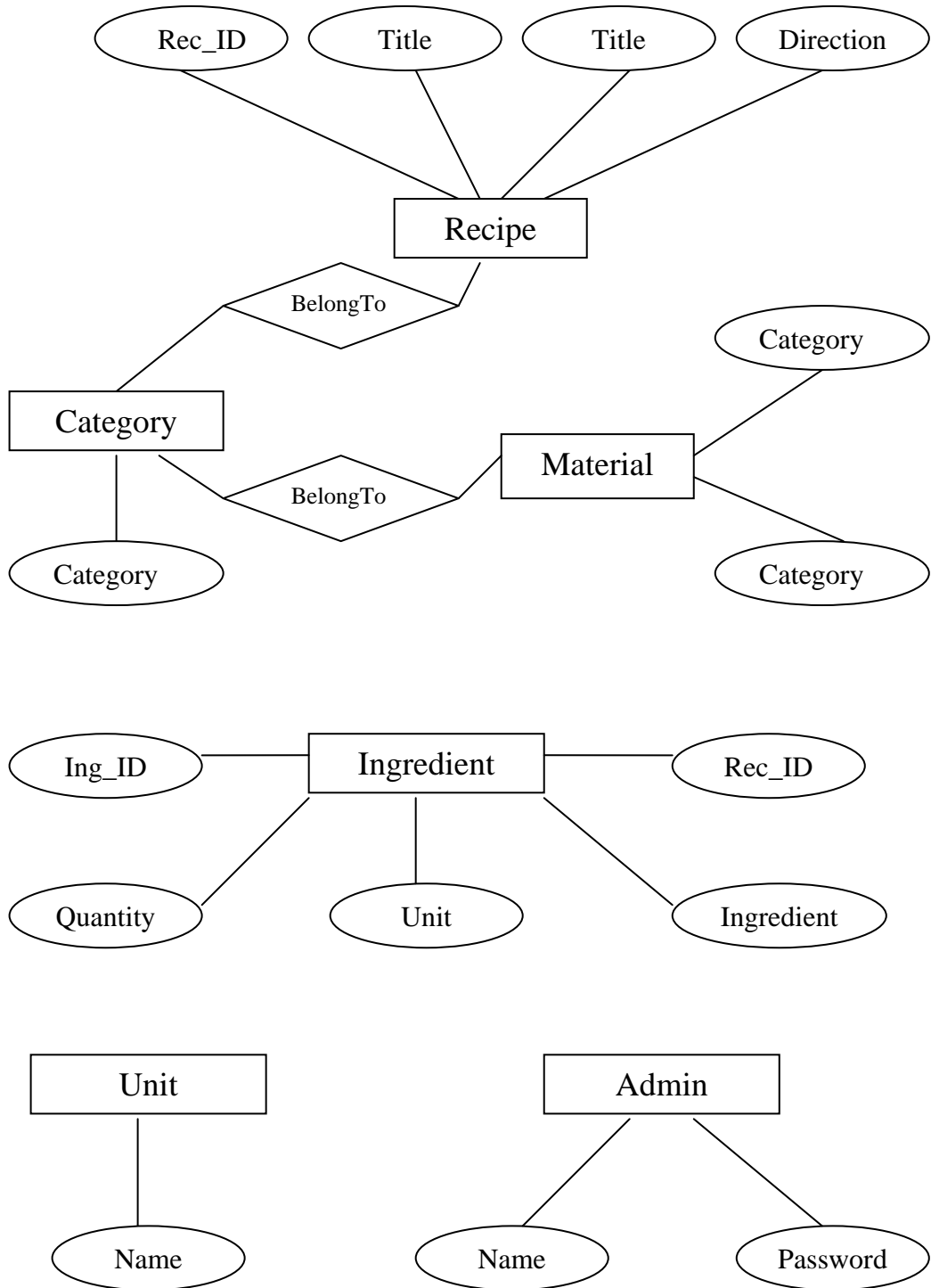


Figure 2 E-R Diagram

2.2.2.3 Use Case Model

Use case modelling from the user view or event flow view; which covers a problem and solutions which involves use case diagrams to use case descriptions.

To successfully apply use case diagrams, the types of elements used should be aware of.

Actor: are used for modelling and representing users' role to a system, which maybe human users or other systems.

Use case: are used for modelling and representing the system behaviours from the user view and it also can be explained to one kind of visible external actions of a system.

Below, the use case model was used for specifying the recipe database system.

Actors:

User-gen -- General user, search recipes from database system

User-adm -- Administrator, manage and manipulate the data in database, which involves modify data, insert data, update database etc.

Use Case Diagram:

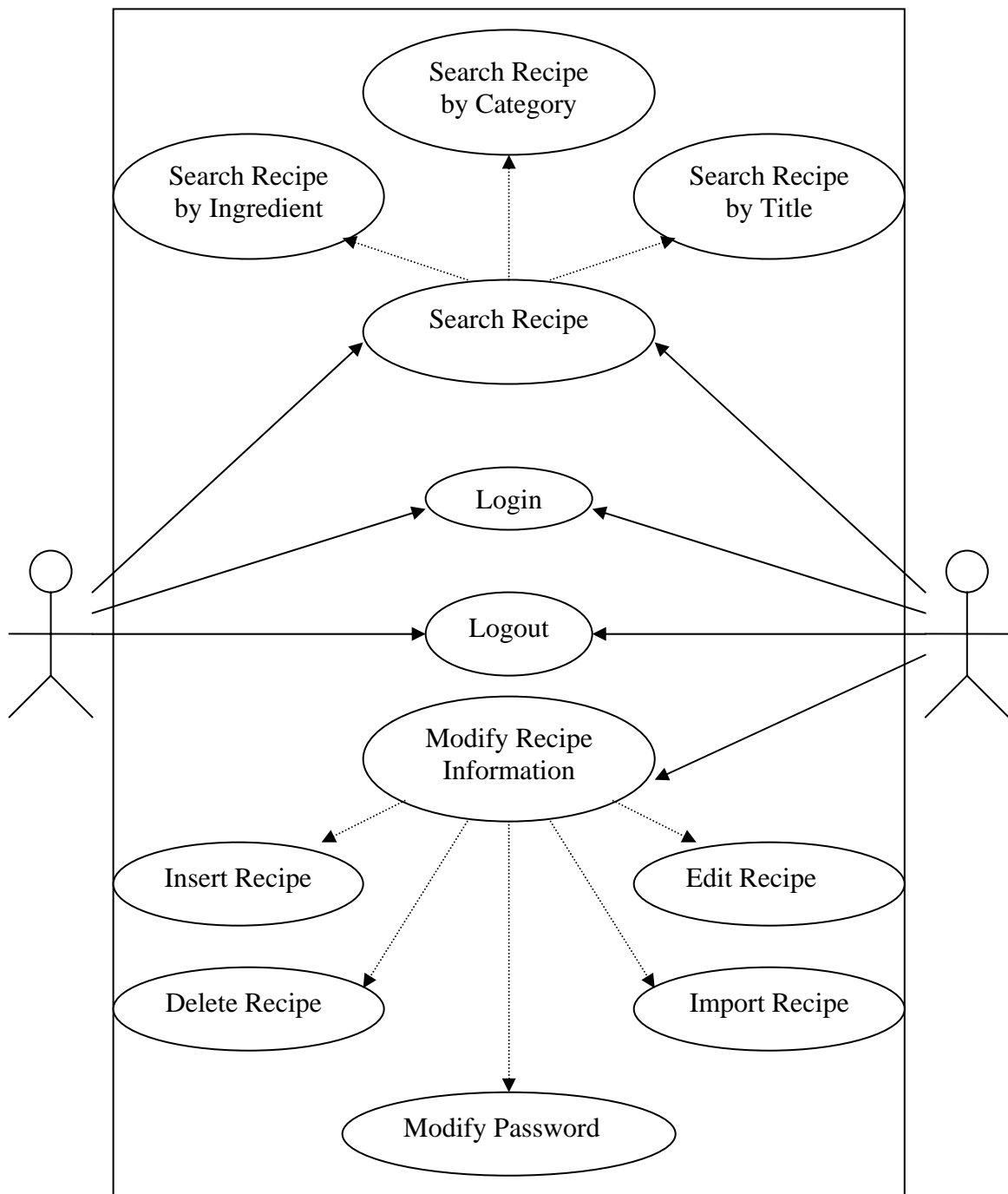


Figure 3 Use Case Diagram

User Cases Description:

- Login of general user

Users can enter the Recipe Query Page without any password.

- Search the Recipes

Users (User-gen and User-adm) search the recipes by inputting keywords.

- Search the Recipes by the Title

Users (User-gen and User-adm) search the recipes by inputting some keywords about recipe title.

- Search the Recipes by the Category

Users (User-gen and User-adm) search the recipes by inputting recipe category.

- Search Recipe by Ingredient

Users (User-gen and User-adm) search recipes by inputting some keywords about recipe ingredients.

- Login of administrator

Users (User-adm) do the login operation to the system. System will detect user's name and password. User will login the system when both the name and password correct and will be rejected when either the name or the password wrong.

- Modify the Recipe Database

Users (User-adm) can modify the recipe database, include insert the new recipes, edit the old recipes, delete the old recipes, and import the new recipes.

- Insert the Recipe

Users (User-adm) insert new recipes data to the database, the recipe information should contain: the title, the category, the ingredient, and the direction.

- Edit the Recipe

Users (User-adm) can modify the information of the old recipes in the database.

- Delete the Recipe

Users (User-adm) can delete the useless recipes from the database.

- Import the Recipe

Users (User-adm) can import the recipes from the external files.

- Modify the Password

Users (User-adm) can modify their passwords freely.

- Logout of general user

Users (User-gen) can do the logout operations when they are out of the system

- Logout of administrator

Users (User-adm) can do the logout operations when they are out of the system.

Use Case Table:

Use Case Table: Login of the general user

Use Case	Login of the general user	
Number	UC01	
Actors	User-gen	
Preconditions	User visit the Recipe Query System entrance page	
Description	Step	Branching Action
	1	The user click on the button 'General User' directly without any password
Success End Condition	The user enter the Recipe Query page	
Failed End Condition		

Table 1 Login of the general user

Use Case Table: Login of the administrator

Use Case	Login of administrator	
Number	UC02	
Actors	User-adm	
Preconditions	The user visits the Recipe Query System page	
Description	Step	Branching Action
	1	The user click on the button 'Administrator', and then a dialog window will appear
		It indicate the user to input the name and the password
	2	The user input the name and the password
Success End Condition	The user enter the administrator page when both the name and the password right	
Failed End Condition	The error message will be returned when either the name or the password wrong	

Table 2 Login of the administrator

Use Case Table: Search the Recipe

Use Case:	Search the Recipe	
Number:	UC03	
Actors:	User-adm, User-gen	
Preconditions:	User has entered the Recipe Query page	
Description	Step	Branching Action
	1	Click on 'Ok' button to search recipes or
		click on 'Back' button to return back the previous page
Success End Condition	The Recipe Display page will be shown out when one or more recipes have been found	
Failed End Condition	The 'No recipes was found!' message will be returned if no recipe matched	

Table 3 Search the recipe

Use Case Table: Search the Recipe by the Title

Use Case	Search the Recipe by the Title	
Number	UC04	
Actors	User-adm, User-gen	
Preconditions	The user has entered the Recipe Query page	
Description	Step	Branching Action
	1	The user input some keywords of the recipe title
	2	Click on the 'Ok' button to search recipes or click on the 'Back' button for returning to the previous page
Success End Condition	The Recipe Display page will be shown out when one or more recipes have been found	
Failed End Condition	The 'No recipes was found!' message will be returned if no recipe matched	

Table 4 Search the Recipe by the Title

Use Case Table: Search the Recipe by the Ingredient

Use Case	Search the Recipe by the Ingredient	
Number	UC05	
Actors	User-adm, User-gen	
Preconditions	The user has entered the Recipe Query page	
Description	Step	Branching Action
	1	The user can input some keywords of the ingredient
	2	Click on the 'Ok' button to search the recipes or click on the 'Back' button for returning to the previous page

Success End Condition	The Recipe Display page will be shown out when one or more recipes have been found
Failed End Condition	The 'No recipes was found!' message will be returned if no recipe matched

Table 5 Search the Recipe by the Ingredient

Use Case Table: Search the Recipe by the Category

Use Case	Use Case: Search the Recipe by the Category	
Number	Number: UC06	
Actors	Actors: User-adm, User-gen	
Preconditions	Preconditions: The user has entered the Recipe Query page	
Description	Step	Branching Action
	1	The user select one kind of recipe category from the category list
	2	Click on the 'Ok' button to search recipes or click on the 'Back' button for returning to the previous page
Success End Condition	The Recipe Display page will be shown out when one or more recipes have been found	
Failed End Condition	The 'No recipes was found!' message will be returned if no recipe matched	

Table 6 Search the Recipe by the Category

Use Case Table: Modify the Recipe Database

Use Case	Modify the Recipe Database	
Number	UC07	
Actors	User-adm	
Preconditions	User has entered the Administrator page	
Description	Step	Branching Action
	1	The user can choose any panels to modify the database, such as the InsertPanel, EditPanel, DeletePanel, ImportPanel, and the PersonSettingPanel.
Success End Condition	The corresponding panel will be lay out.	
Failed End Condition		

Table 7 Modify the Recipe Database

Use Case Table: Insert the Recipe

Use Case	Insert the Recipe	
Number	UC08	
Actors	User-adm	
Preconditions	The user has chosen the InsertPanel	
Description	Step	Branching Action
	1	The user should completely fill in the recipe information, including the title, ingredient, direction and the category
	2	Click on the 'Save' button to save the new recipe in the database or Click on the 'Clear' button to clear the panel
Success End Condition	A new recipe is saved in database when the new recipe information is valid	
Failed End Condition	An error message should be returned when the new recipe is not completely filled in or when the new recipe information is invalid Another error message should be returned when the new recipe is filled in the wrong format.	

Table 8 Insert the Recipe

Use Case Table: Edit the Recipe

Use Case	Edit the Recipe	
Number	UC09	
Actors	User-adm	
Preconditions	The user has chosen the EditPanel	
Description	Step	Branching Action
	1	The user chooses the recipe ID Then the corresponding recipe information is displayed on the panel
	2	The user can modify the recipe information such as: the title, ingredient, direction, category except the ID.
	3	Click on the 'Update' button to update the recipe data or Click on the 'Clear' to initialize the panel
Success End Condition	The old recipe is updated if the new data is valid	
Failed End Condition	An error message will be returned if the new recipe information is invalid, such as wrong format.	

Table 9 Edit the Recipe

Use Case Table: Delete the Recipe

Use Case	Delete the Recipe	
Number	UC10	
Actors	User-adm	
Preconditions	The user has chosen the DeletePanel	
Description	Step	Branching Action
	1	The user chooses the recipe ID
	2	the recipe information is displayed on the panel And all the data fields displayed are non-editable Click on the 'Delete' button to delete the recipe from the database or Click on the 'Clear' to initialize the panel
Success End Condition	The old recipe will be deleted from the database	
Failed End Condition		

Table 10 Delete the Recipe

Use Case Table: Import the Recipe

Use Case	Import the Recipe	
Number	UC11	
Actors	User-adm	
Preconditions	The user has chosen the ImportPanel	
Description	Step	Branching Action
	1	The user chooses one recipe file from the local disk
	2	Click on the 'Import' to import this new recipe into the database or Click on the 'Cancel' to initialize this panel
Success End Condition	The new recipe that has been chosen is imported if the recipe file is valid.	
Failed End Condition	An error message will be returned if the recipe file is invalid such as: there isn't any recipe information existing in the recipe file or the recipe information is incomplete	

Table 11 Import the Recipe

Use Case Table: Modify the Password

Use Case	Modify the Password	
Number	UC12	
Actors	User-adm	
Preconditions	The user has chosen the PersonSettingPanel	
Description	Step	Branching Action
	1	The user name has been displayed and it is non-editable
	2	Input the original password and input the new password twice for ensuring
	3	Click on the 'Modify' to change the password or Click on the 'Clear' to initialize this panel
Success End Condition	The password will be changed if all the input data is correct	
Failed End Condition		

Table 12 Modify the Password

Use Case Table: Logout of the general user

Use Case	Use Case: Logout of general user	
Number	Number: UC13	
Actors	Actors: User-gen	
Preconditions	Preconditions: The user has entered the Recipe Query page	
Description	Step	
	1	The user click on the button 'Logout' A dialog window will appear It indicates the user to confirm the logout operation
Success End Condition	The user logout the system, And the GUI returns back to initial page when 'Yes' is selected	
Failed End Condition	The page will be remained when the 'No' is selected	

Table 13 Modify the Password

Use Case Table: Logout of the administrator

Use Case	Logout of the administrator
Number	UC14
Actors	User-adm
Preconditions	The user has entered the Administrator page

Description	Step	Branching Action
	1	The user click on the button 'Logout' A dialog window will appear It indicates the user to confirm the logout operation
Success End Condition		The user logout the system, And the GUI returns back to initial page when 'Yes' is selected
Failed End Condition		This page will be remained when the 'No' is selected

2.2.3 Graphics User Interface

The graphics user interface's component specification are following:

* Entrance Interface

Contains two Buttons. One is the entry button for general users, and the other is for administrators.

General users can enter the Recipe Query page without any password, while the administrator has to input the correct user-id and password for entering enter the Administrator page.

* Users Interface

Contains two TextFields and one ComboBox. One TextField is for inputting recipe title and the other is for inputting recipe ingredient; The ComboBox is for displaying recipe category list, which allows the users select the category manually.

* Administrator Interface

Contains one TabbedPane on which there are InsertPanel, EditPanel, DeletePanel, ImportPanel and PersonSettingPanel.

- InsertPanel

Contains a TextField for inputting recipe title; A ComboBox display recipe category users can select; A Table for inputting ingredient elements; And a TextArea for inputting recipe direction.

- EditPanel

Contains a ComboBox display recipe ID which users can select; a TextField for displaying recipe title which also can be used for modifying recipe title; and a Table for displaying recipe ingredient which also can be used for editing recipe ingredient.

- DeletePanel

Contains a ComboBox display recipe ID can be selected; a non-editable TextField for displaying recipe title; a Table for displaying recipe ingredient; and a TextFiled for displaying recipe direction

- ImportPanel

Contains a Button for browsing the recipe file will be imported; And a ComBox displaying recipe category can be selected.

- PersonSetting interface

Contains four TextFields, one is non-editable for displaying the administrator's name, one for inputting original password, one for inputting new password, and the last one for re-entering new password for making sure the new password correct or not.

* Recipe Display Interface

Two types Recipe Display Interface should be offered

- One for displaying the recipe which has been imported into the database from external recipe file

Contains two TextAreas for displaying recipe ingredient and recipe direction; three TextLables for displaying recipe title, recipe ID, recipe category.

- The other Recipe Display Interface for displaying those recipes which was searched by general users

Contains two TextAreas for displaying recipe ingredient and recipe direction; three TextLables for displaying recipe title, recipe ID, recipe category; And a List for displaying those recipes' name.

2.3 System Implementation

2.3.1 System Architecture

This recipe database system is based on the Model-View-Controller architecture. The Model-View-Controller (MVC) is a powerful commonly used architecture for GUIs. The MVC paradigm is a way of separating an application into three parts: the model, the view, and the controller. MVC was originally developed to map the traditional input, processing, output roles into the GUI realm:

Input	-->	Processing	-->	Output
Controller	-->	Model	-->	View

In the MVC paradigm the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of object, each specialized for its task.

- **View** manages the graphical and/or textual output to the display that is assigned to its application.
- **Controller** interprets the mouse and keyboard inputs from the user, maps these user actions into commands that are sent to the model and/or view to effect the appropriate change.
- **Model** manages the behavior and one or more data elements of the application, responds to requirement for information about its situation and responds to instructions to change state.

The basic Model-View-Controller can be illustrated by the following picture:

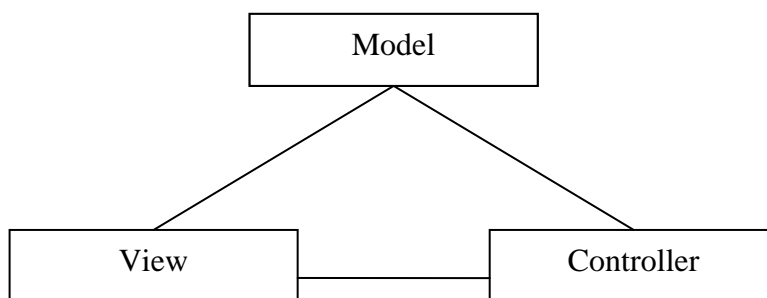


Figure 4The MVC model

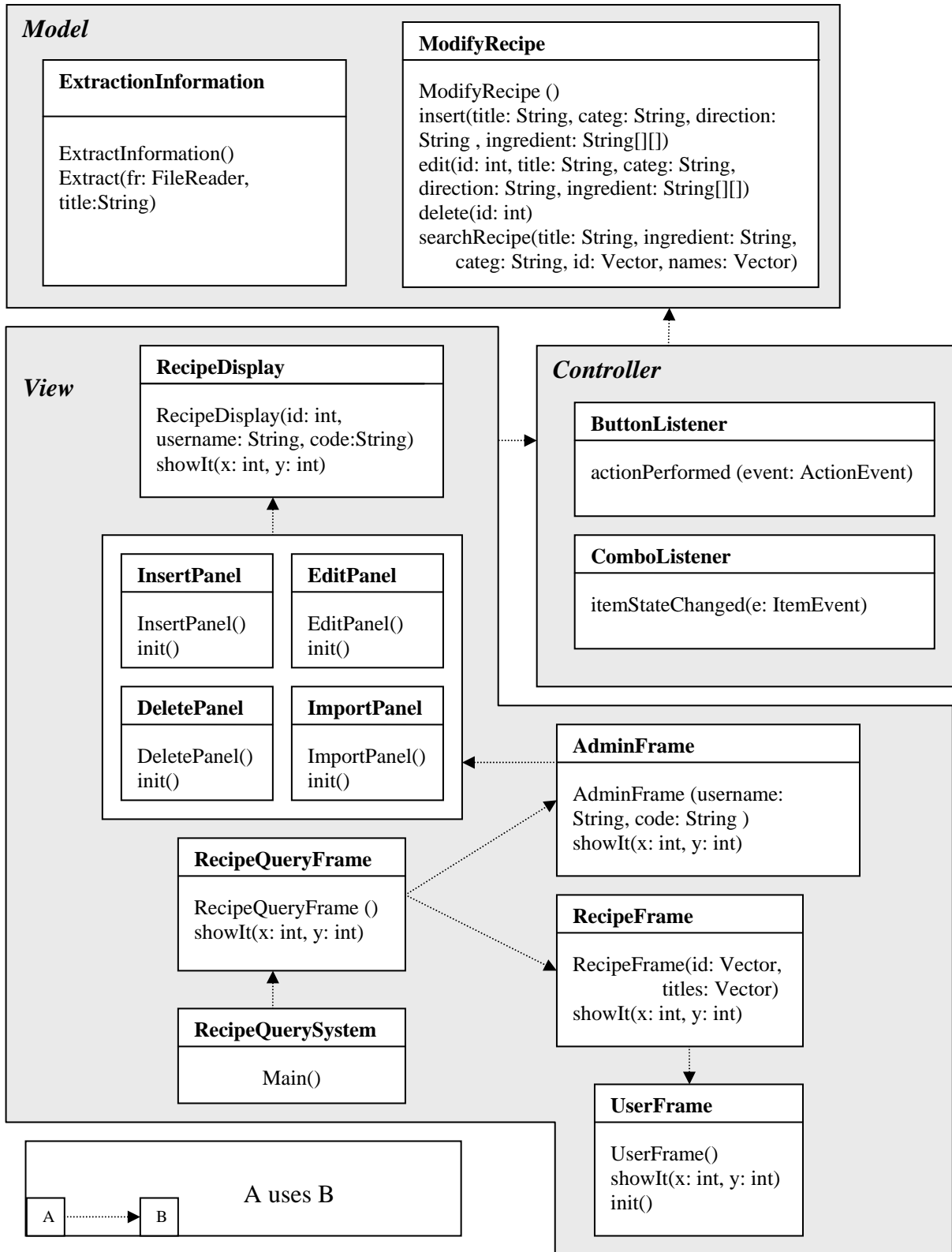


Figure 5 UML Class Diagram

2.3.2 Microsoft Access Databases Design and Implementation

The procedure of database design and implementation was illustrated by following picture:

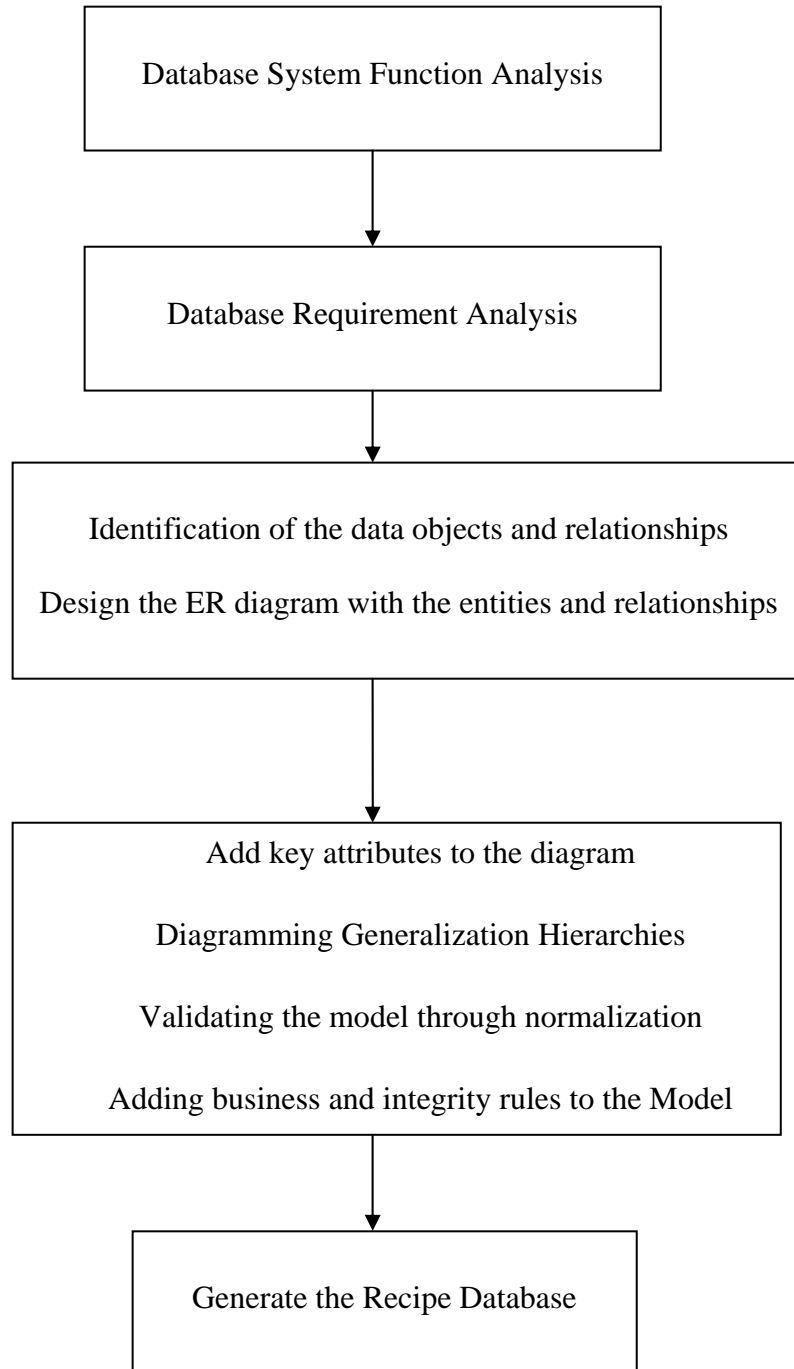


Figure 6 the procedure of the design and implementation of the database

The table is the central element in Access, which consists of data records that contain all the data information. Each table is composed by many fields which have different data types. Each row in the table is a record of the database. The procedure of implementing the database is to convert the E-R diagram to the tables. Every entity can be converted to one table and their attributes can be converted to the fields. Refer to my E-R diagram above; six tables are built in my database.

1. The Recipe Table

	Rec_ID	Title	Category	Direction
▶	(AutoNumber)			

Table 14 the Recipe Table

In the recipe table, the recipe ID was set as the primary key, because in the recipe database, some different recipes might have the same title names and only the recipe ID is unique. The program offered an automatic import function and the 'Rec_ID' field's data type was set *AutoNumber* which means the recipe ID will be generated automatically when a new recipe was imported. The data types of the field 'Title' and 'Category' were set *Text* and the data type of the field 'Direction' was set *Memo*.

2. The Ingredient Table

	Ing_ID	Rec_ID	Quantity	Unit	Ingredient
▶	(AutoNumber)				

Table 15 The Ingredient Table

In the ingredient table, the ingredient's ID was set as the primary key. The field 'Rec_ID' here is matched to the 'Rec_ID' field in the recipe table. The reason for the 'Ing_ID' field's data type was set *AutoNumber* is the same as the one for 'Rec_ID' field in recipe table. When a new recipe was imported into the database, the ingredient description items of the recipe its were filled into the ingredient table automatically.

3. The Category Table

Category	
▶	+ Beef
	+ Bread
	+ Chichen
	+ Duck
	+ Lamb
	+ Others
	+ Pasta & Pizza
	+ Pork
	+ Seafood
	+ Sweets & Desserts
	+ Vegetable & Fruit
*	

Table 16 The Category Table

Refer to the requirement specification; a category table is needed in my database. The 'Category' fields in the recipe table and in the material table as below are both matched to the 'Category' field in this table.

4. The Material Table

Name	Category
▶	

Table 17 The Material Table

The 'Name' field was set as the primary key and the 'Category' field is match to the 'Category' field in the category table

5. The Unit Table

	Name
▶	cl
	can
	cans
	cl
	cup
	cups
	g
	kg
	l
	..

Table 18 The Unit Table

The words that represent the units should be filled into the unit table as many as possible in advance. The program will extract the unit word from the recipe file according to the values in this table.

6. The Admin Table

	Name	Password
▶		

Table 19 The Admin Table

The admin table was used to store the administrators' records. The 'Name' was set as the primary key.

2.3.2 Model Implementation

There are two classes contained in the system model part, they are: *ExtractionInformation* and *ModifyRecipe*.

The *Extract(FileReader fr, String title)* method in the *ExtractionInformation* class will be called when the import operation was performed. The *ModifyRecipe* class contains insert, edit, delete, researchRecipe methods.

SQL and JDBC techniques are primarily used for implementing these models. In the following sections, I will give a brief introduction about them.

- SQL Introduction⁵

SQL (Structure Query Language) is a kind of ANSI (American National Standards Institute) standard computer languages for accessing and manipulating database systems; now it is widely used as one kind of relational databases query languages. SQL consists of four functions such as: query, manipulate, definition and control. It integrates Data Manipulation Language (DML) and Data Definition Language (DDL).

Data Manipulation Language (DML) is used to query and manipulate the database. The basic commands are shown below:

SELECT – is used to select data from a table.

Syntax --

```
SELECT column_name(s)
FROM table_name
```

Table 20 Select

UPDATE –is used to modify the data in a table.

Syntax --

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Table 21 UPDATE

DELETE –is used to delete rows in a table.

Syntax –

```
DELETE FROM table_name
```

⁵ refer to the link: <http://www.w3schools.com/sql/default.asp>

```
WHERE column_name = some_value
```

Table 22 Delete

INSERT INTO –is used to insert new rows into a table.

Syntax –

```
INSERT INTO table_name  
VALUES (value1, value2,....)
```

Table 23 Insert Into

The Data Definition Language (DDL) is used to create and delete the database. We can also define indexes (keys), specify links between tables, and impose constraints between database tables. The commands are shown below:

CREATE – to create a database: create a table and create index

DROP – drop index, table and database:

ALTER TABLE –is used to add or drop columns in an existing table.

In my project, Data Manipulation Language (DML) was used to connect the SQL engine by using JDBC.

- JDBC Introduction⁶

JDBC (Java Data Base Connectivity) is a set of Java APIs that provide Java programs with a way to connect to and use relational databases. The JDBC API makes it easy to send SQL statements to relational database systems. The combination of Java's JDBC and standard SQL provides a simple and powerful database solution.

To make JDBC work, the first thing is to install Java and JDBC. The JDBC has been included in the JDK.

The Next step is to install a JDBC driver.

The SUN Company has defined four types of JDBC drivers. They are:

Type 1: JDBC-ODBC Bridge

Type 2: Native-API/partly Java driver

Type 3: Net-protocol/all-Java driver

Type 4: Native-protocol/all-Java driver

⁶ refer to the link: <http://www.w3schools.com/sql/default.asp>

In my project, the type1 (JDBC-ODBC) was selected as my JDBC driver, because the Access adopts ODBC as the interface of its database.

- Establish a connection

Establishing a connection with the DBMS involves two steps: (1) loading the driver and (2) establishing the connection.

1. Loading the Drivers

The code used for loading the driver is:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Table 24 Load the Drivers

2. Establishing the Connection

After loading a driver, it is ready for establishing the connection with the DBMS.

```
String url="jdbc:odbc:driver={Microsoft Access  
Driver (*.mdb)};DBQ=Recipes.mdb";  
con = DriverManager.getConnection(url, "mylogin",  
"mypassword");
```

Table 25 Establish the Connection

Con is an open connection and I used it to retrieve values from ResultSets (The dataset that contains the querying results).

A `Statement` object, which includes the SQL statement, is the object that the program sends to the DBMS. The working procedure of it is:

First, create a `Statement` object

```
Statement stmt = con.createStatement();
```

Then, assign the target SQL statement and invoke the appropriate execution method. For instances, for a `SELECT` statement, the method to use is `executeQuery`; For statements that create or modify tables, the method to use is `executeUpdate`.

```
String update = "UPDATE table_name SET column_name = new_value  
WHERE column_name = some_value";  
stmt.executeUpdate(update);
```

Table 26 Execute the Update

```
String delete = "DELETE FROM table_name  
                WHERE column_name = some_value";  
stmt.executeUpdate(delete);
```

Table 27 Execute the Delete

```
String insert = "INSERT INTO table_name  
                VALUES (value1, value2,...)";  
stmt.executeUpdate(insert);
```

Table 28 Execute the Insert

```
String query = " SELECT column_name(s) FROM table_name";  
ResultSet rs = stmt.executeQuery( query );
```

Table 29 Execute the Query

Every time the invoked method requires the return results from the remote Database, JDBC returns the results in a `ResultSet` object. The variable `rs`, the instance of `ResultSet`, contains the rows of the columns. In order to access the values of it, the program should address each row and retrieve the values according to their data types. The method `next` is used to move the pointer to the data row. The first time to call the `next` method of the `rs`, the pointer is moved to the first row. And the later successive invocations of that method move the pointer one row down at a time, from the top to the bottom.

To retrieve the values in a `ResultSet` object, the method `getString` should be called; it is possible to retrieve all the basic SQL types with it.

- Key Functions Implementation

My recipe database offers the main functions: insert recipe, edit recipe, delete recipe, research recipe and import recipe automatically. The implementations of them are listed below:

1. Insert the Recipe (insert values into the recipe table and the ingredient table)

When the operator inserts the recipe information manually, the recipe title, category, direction values will be filled into the recipe table and the recipe ingredient descriptions will be filled into ingredient table. The procedure is: first insert the recipe title, category and direction into the recipe table and get a recipe ID (assigned by the program automatically). After the recipe ID is obtained, insert the recipe ID and the recipe ingredients into the ingredient table. The ingredients are split into three parts and stored, namely quantity, unit, and ingredient. The implementation code is shown below:

Class ModifyRecipe

```
public boolean insert (String title, String categ, String direction, String[][] ingredient )
{
    try{
        stmt = con.createStatement();
        String insert1 = "INSERT INTO Recipe (Title, Category, Direction) VALUES ("
            + title +", "+categ+", "+direction+" ) ";
        stmt.executeUpdate(insert1);
        String IDquery = ("SELECT Rec_ID FROM Recipe")
        rs1 = stmt.executeQuery(IDquery);
        int Rec_ID;
        rs1.next();
        do{
            String Rec_ID = rs1.getString("Rec_ID");
            Rec_ID = Integer.parseInt(Rec_ID1);
        } while(rs1.next());
        String s1 = "", s2 = "", s3 = "";
        for(int i = 0; i < 20; i++)
        {
            s1 = ingredient[i][0];
            s2 = ingredient[i][1];
            s3 = ingredient[i][2];
        }

        String insert2 = "INSERT INTO Ingredient ( Rec_ID, Quantity, Unit, Ingredient)"
            + "VALUES (" + Rec_ID +", " + s1 +", " + s2 +", " + s3 +")";
        stmt.executeUpdate(insert2);
        return true;
    }
    catch (Exception e){return false; }
}
```

Figure 7 the Code for the method ‘Insert the Recipe’

2. Edit the Recipe (Modify the values in the recipe table and the ingredient table)

The users could modify the values in the recipe table and the ingredient table through selecting the recipe ID. According to the recipe ID selected by the user, the target recipe record in the recipe table can be found. And then the field values such as title, category and direction could be replaced by the new values directly. In the ingredient table, the program will delete all the ingredient description records matched to the target recipe ID first, and then insert the new values into the table. The implementation code is shown below:

Class ModifyRecipe

```
public boolean edit (int id,
String title, String categ, String direction, String[][] ingredient )
{
    try {
        stmt = con.createStatement();
        String update = "UPDATE Recipe SET Title = '"+title+"', "+
            "Category='"+categ+"', "+ "Direction = "+
            "'"+direction+"'"+"WHERE Rec_ID = "+id;
        stmt.executeUpdate(update);
        String s1 = "", s2 = "", s3 = "";
        String delete = "DELETE FROM Ingredient "+
            "WHERE Rec_ID = "+id;
        stmt.executeUpdate(delete);
        for(int i = 0; i < 20; i++)
        {
            s1 = ingredient[i][0];
            s2 = ingredient[i][1];
            s3 = ingredient[i][2];
        }
        String insert = "INSERT INTO Ingredient ( Rec_ID, Quantity, Unit,
            Ingredient)"+"VALUES ("+ id+", '"+s1 +"', '"+ s2 +
            "', '"+ s3 +"'");
        stmt.executeUpdate(insert);
        return true;
    }
    catch (Exception e){return false; }
}
```

Figure 8 The code for the method 'edit the recipe'

3. Delete the Recipe (delete the values in the recipe table and the ingredient table)

According to the recipe ID selected by the user, the corresponding records in the recipe table and in the ingredient table can be found and deleted easily by using SQL statement:

```
"DELETE FROM Recipe WHERE Rec_ID = "+id;
"DELETE FROM Ingredient WHERE Rec_ID = "+id;
```



```

Class ModifyRecipe

public void delete (int id)
{
    try{
        stmt = con.createStatement();
        String delete1 = "DELETE FROM Recipe "+
            "WHERE Rec_ID = "+id;
        stmt.executeUpdate(delete1);
        String delete2 = "DELETE FROM Ingredient "+
            "WHERE Rec_ID = "+id;
        stmt.executeUpdate(delete2);
    }
    catch (Exception e) {}
}

```

Figure 9 The code for the method 'delete the recipe'

4. Search Recipes

As mentioned above, the users can search for a recipe by their interests such as the title, the ingredient and the category of the recipe. In my query algorithm, firstly the program checks if the recipe category is specified by the users. If the recipe category was specified, those recipes matched by this specified category can be found. Then the program checks which recipes contain the keywords input by the users in the title and ingredients. At last the program put those matched recipe IDs and titles into two Vector objects: "id" and "names", and then invoke the RecipeFrame class to display them.

Here, an 'isElement(a, b)' method was used to check if String b contains String a. In this program, 'isElement(a,b)' was used to check if the recipe title and ingredient contain the keywords input by the users.

Class ModifyRecipe

```
public void searchRecipe(String title, String ingredient, String categ,
Vector id, Vector names)
{
    try{
        if (categ != null)
        {
            String get_ID = ("SELECT Rec_ID, Title, Direction FROM Recipe "+
                "WHERE Category =" + categ + "");
            stmt = con.createStatement();
            rs = stmt.executeQuery(get_ID);
            while( rs.next())
            {
                String rtitle= rs.getString("Title");
                String ring = rs.getString("Direction");
                if ( isElement(title, rtitle)&&isElement(ingredient, ring))
                {
                    String ID = rs.getString("Rec_ID");
                    id.add(ID);
                    names.add(rtitle);
                }
            }
        }
        else {
            String get_ID = ("SELECT Rec_ID, Title, Direction FROM Recipe ");
            stmt = con.createStatement();
            rs = stmt.executeQuery(get_ID);
            while( rs.next())
            {
                String rtitle = rs.getString("Title");
                String ring = rs.getString("Direction");
                if ( isElement(title, rtitle) && isElement(ingredient, ring))
                {
                    String ID = rs.getString("Rec_ID");
                    id.add(ID);
                    names.add(rtitle)}
                }
            }
        }
    }
    catch (Exception e) {}
}
```

Figure 10 The code for the method 'search the recipe'

5. Import External Recipe

Referring to the requirement specification as mentioned before, the algorithm for implementing the import function is quite complicated. Therefore it is better to explain the algorithm by some flow charts.

Chart [Flow01] explains how the primary model for the import function is implemented. This primary model is implemented in the method: `Extract ()`, which returns a value of the `int` type. The implementations of other models are illustrated in detail by the following flow charts.

Chart [Flow02] explains how the model used for checking if the external recipe file valid is implemented. This model is implemented in the method: `ValidRecipe ()`, which returns a value of the `boolean` type.

Chart [Flow03] explains how the model for extracting the useful paragraphs from the file is implemented. This model is implemented in the method: `ExtractParagraph ()`, which doesn't return any value but generate a file which holds the extracted paragraphs.

Chart [Flow04] explains how the model for extracting the recipe direction part is implemented. This model is implemented in the method: `ExtractDirection ()`, which returns a value of the `String` type.

Chart [Flow05] explains how the model for extracting the recipe category is implemented. This model is implemented in the method: `ExtractCategory ()`, which returns a value of `String` type.

Chart [Flow06] explain how the model for extracting the recipe ingredient part is implemented. This model was implemented in the method: `ExtractIngredient ()`, which doesn't return any value but put the recipe ingredient into the database directly.

First, two private fields are defined in the `ExtractInformation` class:

```
private final static int RecipeExist = -1;  
private final static int InvalidRecipe = -2;
```

The Flow charts are shown below:

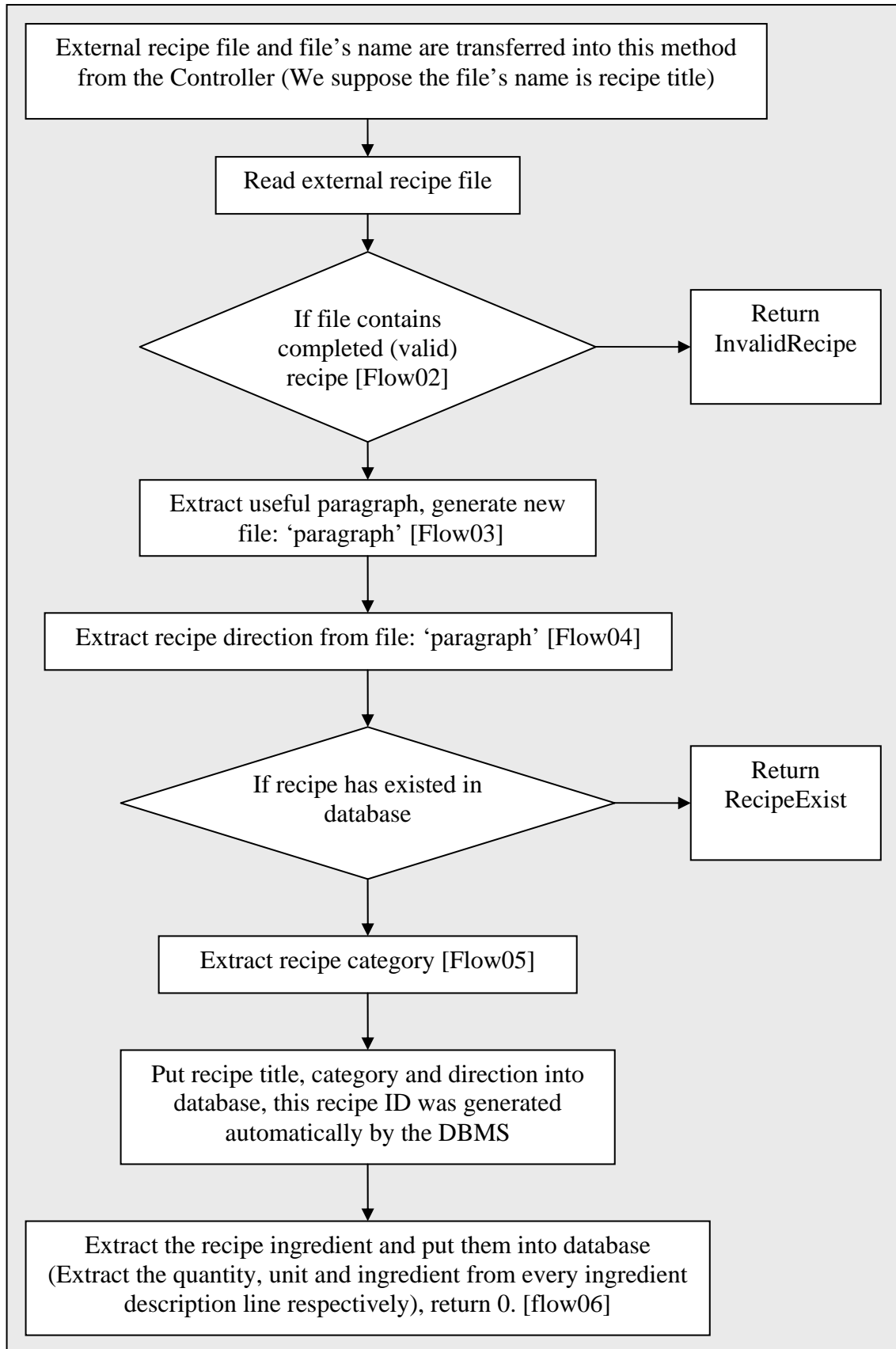


Figure 11 Flow01

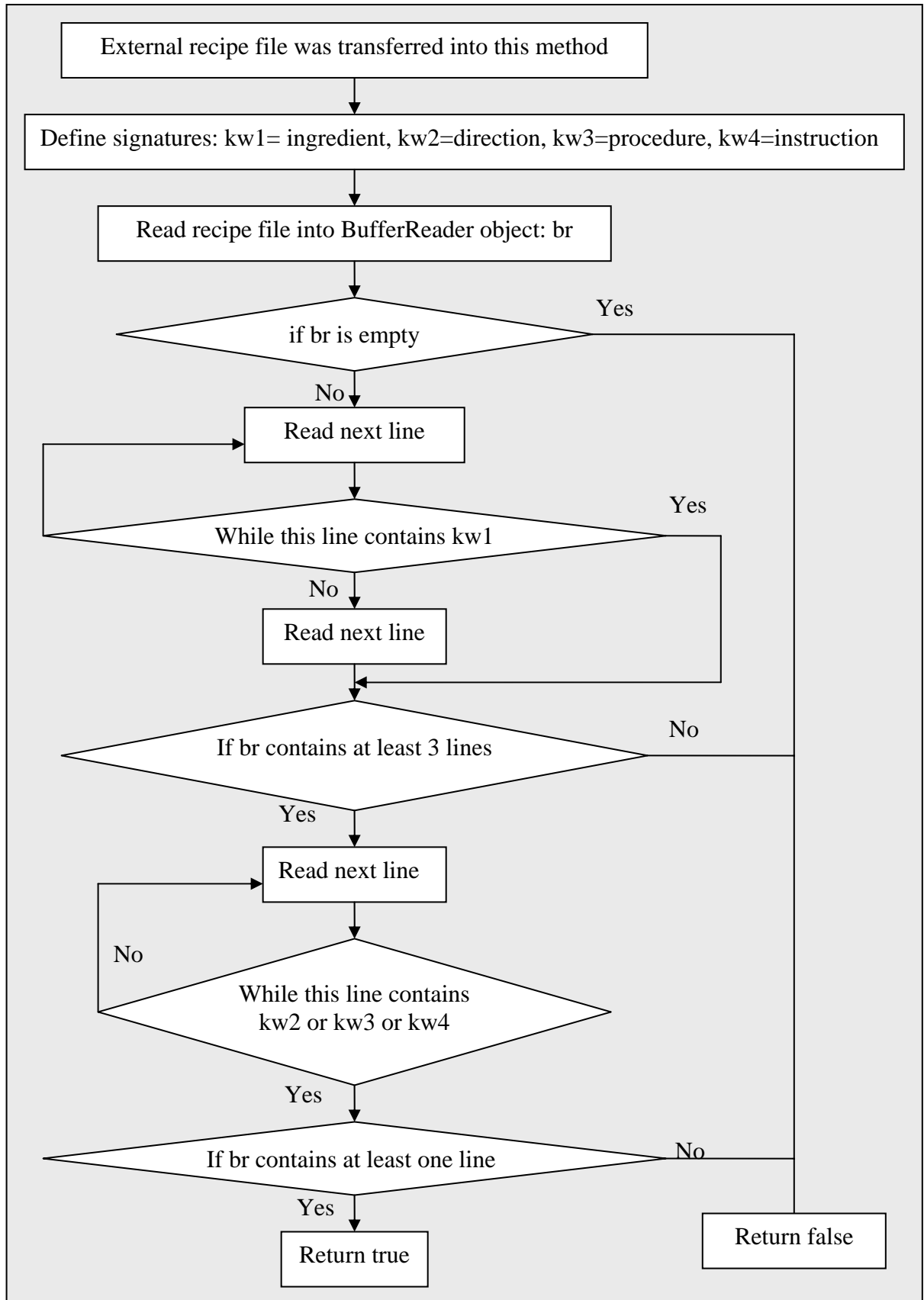


Figure 12 Flow02

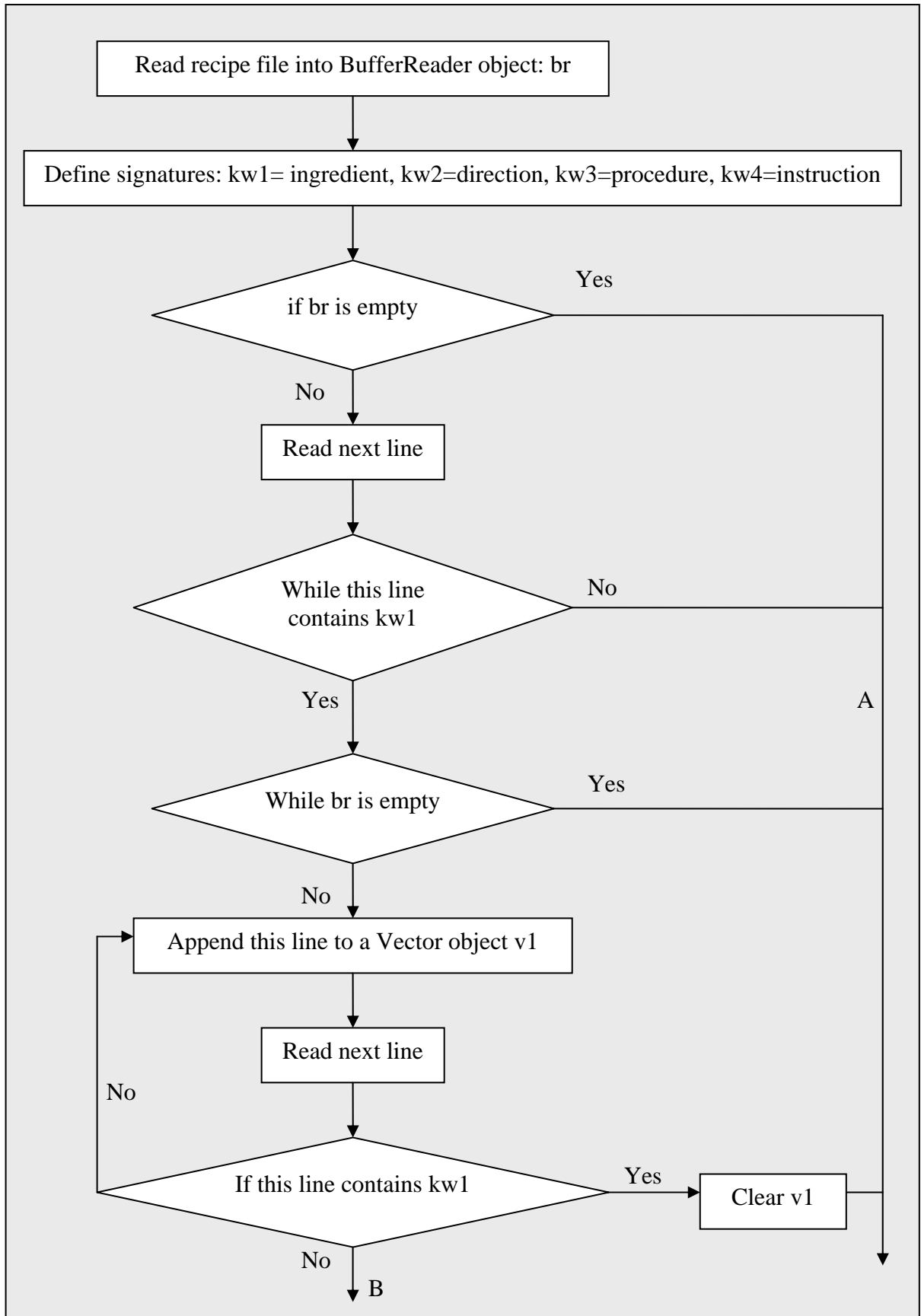


Figure 13 Flow03-a

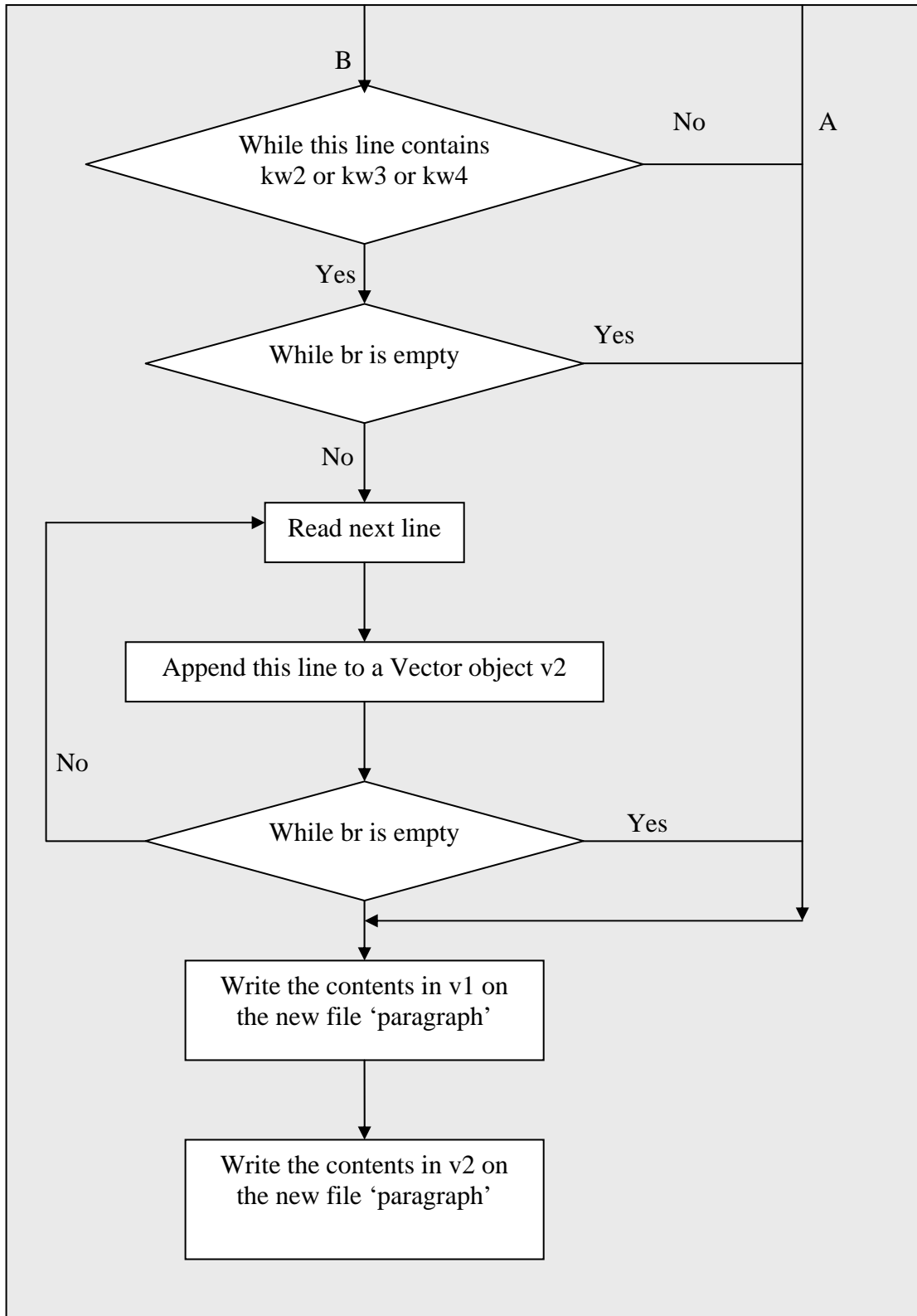


Figure 14 Flow03-b

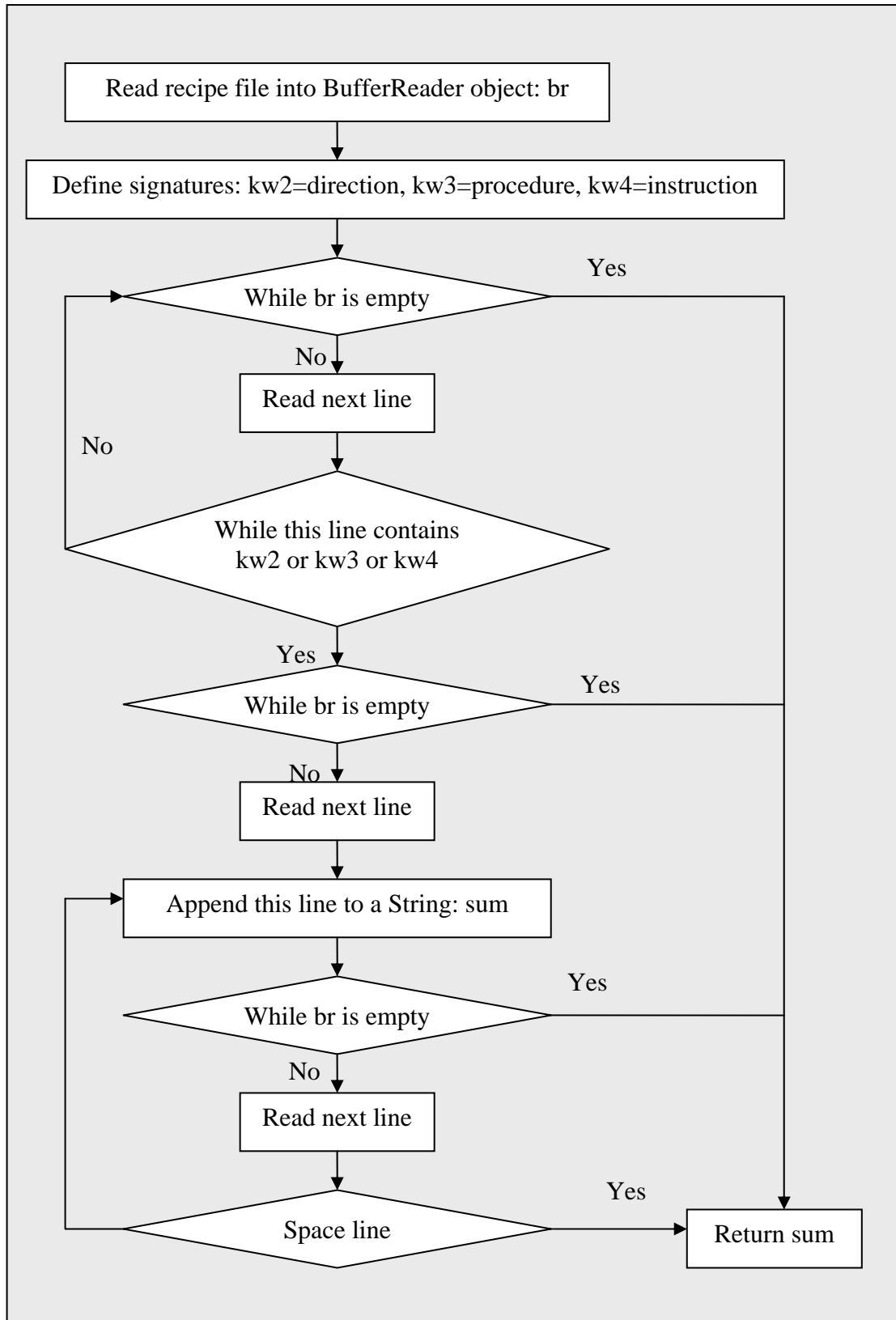


Figure 15 Flow04

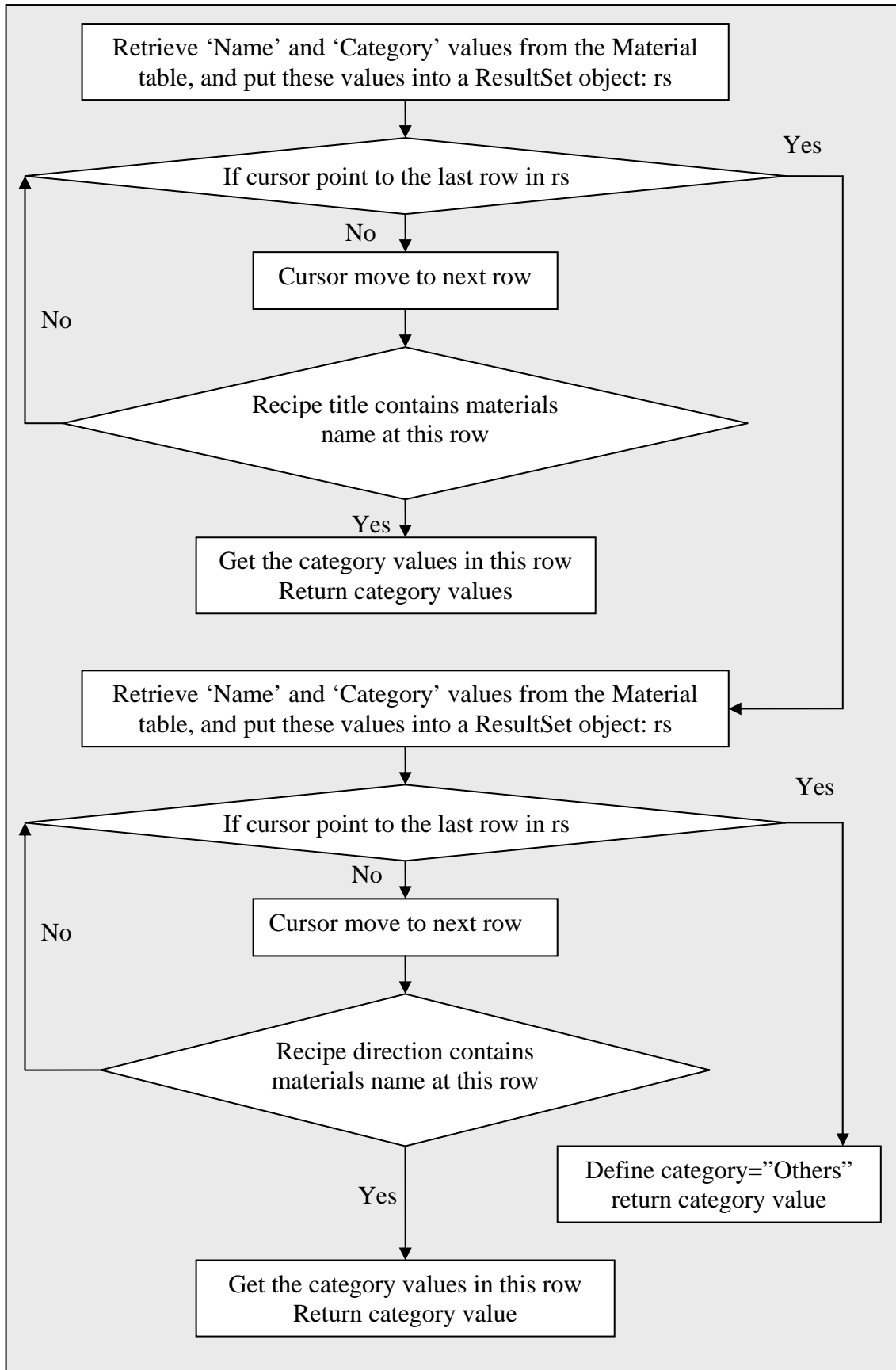


Figure 16 Flow05

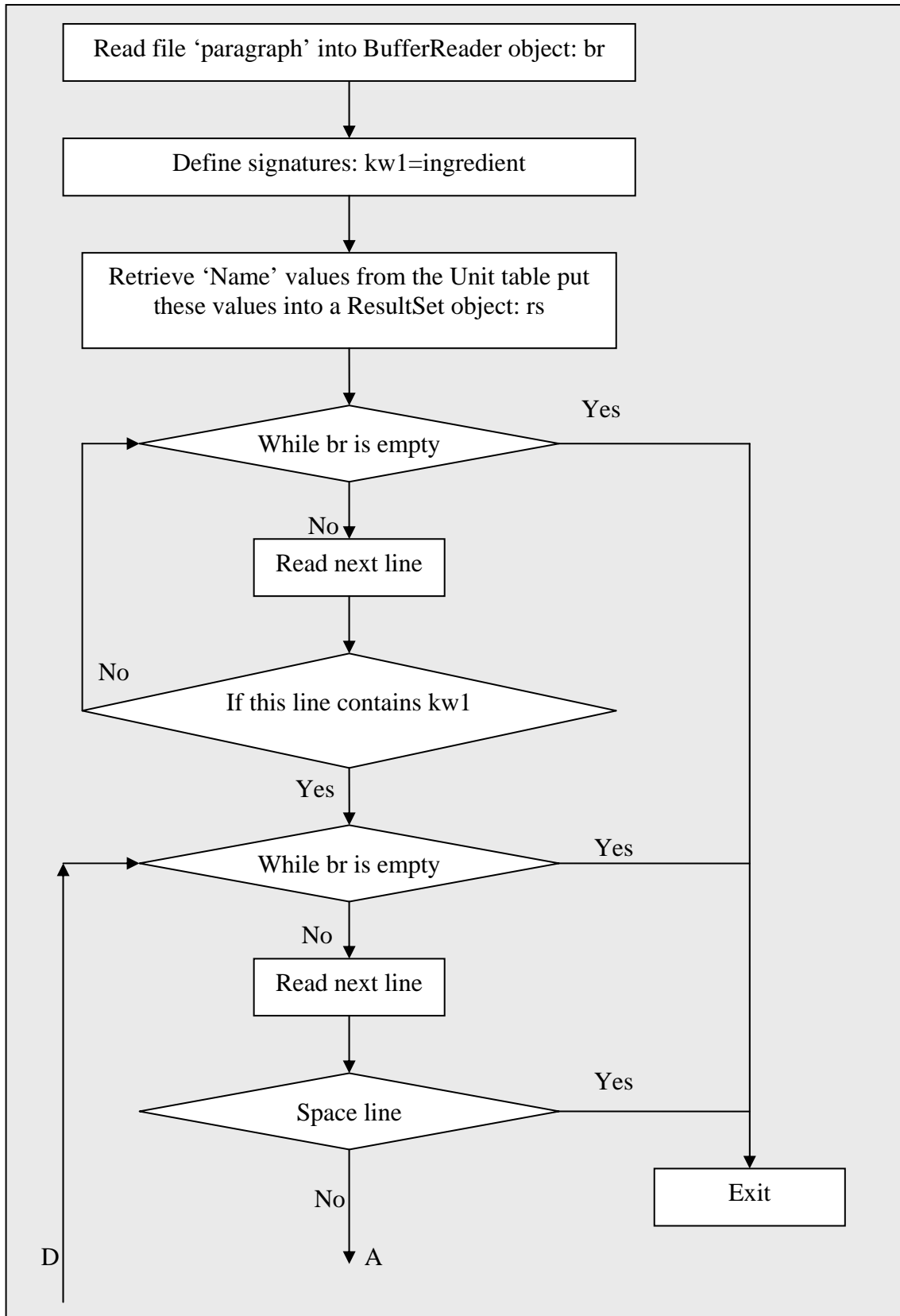


Figure 17 Flow06-a

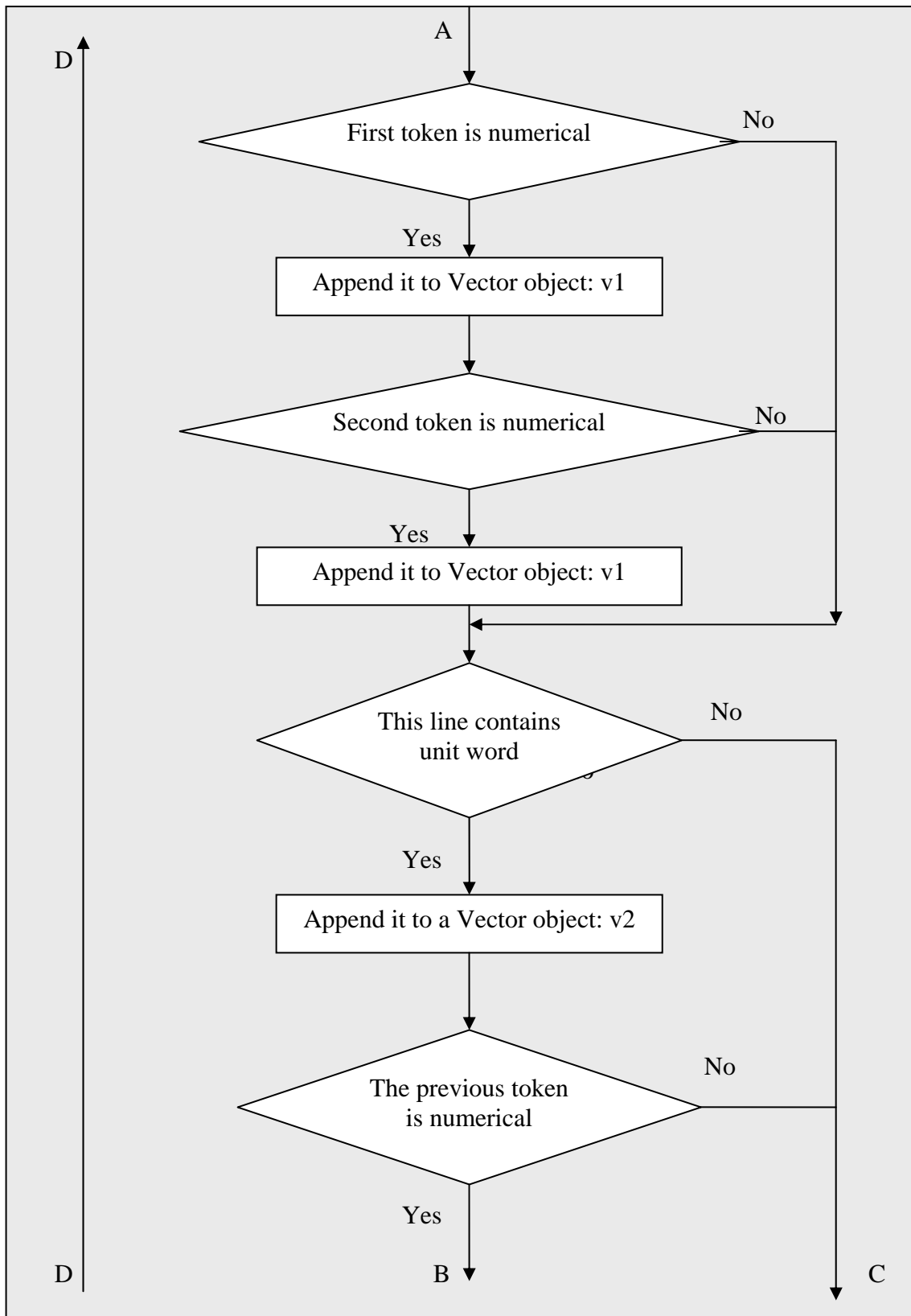


Figure 18 Flow06-b

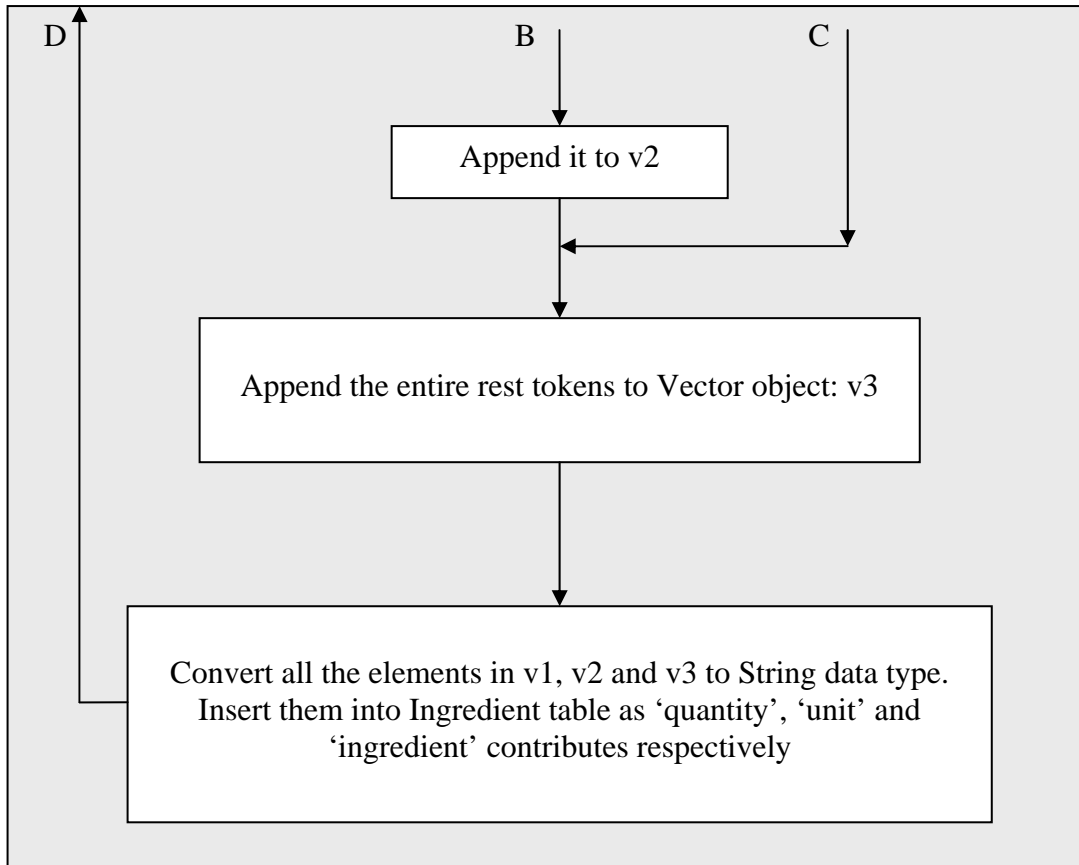


Figure 19 Flow06-c

2.3.3 View Implementation

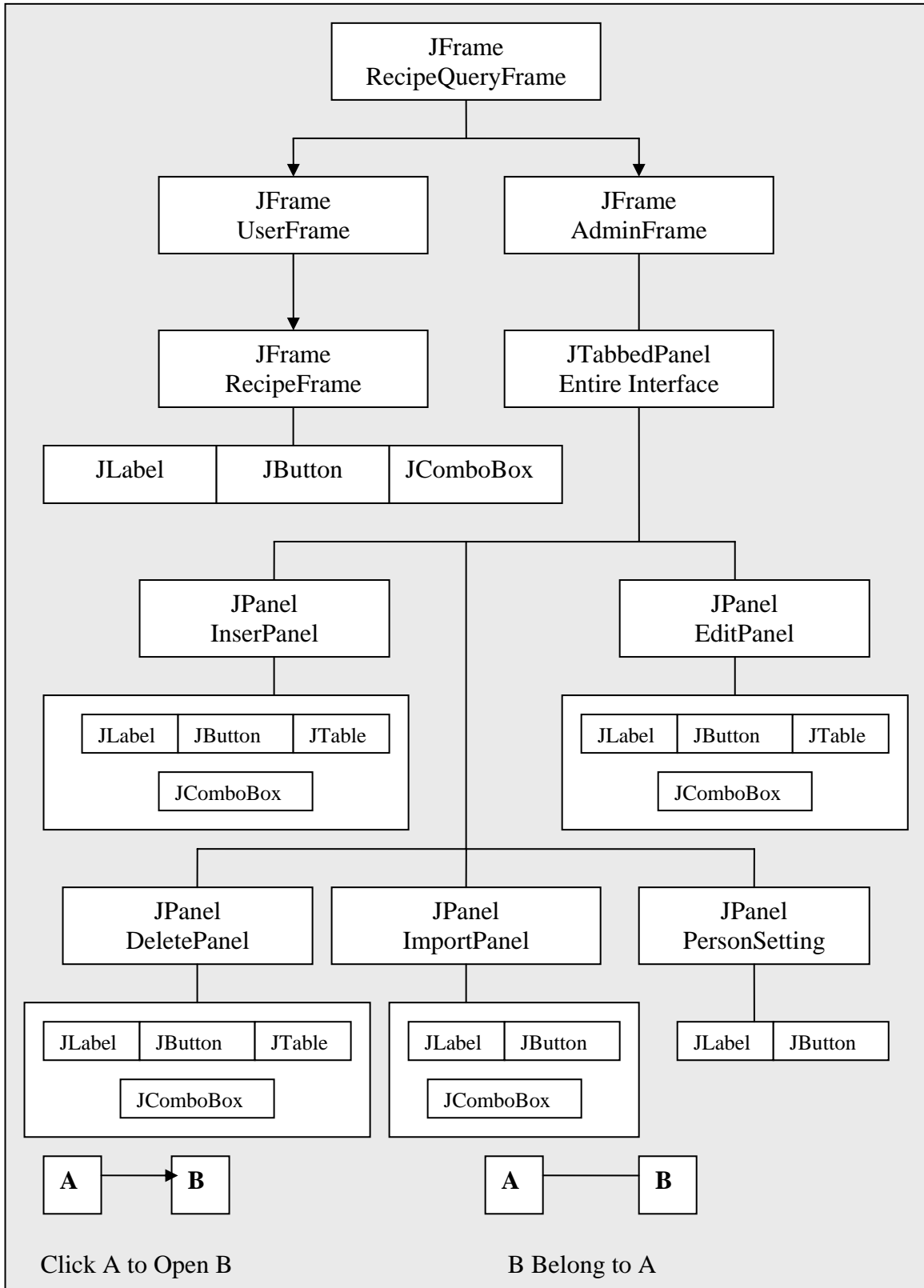


Figure 20 GUI Component Hierarchy Tree

2.3.4 Controller Implementation

The controller acts as the bridge between the user and the application. The controller receives the input from the user and informs the model and view to perform the corresponding actions. For example, when the user clicks the mouse button or chooses a menu item, it is the controller that determine how the application should response.

The implementations of the key controllers are illustrated by the UML sequence diagrams as below.

- Diagram 01 – controller for system entrance frame

When the 'General User' button is clicked by the user, the UserFrame class was called and then the search recipe window is opened. When the 'Administrator' button is clicked, the RecipeQueryFrame class will check the administrator's id and password. If both the id and password are correct, the AdminFrame class is called and then the modify recipe database window is opened. Otherwise, an error message will be shown to the user.

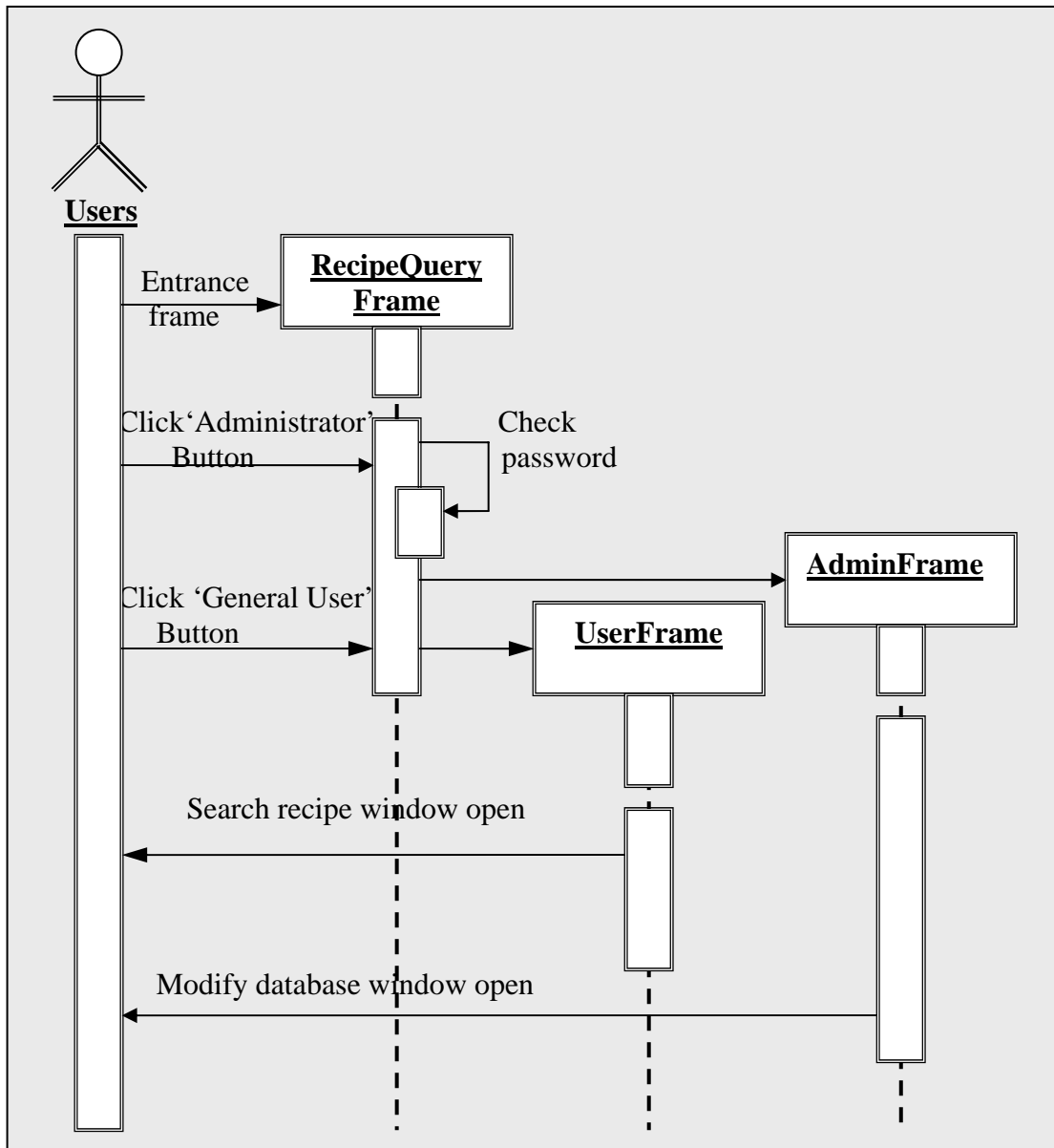


Figure 21 Diagram 01: System Entrance Frame Sequence Diagram

- Diagram 02 – controller for inserting new recipe panel

When the 'Insert' button is clicked by the user, the InsertPanel class firstly checks if the recipe data input by the user completed or not. The 'insert' method in ModifyRecipe class will be called to insert the new recipe into the database if the input recipe data is completed. If the insert action is successful, a 'true' value will be returned by the method and the message 'New recipe has been inserted into database' will be shown to the user. Otherwise, a 'false' value will be returned and the message 'This recipe has existed in database' will be shown to the user. The 'Clear' button is used for initializing this panel.

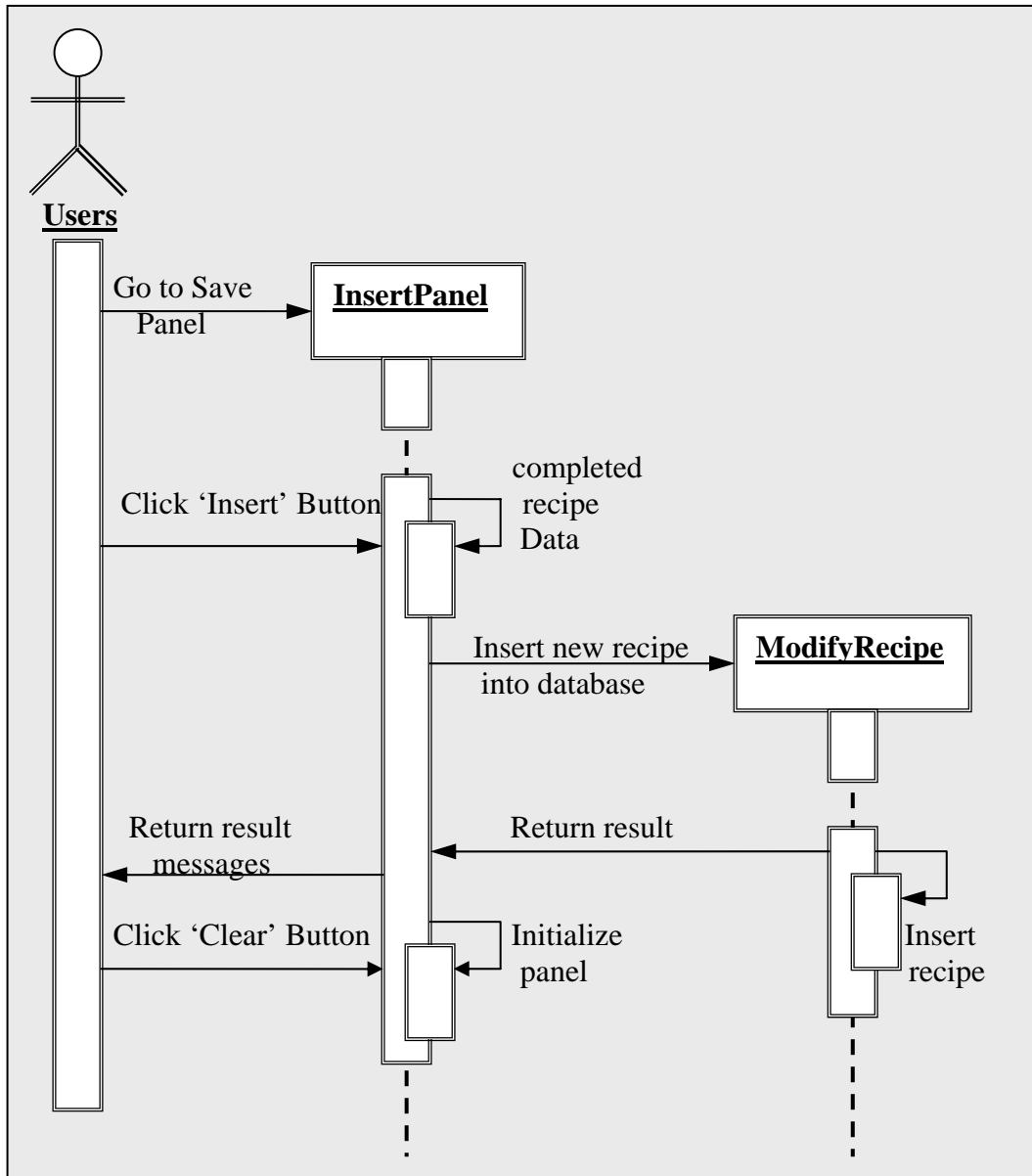


Figure 22 Diagram 02: Insert New Recipe Sequence Diagram

- Diagram 03 – controller for editing recipe panel

When the 'Update' button is clicked by the user, the EditPanel class firstly checks if the recipe data input by the user completed or not. The 'edit' method in ModifyRecipe class will be called to edit the old recipe data in the database if the input recipe data is completed. The message 'Recipe has been updated in database' will be shown to the user.

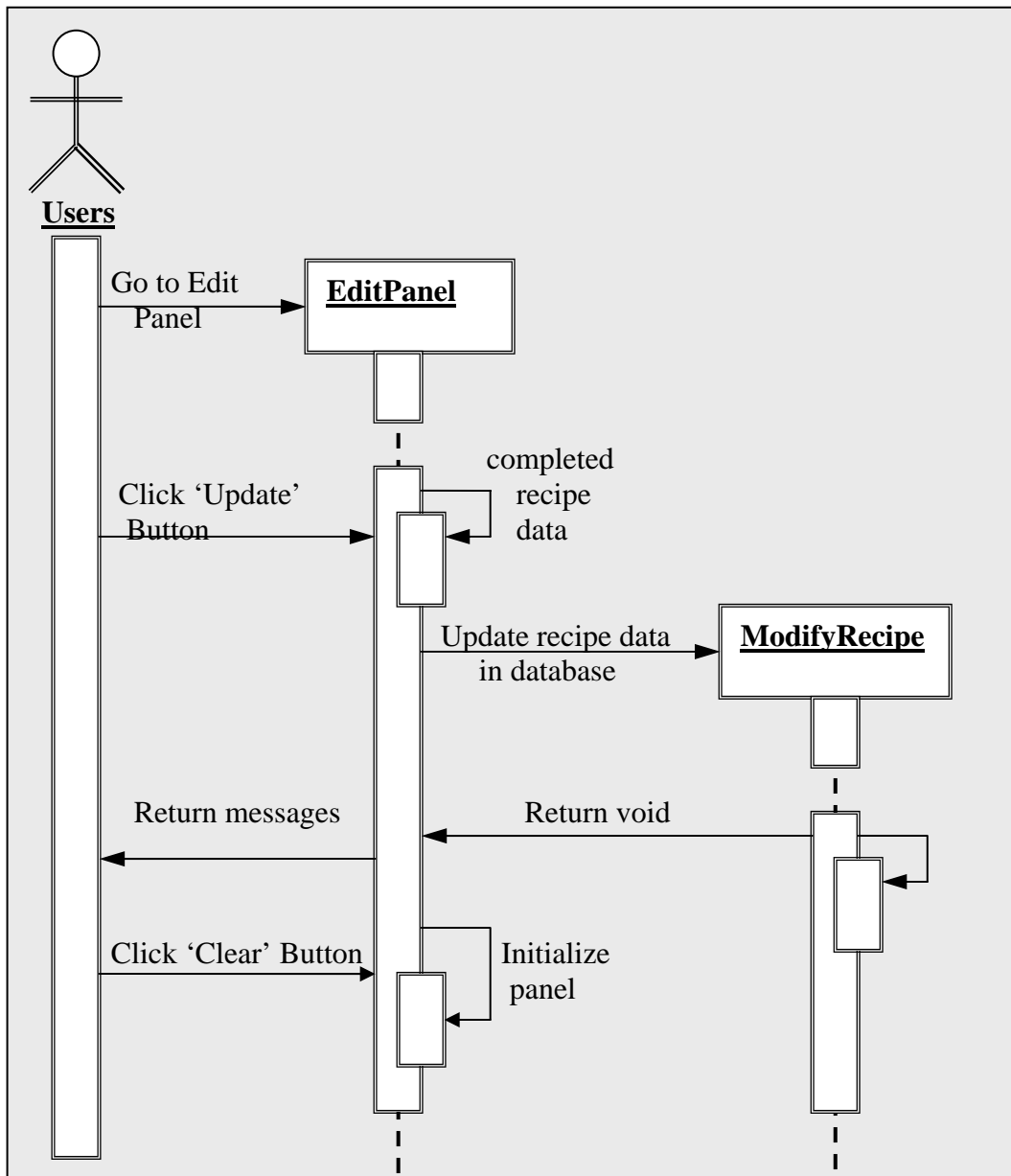


Figure 23 Diagram 03: Edit Recipe Sequence Diagram

- Diagram 04 – controller for delete recipe panel

When the 'Delete' button is clicked by the user, the DeletePanel class firstly checks if the recipe ID is selected by the user. The 'delete' method in ModifyRecipe class will be called to delete the recipe from the database if the recipe ID is selected. The message 'Recipe has been deleted from database' will be shown to the user.

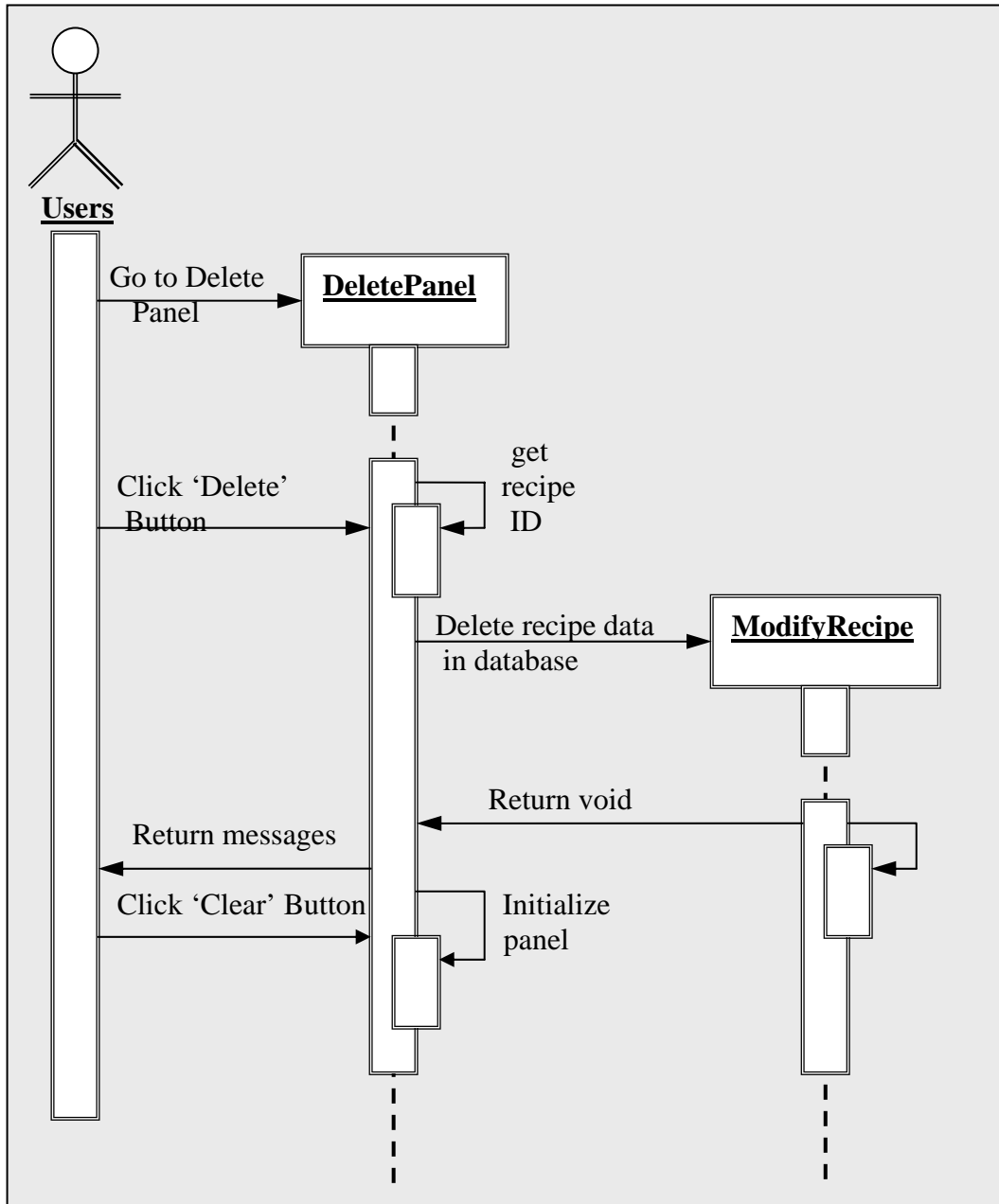


Figure 24 Diagram 04: Delete Recipe Sequence Diagram

- Diagram 05 – controller for importing external recipe panel

When the 'Browse' button is clicked by the user, a dialog for browsing and choosing file is opened. When the 'Import' button is clicked, the ImportPanel class will check if any external file is selected. The 'Extract' method in ExtractInformation class will be called to import the recipe data from the file. The corresponding value, which is returned by this method, will decide whether the 'Successful' or 'Fail' message should be shown to the user. If the external file was imported successfully, the RecipeDisplay class will be called to display this recipe information.

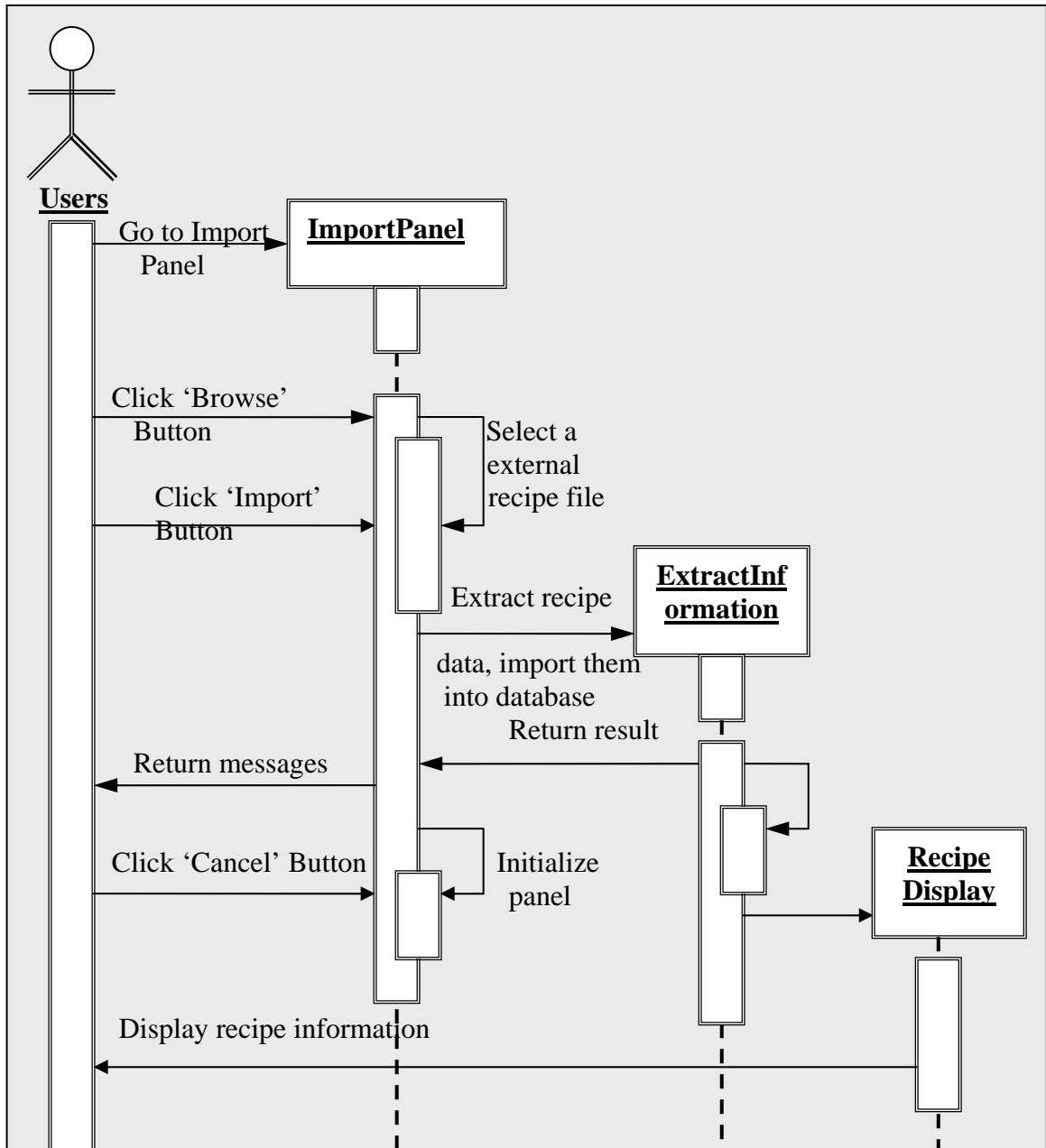


Figure 25 Diagram 05: Import Recipe Sequence Diagram

- Diagram 06 – controller for search recipe frame

When the 'Ok' button is clicked by the user, the 'searchRecipe' method in ModifyRecipe class will be called to search recipes from the database according to the input data by the user. The message 'No recipe matched' will be shown to the user if there aren't any matched recipes found in the database. If some matched recipes is found, the RecipeFrame class will be called to display these recipes' information.

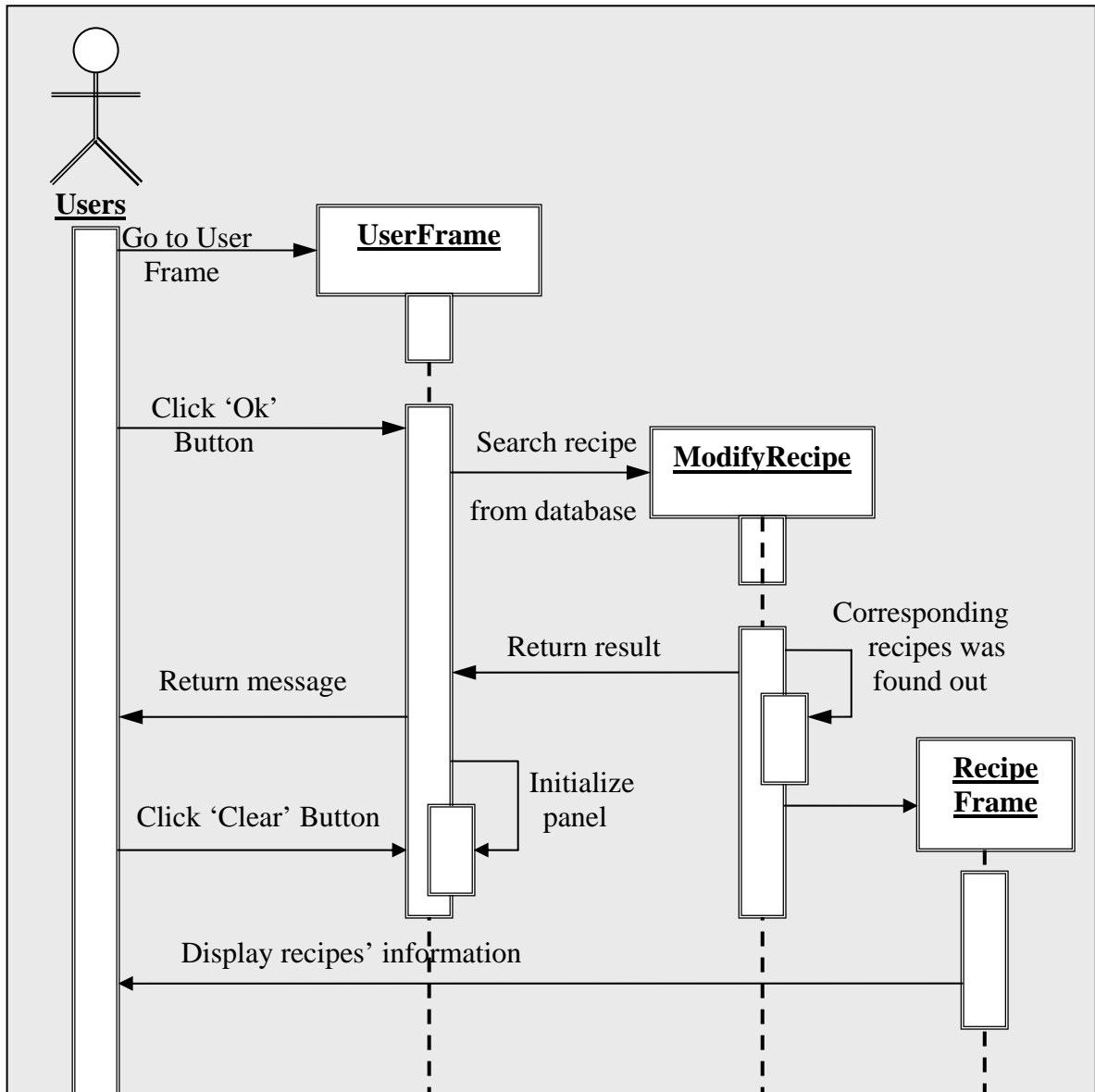


Figure 26 Diagram 06: Search Recipe Frame Sequence Diagram

- Diagram 07 – controller for changing password panel

When the 'Modify' button is clicked by the user, the PerPanel class will check if both the original password and the new password entered by the user are correct. If both the original password and the new password are correct, the message 'New password has been admitted' will be returned. Otherwise, an error message will be returned.

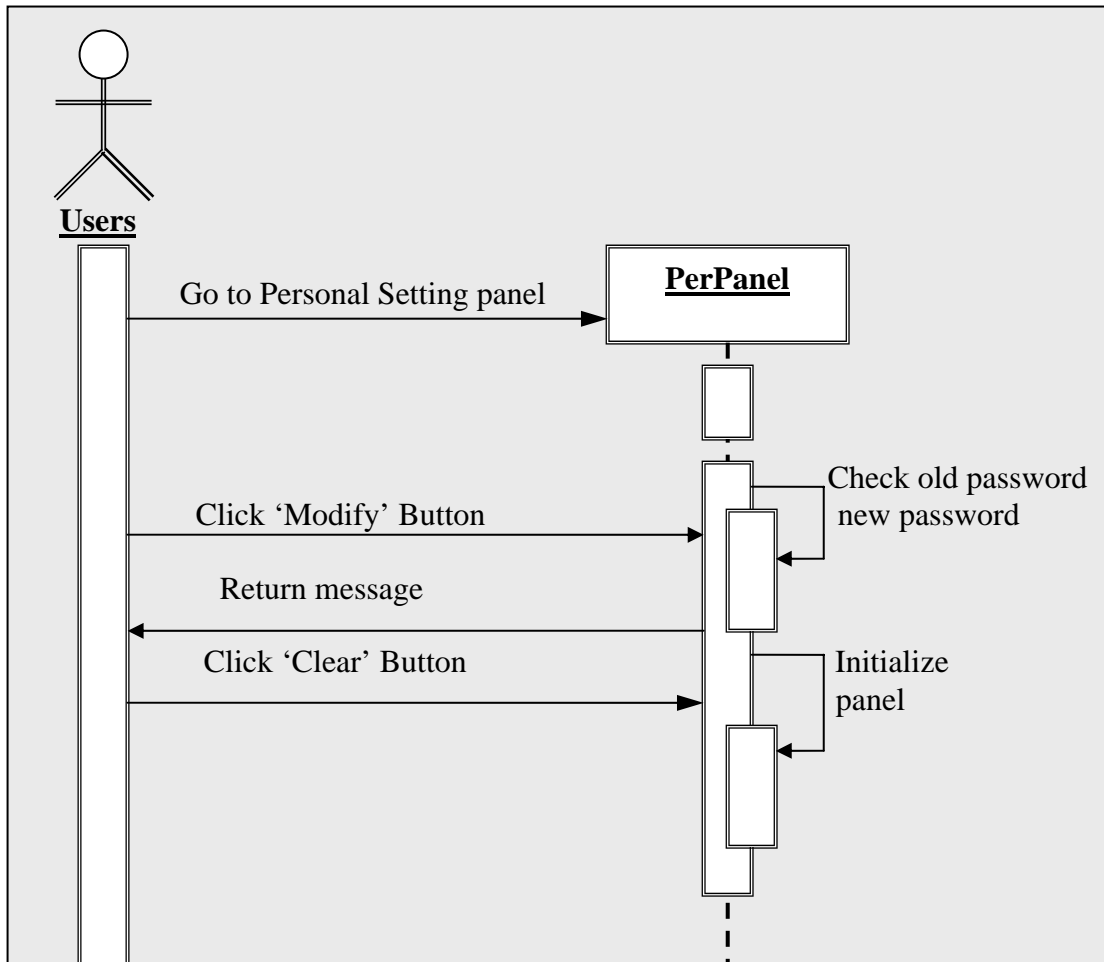


Figure 27 Diagram 07: Change Password Panel Sequence Diagram

2.4 System Test and Results

Once the entire system has been implemented, it has to be fully tested to check if it meets the requirement specification or not.

In essence the system testing focuses on the whole system, not the individual parts.

There are two types of Software System Test: functional test and structural test.

- Functional Test

The functional test is to separate the program into many function models and then based on the abstract data check the generated test results from each function models. The functional test is to check if all the functions can be performed normally and never considers the program's internal structure.

- Structural Test

Structural test is designed and performed according to the internal structures of the program. The tester should check every branch in the program and get the test results. Compared to the functional test, the structural test focuses on the internal structure of the program. Although the user prefers to do the functional test based on program specification guide, some latent errors can be found out through the structural test rather than the functional test.

In this chapter, I will give out some key functional tests and their results, the system structure test will be shown in the appendix.

First, I will give an overview about the GUI (Graphic user Interfaces) of the system.

- System Entrance Interface



Figure 28 the System Entrance Interface

- General User Interface

Recipe Query !

Title :

Ingredient :

Category :

Ok Clear Back

Figure 29 the General User Interface

- Administrator Interface (Insert Panel, Edit Panel, Delete Panel, Import Panel, Personal Setting Panel)

Recipe Database Update

Insert Edit Delete Import Personal Setting

Title :

Choose Category

Quantity	Unit	Ingredient

Direciton :

Save Clear

Figure 30 the Administrator Interface-Insert Panel

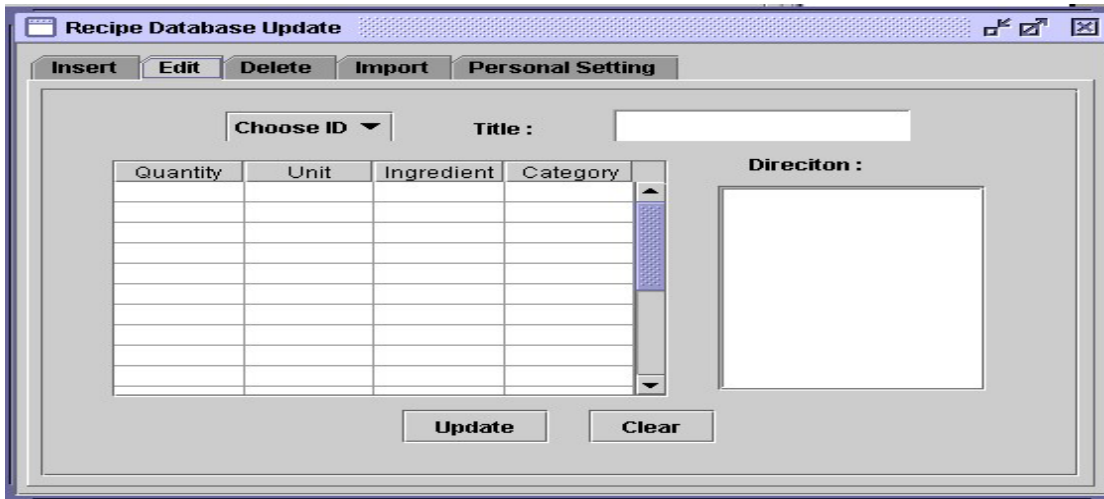


Figure 31 the Administrator Interface-edit panel

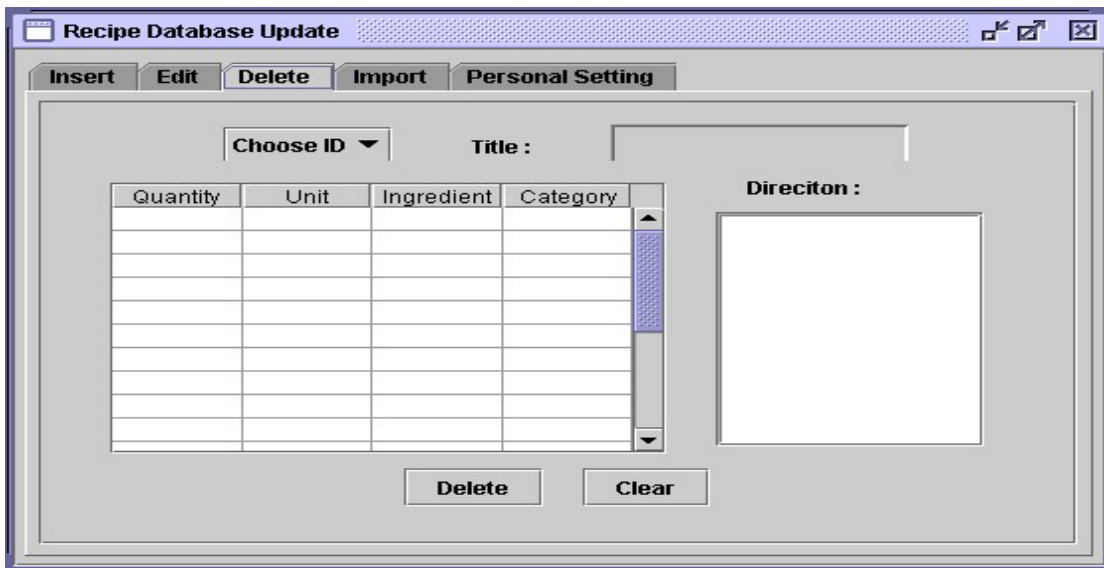


Figure 32 the Administrator Interface-delete panel

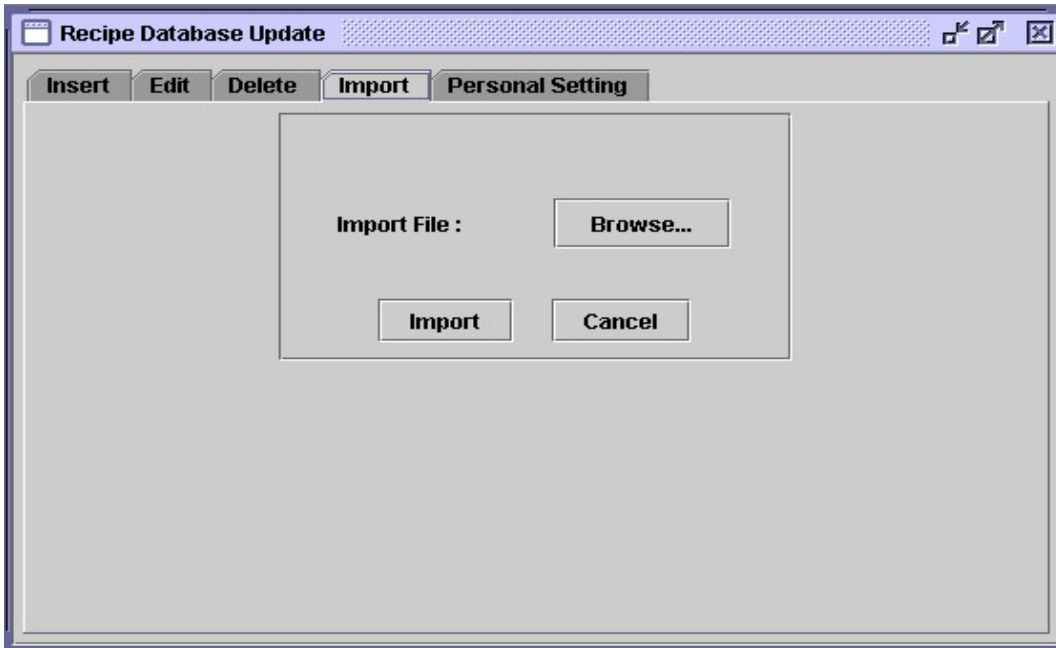


Figure 33 the Administrator Interface-import panel

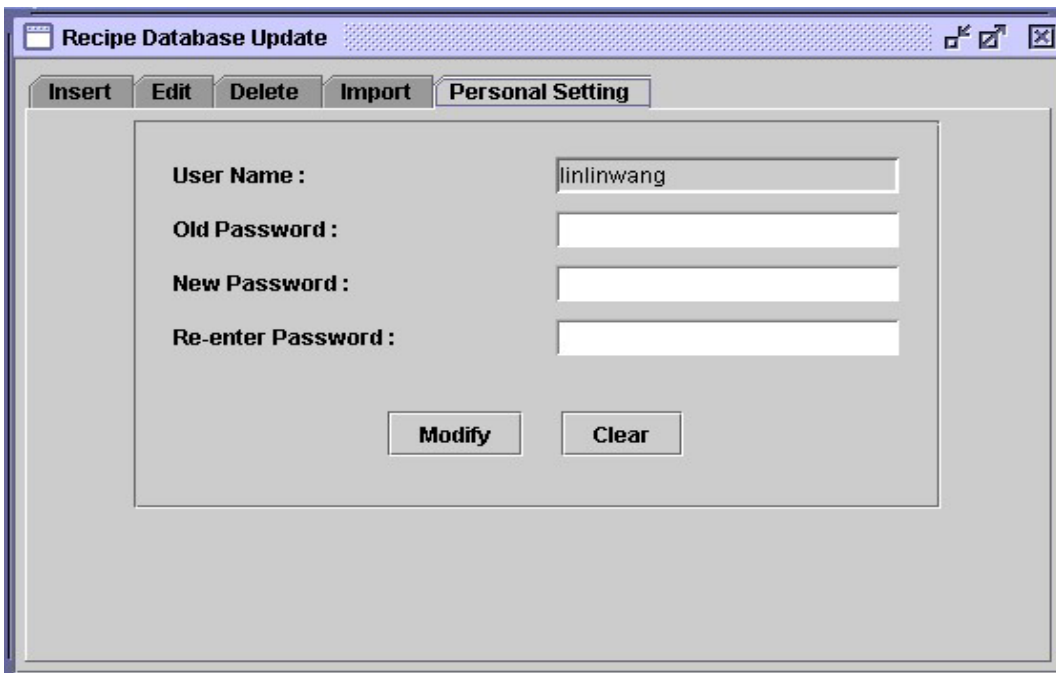


Figure 34 the Administrator Interface-personal setting panel

The recipe database system offers some basic functions such as: login, search recipe, modify recipe database, modify administrator's information etc. Here, I focused on the functions: Administrator Login, Search Recipes and Modify Database.

- Search the Recipes

Assume that there are 4 recipes already exist in the database.



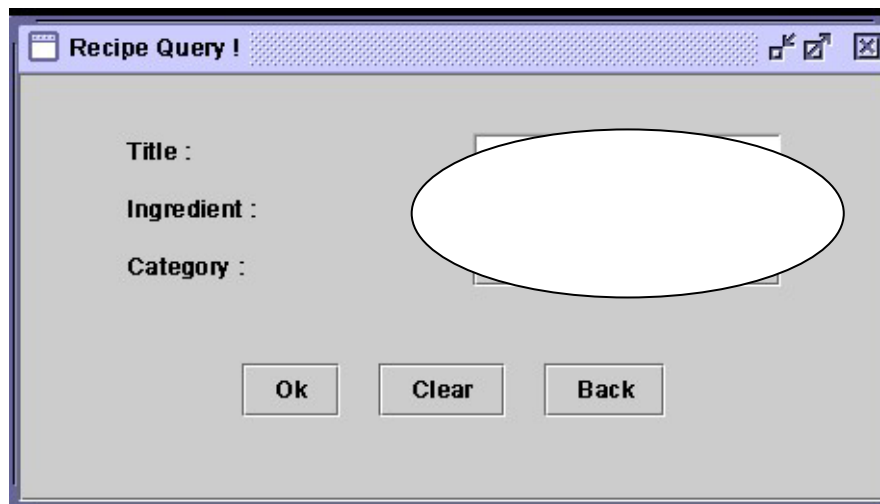
Rec_ID	Title	Category	Direct
1	Beef Pepper Steak.txt	Beef	In a small, nonporous bowl, combine the peppercorns, t
2	Dutch Oven Buttermilk Cornbread.txt	Others	In a large bowl, mix the dry ingredients together, and
3	Ground steak Mexican Style.txt	Beef	In a large nonstick skillet, cook and stir ground beef
4	Macaroni With Beans.txt	Vegetable & Fruit	Prepare pasta according to package directions. While p

Figure 35 the example recipe table

When the user clicks the 'General User' button on the system entrance interface, the search recipe interface will appear.

1. Test 1:

The input ingredient's keyword is pepper and the recipe category is beef:



Recipe Query !

Title :

Ingredient :

Category :

Ok Clear Back

Figure 36 input for the test 1

Press the Ok button, two matched recipes in the database have been found.

The Recipe information display result is shown as below:

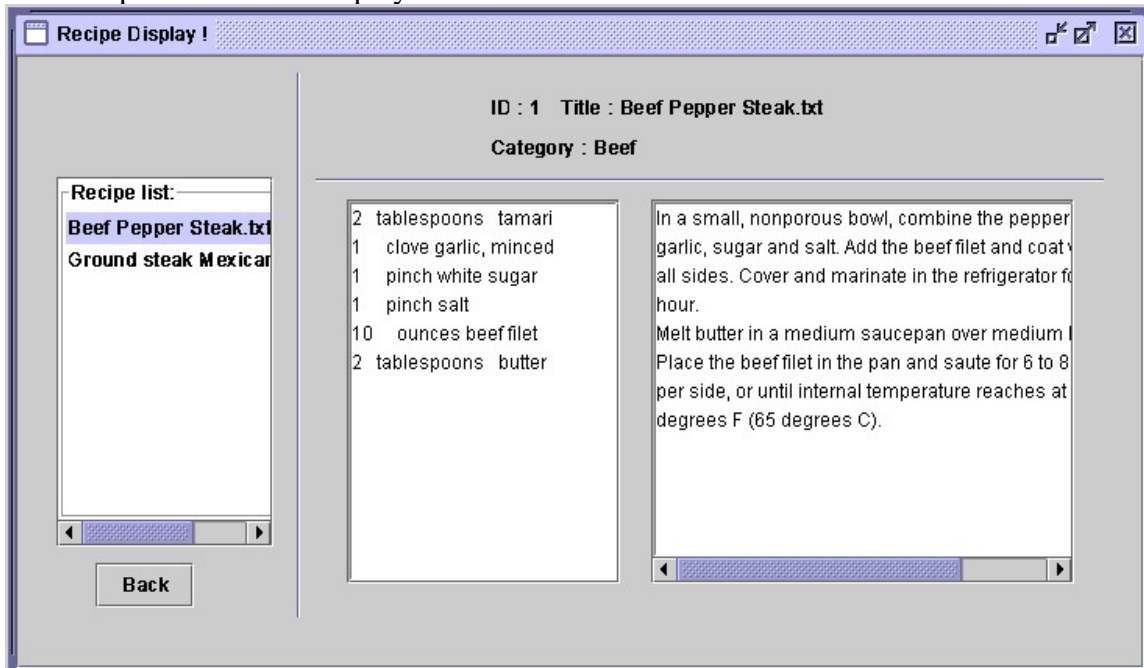


Figure 37 result of the test 1

2. Test 2:

The input recipe title keyword is 'cake', the recipe ingredient keyword is 'banana', and the recipe category is not specified.

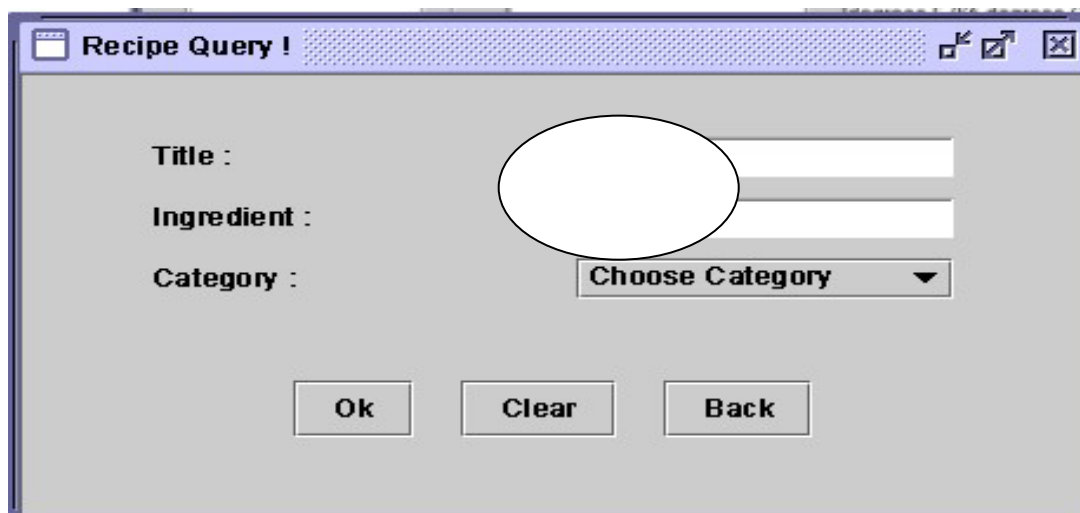


Figure 38 the input for the test 2

Press the Ok button, we can get the following result:

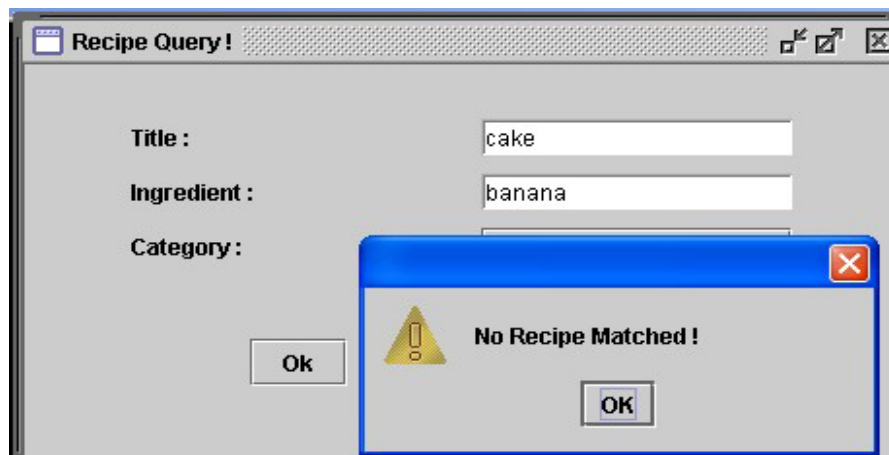


Figure 39 the result of the test 2

No recipe is matched by those input keywords.

- Administrator Login

Assume that there is one administrator record already exist in the database. The name is: linlinwang and the password is: 19781130.

	Name	Password
▶	linlinwang	19781130
*		

Figure 40 the example admin table

When the user clicks the 'Administrator' button on the system entrance interface, a check administrator's information window will appear which indicates the user to input name and password.



Figure 41 the input for the test 'admin. login'

Click Ok button, the administrator's interface will be shown.

- Import External Recipe

I will test the following three conditions for importing the external recipe files:

1. The recipe file is valid and it can be imported successfully.
2. The recipe file is invalid.
3. The recipe that will be imported has existed in the database

The import interface is:

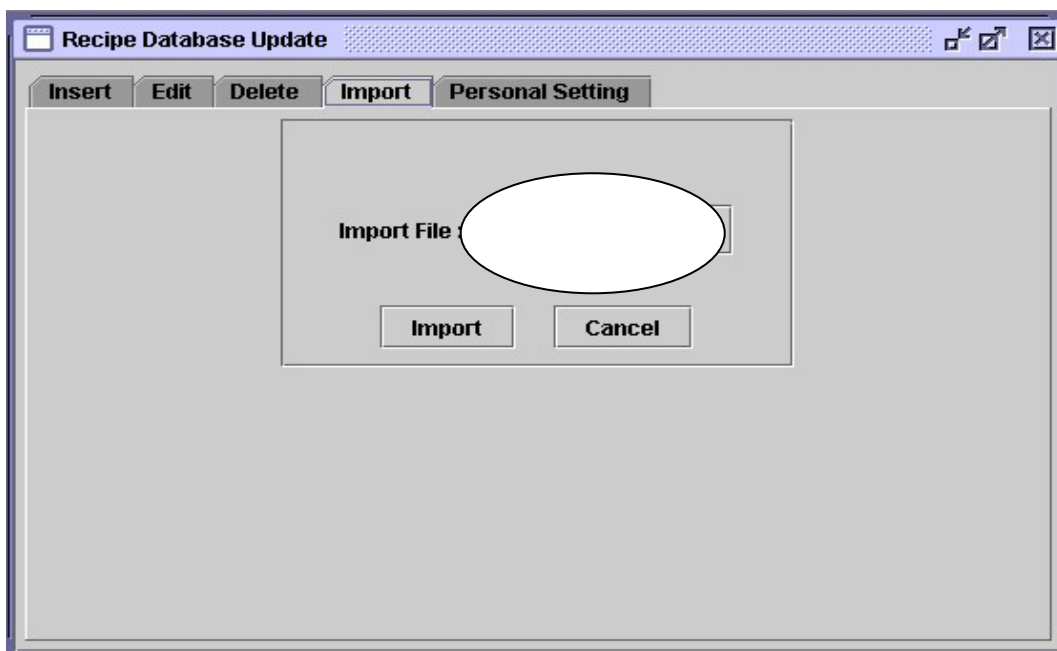


Figure 42 the import panel

Click 'Browse' button, an open file window will appear:

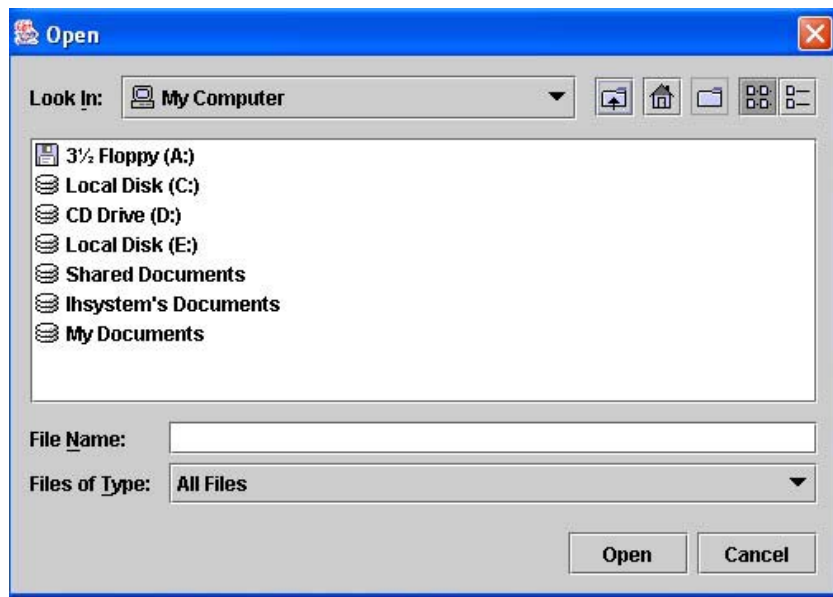


Figure 43 the window for selecting files

1. Test 1 – valid recipe file is imported successfully

Assume that an external recipe file named “Whipping Cream Pound Cake.txt” was selected.



Figure 44 a valid recipe file is selected

This recipe file's content was shown below (.txt format):

```
Whipping Cream Pound Cake.txt - Notepad
File Edit Format View Help
error text
your name:(optional) &laquo; Please check: error text
your email address:&laquo; Please check: it appears that you may
have entered an incorrect email address.
add a personal note:
(optional) &laquo; Please check: error text

Also send a copy to yourself
email format:
Complete Recipe
Link only
1 1/2 cups white sugar
1 cup butter
7 eggs
6 tablespoons cornstarch
2 5/8 cups all-purpose flour
1 cup heavy whipping cream
2 tablespoons vanilla extract

Preheat oven to 350 degrees F(175 degrees C). Grease and flour a
10 inch tube pan. Set aside.
Cream together the sugar and butter until light. Continue beating
and add 7 eggs, one at a time; beating well after each egg
In a separate bowl, mix together flour and cornstarch. Beat half
of the flour mixture into the egg and sugar mixture.
Beat in 1/2 cup whipping cream, and then beat in the remainder of
the flour mixture. Finish by beating in 1/2 cup more of whipping
cream and vanilla.
Pour into prepared pan and bake for about 60 to 75 minutes. Cool
on rack for 10 minutes before turning it out onto a serving plate.

Customize this Recipe
Change to servings
Convert to: U.S Standard / Metric
About Scaling and Conversions

Weight Management
```

Figure 45 the text content of the file ‘Whipping Cream Pound Cake.txt’

Clearly, this is a valid and completed recipe file because the key words, both ‘ingredient’ and ‘direction’, exist and the recipe ingredient and direction description exist as well.

When the ‘Import’ button is clicked, the message: “New Recipe has been added into the Database” is shown to the user.

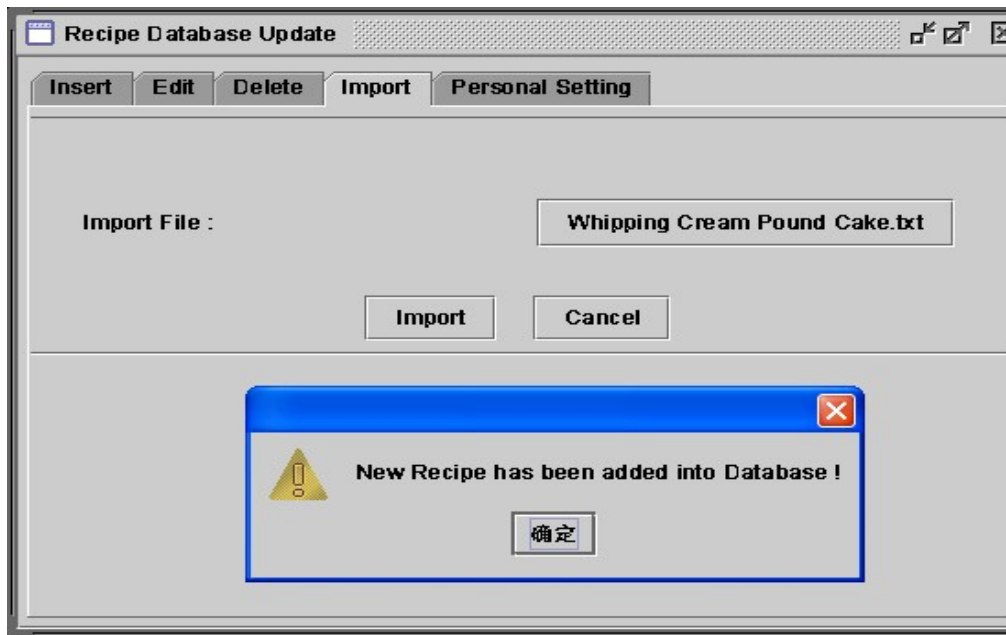


Figure 46 successfully insert the recipe 'Whipping Cream Pound Cake'

Then a recipe information window will appear to display this imported recipe:

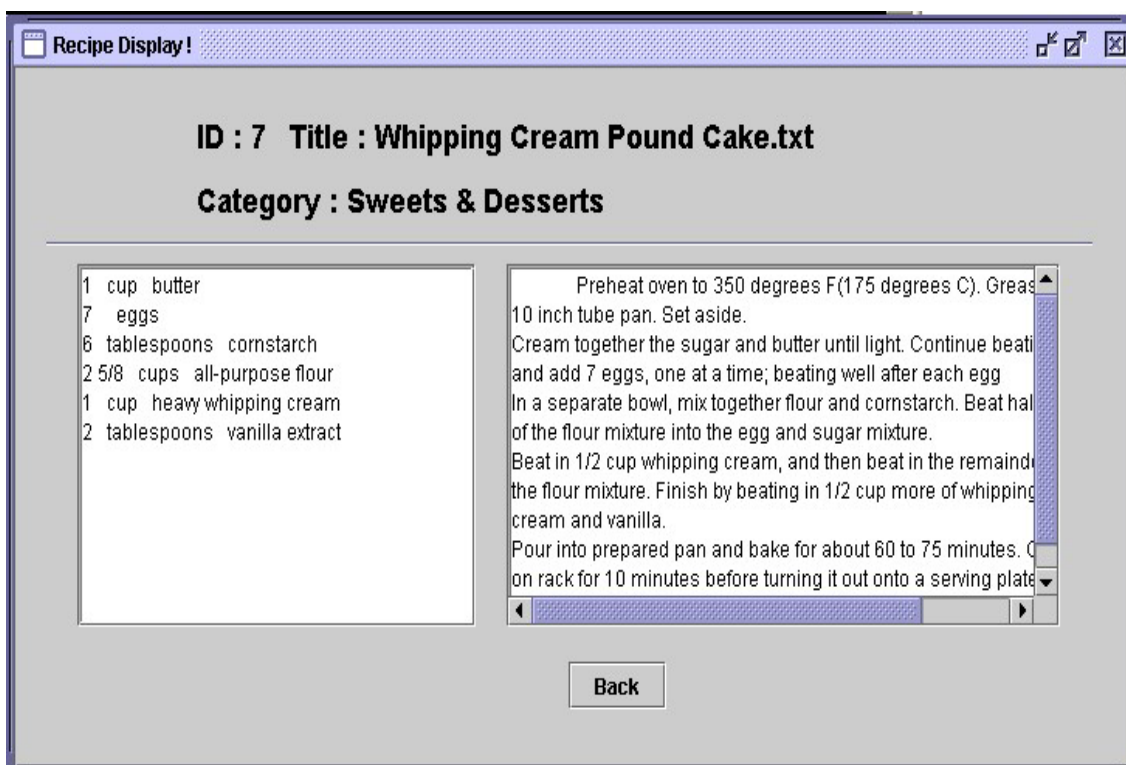


Figure 47 the display of the recipe "Whipping Cream Pound Cake"

We can find that the recipe category is set to ‘Sweets & Desserts’, because the keyword ‘Cake’ which can match the ‘Sweets & Desserts’ category through the ‘Material table’ in the database has been found in the recipe title.

Name	Category
Bean	Vegetable & Fruit
Beef	Beef
Chicken	Chichen
Filet	Seafood
Fuit	Vegetable & Fruit
Lamb	Lamb
Peanut	Vegetable & Fruit
Pig	Pork
Pork	Pork
Potato	Vegetable & Fruit
Salad	Vegetable & Fruit

Figure 48 the example material table

Moreover, a text file named ‘paragraph.txt’ was generated by the program:

```

INGREDIENTS:
  2 1/2 cups white sugar
  1 cup butter
  7 eggs
  6 tablespoons cornstarch
  2 5/8 cups all-purpose flour
  1 cup heavy whipping cream
  2 tablespoons vanilla extract

DIRECTIONS:
  Preheat oven to 350 degrees F(175 degrees C). Grease and flour a
  10 inch tube pan. Set aside.
  Cream together the sugar and butter until light. Continue beating
  and add 7 eggs, one at a time; beating well after each egg
  In a separate bowl, mix together flour and cornstarch. Beat half
  of the flour mixture into the egg and sugar mixture.
  Beat in 1/2 cup whipping cream, and then beat in the remainder of
  the flour mixture. Finish by beating in 1/2 cup more of whipping
  cream and vanilla.
  Pour into prepared pan and bake for about 60 to 75 minutes. Cool
  on rack for 10 minutes before turning it out onto a serving plate.
  
```

Figure 49 thetemporary paragraph generated during the import

After the recipe data is imported into the database, one recipe record and 6 ingredient’s records are created in the ‘Recipe table’ and ‘Ingredient table’ respectively.

Rec_ID	Title	Category	
1	Beef Pepper Steak.txt	Beef	In a small, nonporous bowl, combine
2	Dutch Oven Buttermilk Cornbread.txt	Others	In a large bowl, mix the dry ingredi
3	Ground steak Mexican Style.txt	Beef	In a large nonstick skillet, cook an
4	Macaroni With Beans.txt	Vegetable & Fruit	Prepare pasta according to package d
5	Soups and Stews.txt	Others	1. Melt butter in a 5-6 qt pan over
*	(AutoNumber)		

Figure 50 the target recipe has been inserted into the recipe table

Ing_ID	Rec_ID	Unit	Ingredient
*			
			ur
			g cream
			extract
*			(AutoNumber)

Figure 51 the ingredients of the targeted recipe has been inserted into the ingredient table

2. Test 2 – recipe has existed in the database

Import the recipe file named “Whipping Cream Pound Cake.txt” again. A message ‘Recipe has already existed in database’ will be returned to the user.

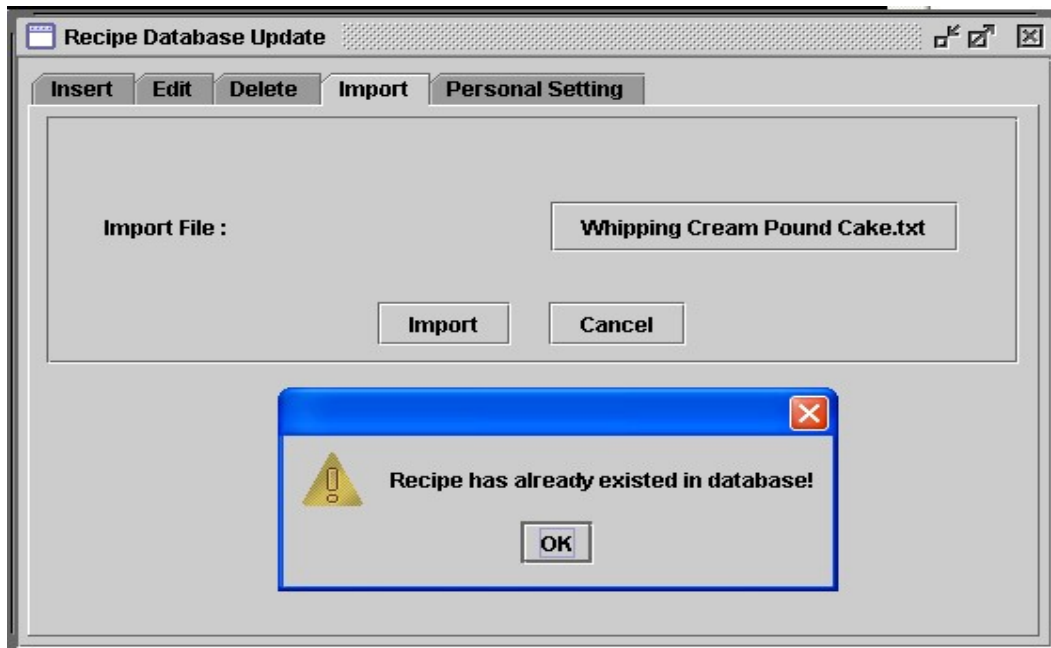


Figure 52 the recipe 'Whipping Cream Pound Cake.txt' has already stored in the database

3. Test 3 – invalid external recipe file

The invalid files can be two types: the one that lost the keywords information and the one that doesn't have the description of the ingredient or direction. Assume that the test file we used here has lost the ingredient description part: Mom Best Peanut Brittle.txt.

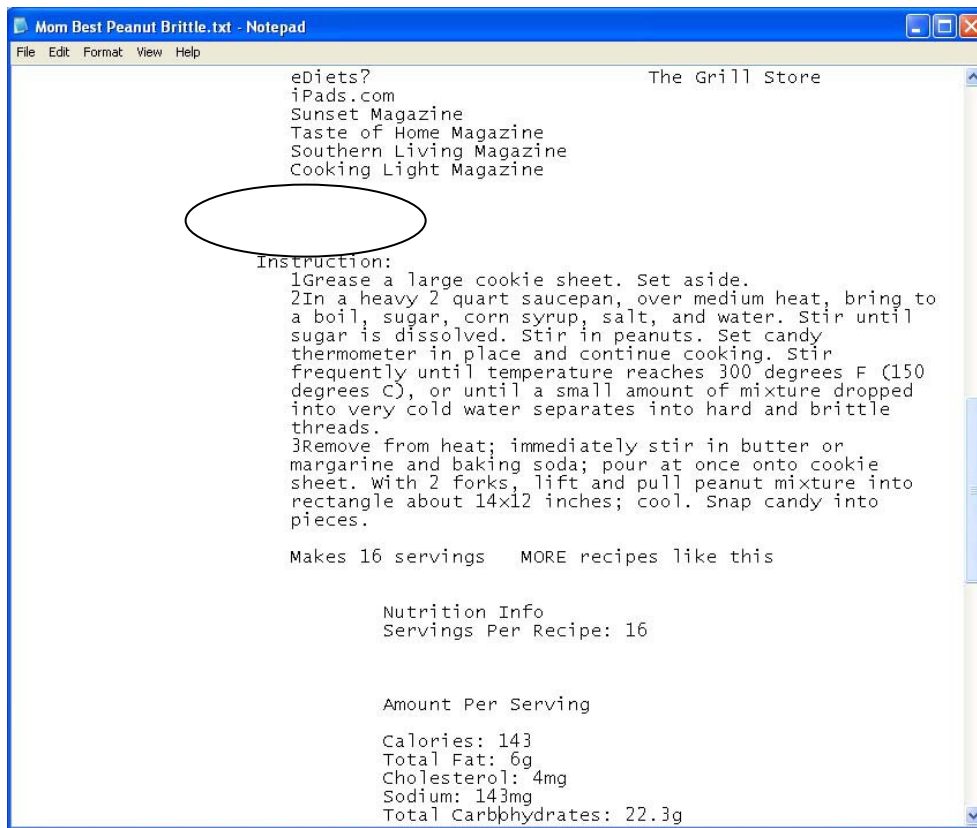


Figure 53 the example invalid file which loses the ingredient description

Then the message: 'Recipe File is Valid' will be returned.

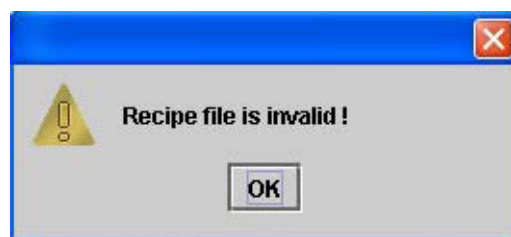


Figure 54 the response indicates that the file is invalid

2.5 Summary

In solution 1, the recipe system have been analysed, designed and implemented. According to the “project statement” listed in chapter 1, the recipe system is designed to offer the following functions:

- Insert a new recipe record manually
- Modify the items of the recipe record manually
- Delete the recipe record manually
- Import the external recipe files automatically
- Search the recipe by the category manually
- Search the recipe by the ingredients manually
- Search the recipe by the title manually
- Modify the password of the super user

It has fulfilled the requirements defined in the “project statement”, therefore we can say the solution 1 is successful and has achieved the objective of this project.

However, there are still some limitations in solution 1.

In solution 1, the program can only import the recipe files which are saved as *.txt format. The recipe file must contain the special signatures which indicates the start of the recipe contents such as ‘ingredient’ and ‘direction’ because the system uses these two signatures to locate where the ingredient and direction description are. In addition, in the original recipe files, both the ingredient part and the direction part of the recipe should be described in the individual paragraphs since the extraction of such contents is based on the ‘paragraph’, i.e. the paragraph immediately after the special signature words is considered as either the ingredients description or the direction.

In solution 1, the system simply treats the name of the recipe file as the recipe title since the recipe title is located randomly in the file and can’t be recognized by the system. For the category extraction, the accuracy can be increased by filling in as more as possible ingredients and their categories into the Material table.

As solution 1 still has the limitations listed above, the system can only handle a few number of the recipe files which exactly meet the system requirement. For the rest types of the recipe files, the system can only discard them directly. In order to make the system more flexible and handle more types of the recipe files, there should be some improvement and optimization made for the system, especially for the import functions. That is the motivation for introducing the solution 2!

3. Solution 2

The solution 2 is a kind of improvement for the solution 1.

As we have known, in solution 1 the user has to manually save the recipe files as *.txt format and the program can only recognize the recipe file which contains two special keywords: the 'ingredient' and the 'direction'. And the program simply treats the file's name as the title of the recipe.

In solution 2, HTML format recipe files saved on the local disk can be imported directly without any modification. This means a new algorithm should be found out to extract the recipe data without the need of any special keywords. And the recipe title should be found and imported automatically in solution 2.

Obviously, the solution 2 is more convenient and practical, as it can recognize and import much more recipe files.

3.1 Analysis

3.1.1 HTML Document Analysis

HTML (**H**yper**T**ext **M**arkup **L**anguage) consists of tags, which are enclosed in angle-brackets (< >). The tags typically occur in begin-end pairs, as shown in the following form:

```
<tag> ...content... </tag>
```

The <tag> indicates the beginning of a tag pair, and the </tag> indicates the end. The tag inside the angle-brackets is the actual name of the tag being discussed. The content within a tag pair are formulated according to the rules that defined by the tag. For instance, the text within a pair of <I></I> is displayed in the Italian style. One has to be mentioned is that not all the tags in HTML are paired. Some tags such as the line-break tag don't have the end tag. Such tags are called empty tags.

The pairs of tag set could include another pairs of tag set. Therefore it is important to keep the tag set nested within each other. The following figure shows such an example.

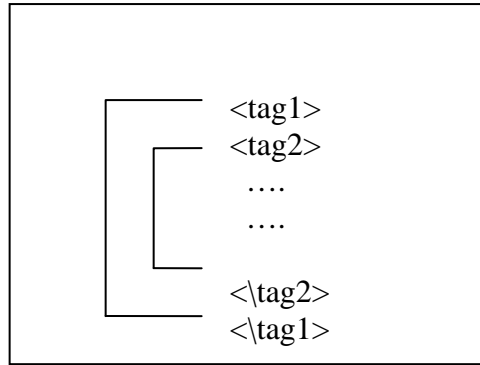


Figure 55 the nested tag pairs

As the pairs of tag set has defined the rules for formulating and displaying the content, it is often the case that the arrangement of text within a tag set is irrelevant for the display on the screen. It means the blank areas in a text file, such as empty lines and extra spaces, sometimes will be neglected by the HTML parsers. For example, within a “paragraph” tag set, the text can be stored in one line, or in several separate lines, or with every word on its own line. However the display of it on the screen will be exactly the same.

3.1.2 External Recipe Files Analysis

Parsing HTML file can be a difficult job, especially in the case that the semantic parsing is requested. Since the HTML specification is loosely defined and almost no HTML designer follows it, there could be dozens of ways for implementing a single HTML page. For instances, the tag name may be uppercase, lowercase or mixed case. Element names may be uppercase or lowercase, and some end tags may or may not be used (such as `</p>`, ``). Therefore in order to well design a HTML parser as requested, the analysis on as more as possible HTML file structures is a must and the design will be a long term process.

A normal recipe web page looks like the following example:

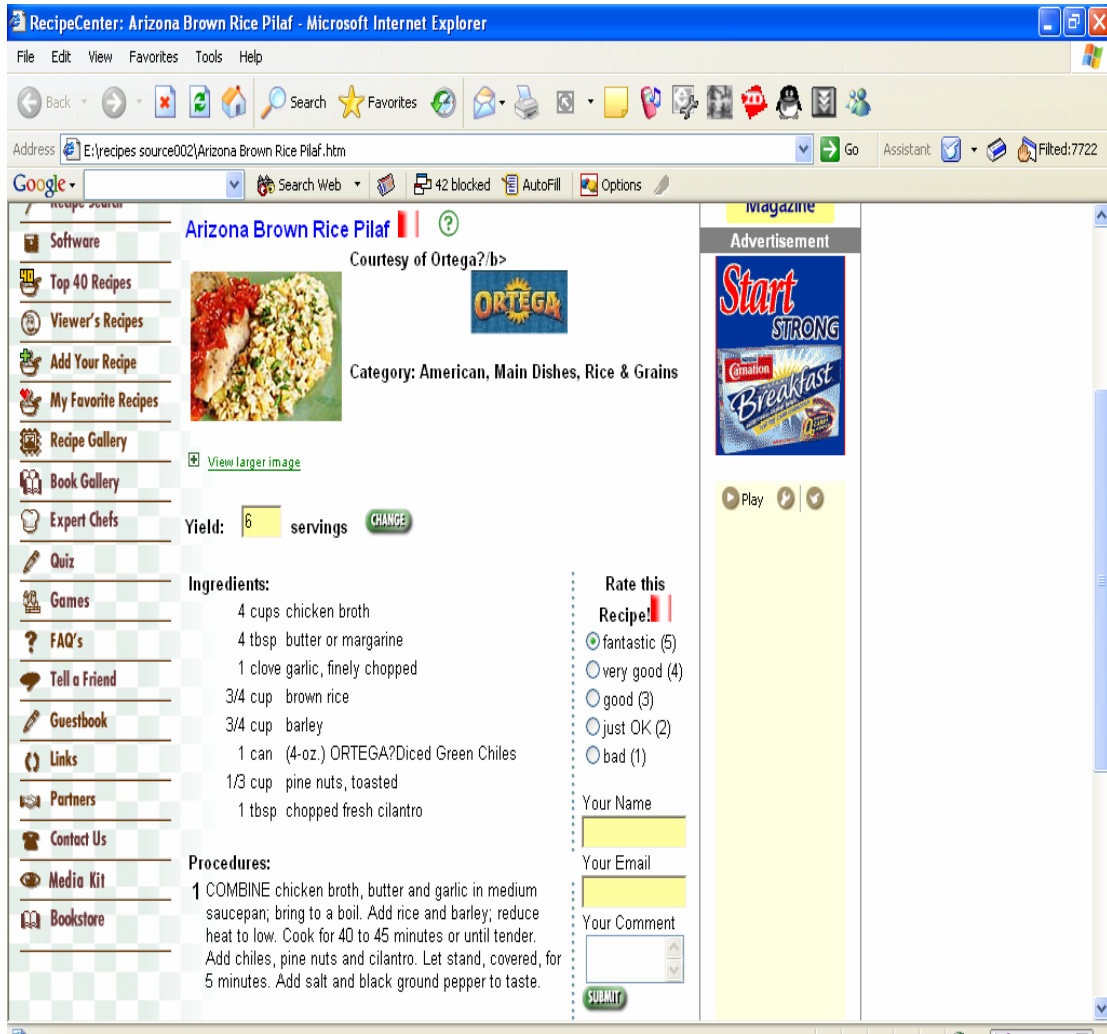


Figure 56 the example HTML page

The source code for that HTML page is displayed as below:

```
Arizona Brown Rice Pilaf.htm - Notepad
File Edit Format View Help
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- saved from url=(0050)http://www.recipecenter.com/Recipe.asp?Code=301274 -->
<HTML><HEAD><TITLE>RecipeCenter: Arizona Brown Rice Pilaf</TITLE>
<META http-equiv=content-type content="text/html; charset=gb2312">
<META
content="Recipe Center has more than 100,000 recipes. Download the RecipeCenter
Software to manage your recipes, import recipe from our collection and exchange
recipes with friends. Recipe features on line recipe sizing that calculates the
quantity of recipe ingredients needed. Excellent for chefs, cooks or anyone."
name=description>
```

Figure 57 the HTML source code (a) for the example page

```

Arizona Brown Rice Pilaf.htm - Notepad
File Edit Format View Help
<TR>
<TD vAlign=top><FONT face=arial
size=2><B>Ingredients:</B></FONT><BR>
<table border="1" style="width: 100%; height: 100%; border-collapse: collapse;">
|  |
| --- |
|  |

```

Figure 58 the source code (b) for the example HTML page

```

<TABLE WIDTH= 100% >
<TBODY>
<TR>
<TD vAlign=top align=right><FONT face=Arial,Helvetica
size=3><B>1</B></FONT></TD>
<TD vAlign=top><FONT face=Arial,Helvetica size=2>COMBINE
chicken broth, butter and garlic in medium saucepan;
bring to a boil. Add rice and barley; reduce heat to
low. Cook for 40 to 45 minutes or until tender. Add
chiles, pine nuts and cilantro. Let stand, covered, for
5 minutes. Add salt and black ground pepper to
taste.</FONT></TD></TR></TBODY></TABLE></TD>
<TD vAlign=top width=5><IMG
src="Arizona Brown Rice Pilaf_files/separator.gif"><BR><BR><IMG
src="Arizona Brown Rice Pilaf_files/separator.gif"></TD>
<TD vAlign=top width=100%

```

Figure 59 the source code (c) for the example HTML page

Since the general structure of the recipe has been discussed in solution 1, here I only analyze some special distinctions about the HTML recipe file.

- recipe tile

As we all know, each web page has its own title. For the web pages that describe the recipes, the title of the web page normally contains the recipe title. The title of the HTML page is encompassed by a pair of HTML tag, such as:

<TITLE>RecipeCenter: Arizona Brown Rice Pilaf</TITLE>

Therefore, we can get the title of the recipe by extracting the title of the web page.

- recipe category

Since there are no special tags in the HTML files indicating the recipe category, the category extraction algorithm here still uses the one adopted in solution 1.

- recipe ingredient

The recipe ingredients can be written in any place of the HTML page, so we can't extract them according to the special HTML tags. Still, in most the cases, the recipe ingredients are written in one single paragraph which consists of many ingredient description lines. As analyzed in solution 1, in the paragraph that describes the recipe ingredients, the first word of each line usually is numerical. The normal, completed ingredient description line contains quantity, unit and ingredient descriptions.

3.2 Design and Specification

The import process can be separated into three parts: Parsing the HTML Document, Extraction and Inserting the Recipe into the Database.

3.2.1 Parsing the HTML Document

First of all, the program should extract the text parts from the HTML file (this process can be called parsing HTML). Then a plain text recipe file which doesn't contain any HTML tags will be generated and the program will extract the recipe data from this text recipe file.

Obviously, it is much more difficult to parse a loose defined language like HTML than a clearly defined language which doesn't allow any ambiguous spelling and syntax errors, such as Java and XML. Fortunately, the `javax.swing.text.HTML` and `javax.swing.text.HTML.parser` packages include classes which can do part of the hard work.

3.2.1.1 HTMLToolkit.Parser

The inner class `javax.swing.HTML.HTMLToolkit.Parser` is one of the key classes for parsing HTML file. The instance of this class reads the HTML file from a `Reader` (the I/O class). It looks for the start tags, end tags, empty tags, text and comments in the HTML file. Every time the parser class meets one of those five items, it invokes the relevant callback method in the `javax.swing.text.HTML.HTMLToolkit.ParserCallback` class. The way to connect the instance of the parser class with the instance of `ParserCallback` class is to call the public method provided by the parser class:

```
public void parse(Reader in, HTMLToolkit.ParserCallback callback,  
boolean ignoreCharacterSet) throws IOException
```

The boolean argument `ignoreCharacterSet` is used to enable or disable the throw of `ChangedCharSetException`, which occurs in the case that a `META` tag was found.

3.2.1.2 HTMLToolkit.ParserCallback

The `javax.swing.text.HTML.HTMLToolkit.ParserCallback` class takes charge of how to parse a HTML file. It provides six callback methods:

```
public void handleText(char[] text, int position)  
public void handleComment(char[] text, int position)  
public void handleStartTag(HTML.Tag tag,MutableAttributeSet attributes, int position)  
public void handleEndTag(HTML.Tag tag, int position)  
public void handleSimpleTag(HTML.Tag tag,MutableAttributeSet attributes, int position)  
public void handleError(String errorMessage, int position)
```

These methods need to be overridden and put into specific source code to handle the corresponding parsing work. E.g. find out the care tag set and process the text in between in a special way.

3.2.2 Extraction

3.2.2.1 Extraction of the recipe title

As analyzed above, the title of the HTML document is always encompassed by a pair of HTML tags: <TITLE> and </TITLE>, and most of titles of the HTML recipe documents contain the recipe titles, therefore the program should extract the title of the HTML document according to the HTML tag: 'TITLE'.

Through analyzing plenty of the HTML recipe pages, I found out that most of the HTML recipe pages have put the title of the recipes into their title. In some cases the title of the recipe document is just the title of the recipe, and in some other situation the title of the HTML document contains not only the title of the recipe but also the information like the web site or recipe category. For the latter case, the extra information and the title of the recipe are separated by some special signs, such as such as colon ':' or bar '|' or line '-'.

For example:

```
<TITLE>iChef.com Free Recipes - Soups and Stews: Albuquerque Corn Soup</TITLE>
```

Therefore, in order to extract the title of the recipe much precisely, the program should only extract the string after those special signs from the title of the HTML document.

3.2.2.2 Extraction of the recipe ingredients paragraph

The program checks each line of the HTML recipe file to see whether it meets the following conditions:

- The first word of the line is numerical.
- This line contains one unit word.
- This line contains at most 7 words.

A piece of the common, completed ingredient description usually meets above conditions. The third condition is used for excluding the exception that some irrelevant lines which may meet the first two conditions but doesn't belong to the recipe ingredient (Based on my experience, normally the ingredient description lines contain less than 8 words). It is reasonable to make an assumption that any line which can meet those three conditions is considered as one piece of the recipe ingredient description and any paragraph of the recipe ingredients description must contain at least such a line. Therefore when a line meets those three conditions is found, then the paragraph where the line belongs to will be considered as the paragraph of the recipe ingredients description.

3.2.2.3 Extraction of the recipe direction

Obviously, most of the recipe directions contain the item of the recipe ingredients. For example, if the recipe ingredient contains 'milk', then the recipe direction must contain this word as well. Therefore a complementary table will be established in the database to store the items of the recipe ingredients like: water, milk, beef etc.

When the program extracts the quantity, unit and ingredients from the ingredients description line, the ingredient word(s) will be extracted and put into that complementary table. This is done as follows:

Check every words of the ingredient description line from the left to the right to see whether it is numerical or unit word. If it is numerical or unit word, the program will ignore this word and continue to check the next word. Otherwise the program should check whether this word belongs to the words group such as article (i.e. the, a), conjunction (i.e. and) and adjectives (i.e. such). If this word doesn't belong to those words group, it will be extracted as the recipe ingredient word and put into the complementary table (named TempMaterial).

As mentioned before, in most of the cases, the recipe direction is written after the recipe ingredient. After extracting the ingredient word(s) and filling them into the complementary table--TempMaterial, the program will continually check the rest parts of the recipe HTML file line by line. If there is any word in the line can be matched by a word listed in the TempMaterial table, this line will be extracted as part of the recipe direction.

However, some exceptions should be considered: the recipe direction may not contain any recipe ingredients. For example:

Grease a large cookie sheet. Set aside.

For extracting this kind of recipe direction, another complementary table (named CommonWord) should be created in the database in advance for storing plenty of typical verbs which can represent recipe direction, such as: pour, preheat, bake, grease, oil, butter, stir, mix, fry etc.

If there isn't any word in the TempMaterial table found in this line, the program will check whether there is any word in the CommonWord table found in this line. If there is, this line is considered as part of the recipe direction and the program will continue to check the next line. The extraction program of recipe direction should stop when it encounters a line which contains neither any word in the TempMaterial table nor any word in the CommonWord table.

3.2.2.4 Extraction of the recipe category

The way to extract the recipe category in solution 2 is the same as the way used in solution 1, so please refer to solution 1.

3.2.2.5 Validation of Recipe File

If there isn't any recipe ingredients paragraph extracted from the HTML recipe file, or there isn't any recipe direction can be found after the extraction of the recipe ingredients paragraph, the recipe file will be considered as invalid recipe file.

3.2.3 Inserting the Recipe into the Database

3.2.3.1 Insert the recipe title, category and direction into the Recipe table

After extracting the recipe title, category and direction, the program should convert these values into the proper data types which are consistent with the data types defined in the database, and then insert them into the recipe table in the database. The process of inserting data into the database is the same as the process in the solution 1.

3.2.3.2 Inserting the recipe ID and recipe ingredient's quantity, unit, and ingredient into the Ingredient table

After extracting the ingredient description paragraph, the program will extract the quantity, unit and ingredient these three parts. The algorithm of extracting the quantity, unit and ingredient is similar with the algorithm used in solution 1.

3.3 Implementation

In solution 2, the key improvement of the system is to re-write the extraction model part: **ExtractInformation** class. Here I name the new class as ExtractInformation02 .

3.3.1 The Overview of the Implementation

The UML class diagram is illustrated as below:

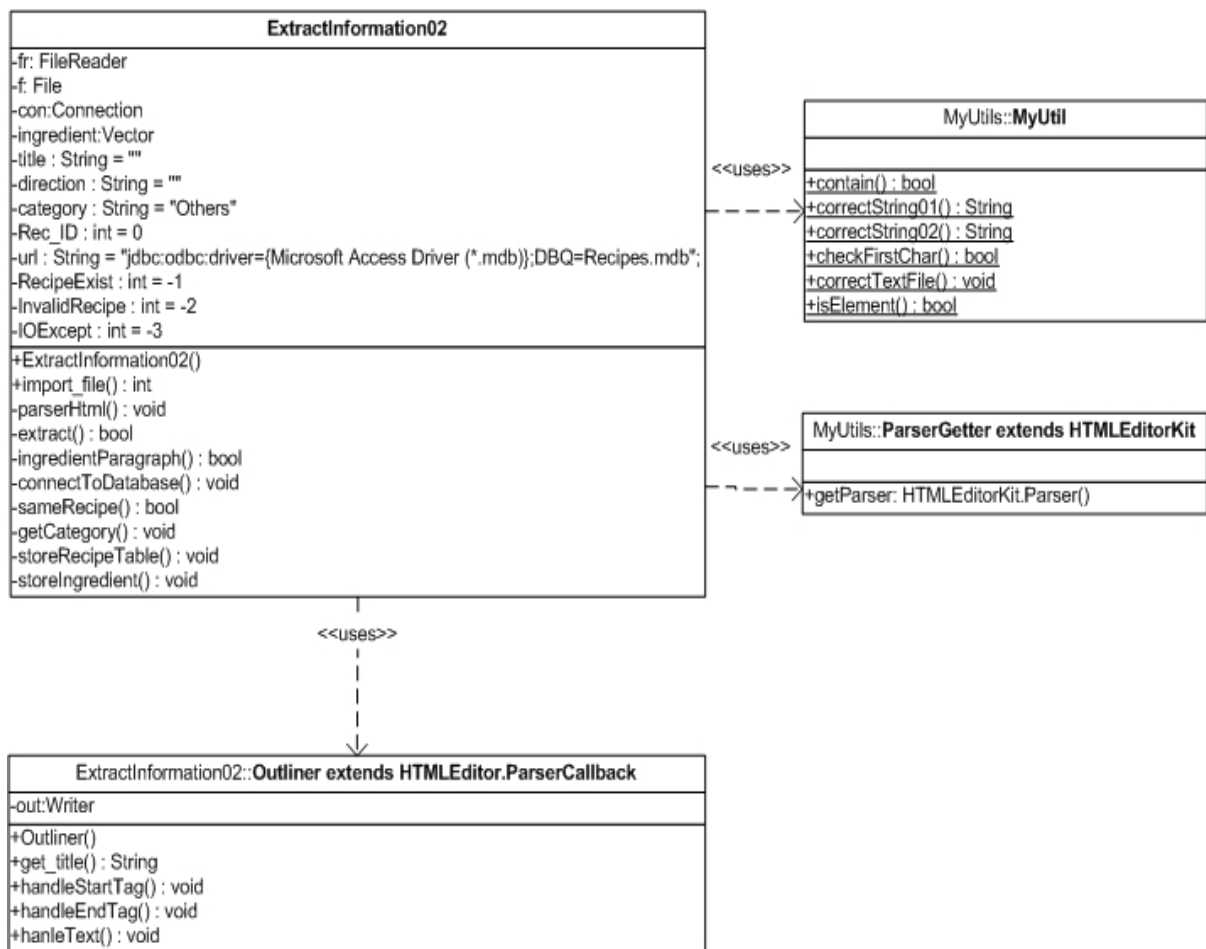


Figure 60 the UML class diagram for the extraction class

Next, I will focus on how the algorithm of extracting and importing the recipe data is implemented.

The procedure of the import process is illustrated by the following flow chart:

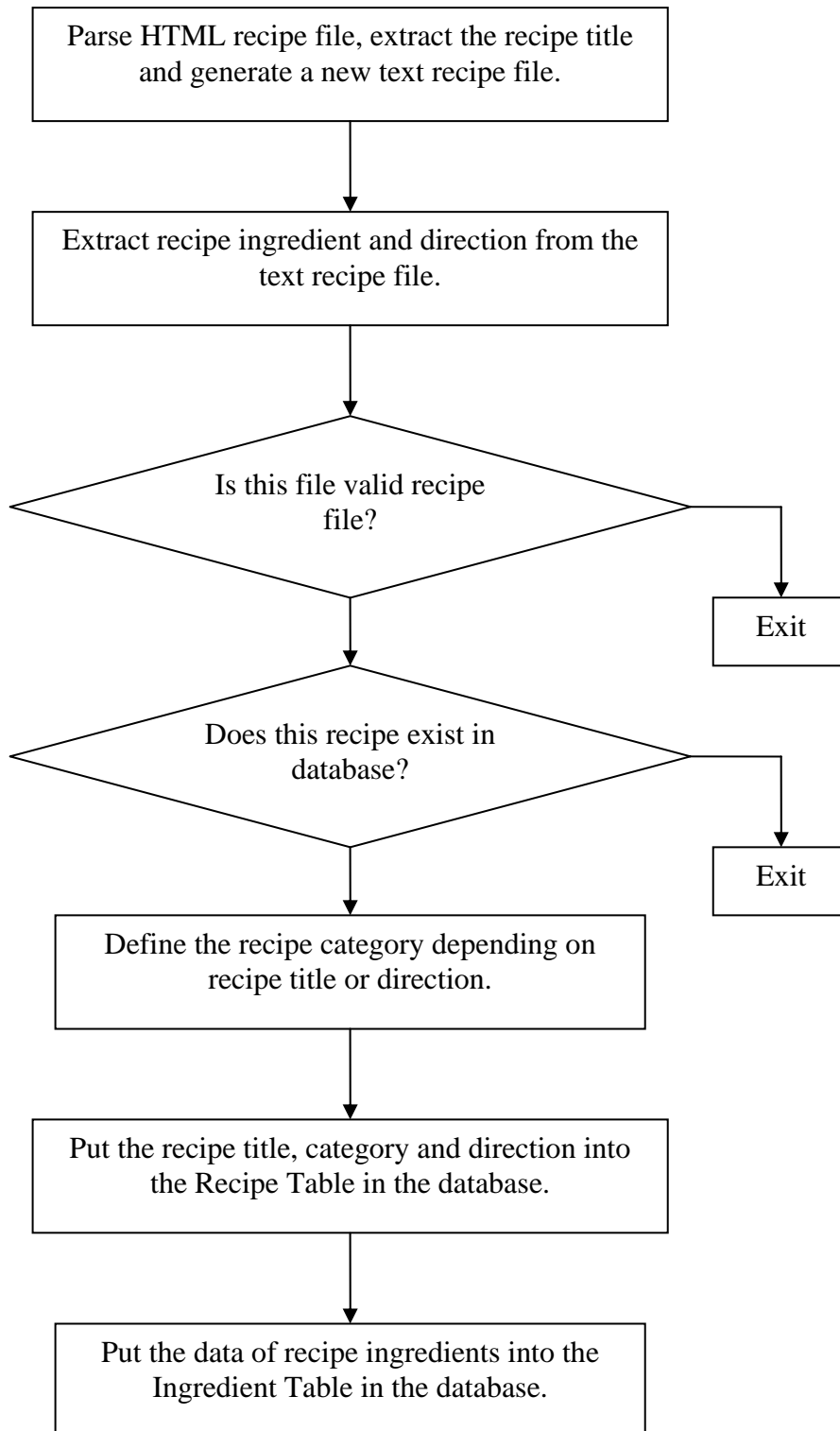


Figure 61 the procedure of the import process

Through the ExtractInformation02 class diagram, we can see the key function models, Parsing the HTML Document, Extraction and Inserting the recipe into the Database, are implemented by the methods in the ExtractionInformation02 class. I will use the flow charts to explain these key models.

3.3.2 The Implementation of Parsing the HTML Document

The implementation process of parsing the HTML document is illustrated by the following flow chart:

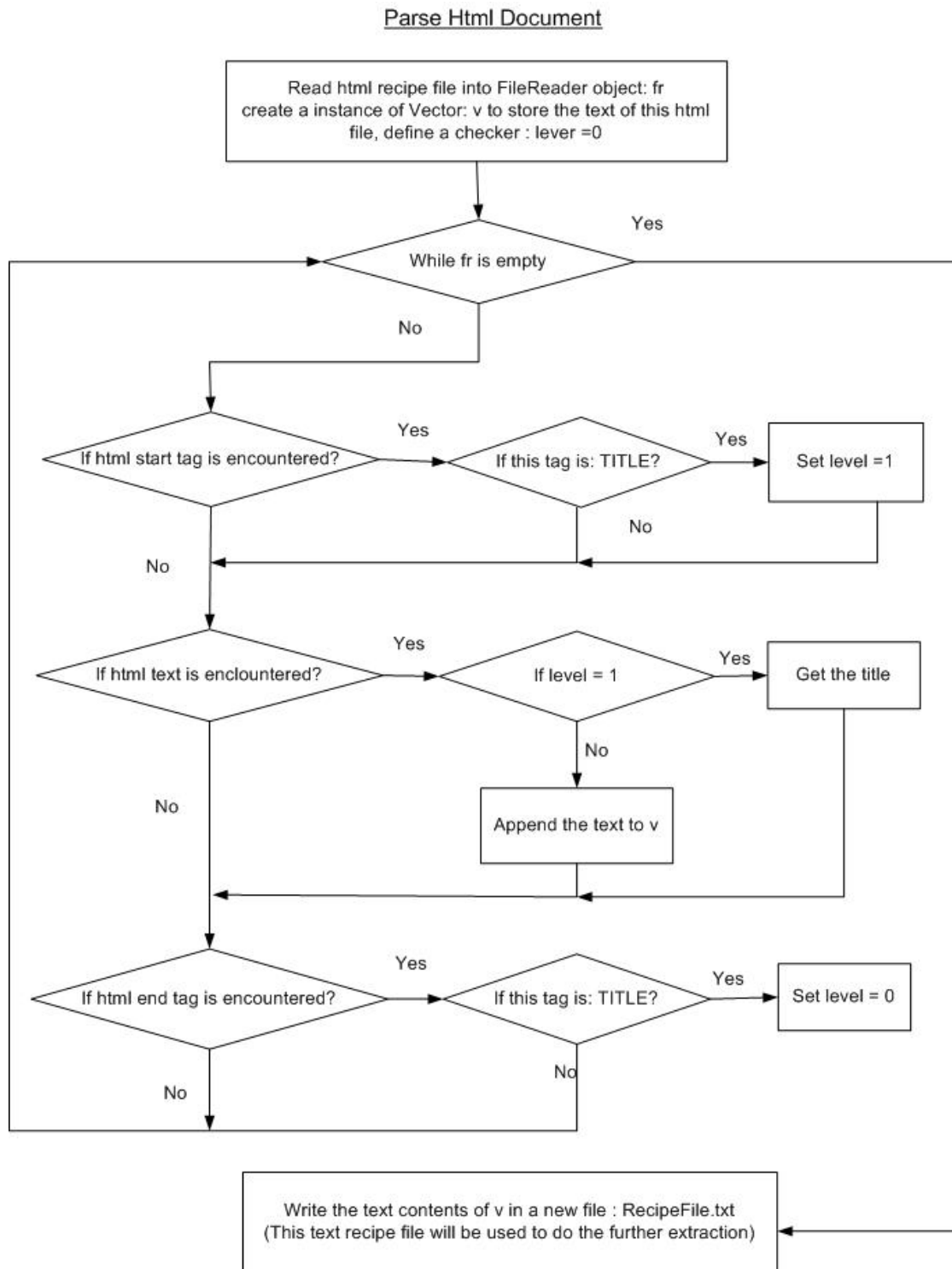


Figure 62 the flow chart for parsing the HTML Document

According to the specification of parsing the HTML documentation, the Outliner class is designed to inherit the abstract class--HTMLToolkit.ParserCallback and override the following three methods:

```
handleStartTag(HTML.Tag tag, MutableAttributeSet attributes, int position)
handleEndTag(HTML.Tag tag, int position)
handleText(char[] int position)
```

When parsing a HTML document, the parse (Reader r, HTMLToolkit.ParserCallback callback, Boolean ignoreCharSet) method will be invoked and the parsing action will be performed by the second argument, the instance of HTMLToolkit.ParserCallback class. When the beginning HTML tag is encountered, the handleStartTag () method will be called; when the closing tag is encountered, the handleEndTag () method will be called and the handleText () will be called when the text is encountered.

During the parsing work, all of the text encountered is stored in an instance of Vector class, named v. After that the program will generate a text file with the content of v for the later extraction. In order to keep the layout of the text file the same as the one of the previous HTML document, here the program should do some special operations when some special tags are encountered.

Currently there are four kinds of HTML tags need to be handled specially in handleStartTag () method.

The first one is the "TITLE" tag. Once the beginning TITLE tag encountered, the program sets a flag variable (belongs to the outliner class), named "level", equal to 1. Thus in the later handleText () method, the program will check this flag. If level == 1, the program knows the text within this pair of tags is the title of the HTML document and will be stored.

The Second one is the beginning tags that indicate a new paragraph needed. These tags are: BODY, TABLE, P and UL. These tags indicate a new paragraph will be generated in the HTML document. Consequently, when these tags are encountered, an empty line should be added into v to insure the layout of the text follow the corresponding structure.

The Third is the tags that represent there should be a new line started. These tags are: BR and LI. Once these two tags encountered, the program sets the flag variable (belongs to outliner class), named "on", equal to 2. Thus in the handleText () method, the program will check this flag. If on ==2, the program will insert the text as a new items into the vector v. Otherwise, the program will treat the text as part of one line which will be appended with the next text.

The fourth one is the "TR" tag which indicates a table row occurs. Some HTML recipe pages use "table" to format the layout of the ingredients, i.e. the amount, unit and ingredient name are separated in each cell within one pair of "TR" tags. And there will be some other pairs of tags, such as font style and size, in between the pair of "TR" tags, i.e. there are HTML tags nested. In order to keep one piece of ingredients in one line, here the program should be able to neglect all the tag pairs except "BR", and "LI" in between, and append the text together and store them into the vector v.

The source code of the Outliner class is shown as below:

```
private class Outliner extends HTMLToolkit.ParserCallback {  
    private Writer out;  
    private String title=null;  
    private int level =0, on = 0;  
    private Vector v=new Vector();  
    public String line = System.getProperty("line.separator", "\r\n"), line1 = "";  
  
    public void handleStartTag(HTML.Tag tag,MutableAttributeSet attributes, int position)  
    {  
        this.level =0;  
        if (tag == HTML.Tag.TITLE) level = 1;  
        if (tag == HTML.Tag.BODY || tag == HTML.Tag.TABLE || tag == HTML.Tag.P ||  
            tag == HTML.Tag.HR || tag == HTML.Tag.DIV)  
            { v.add (this.line1); v.add(this.line); this.line1="";}  
        else if ( tag == HTML.Tag.BR || tag == HTML.Tag.LI) this.on = 1;  
            else if (tag == HTML.Tag.TR) this.line1="";  
        try{out.flush( );}  
        catch (IOException e) {System.err.println(e);} // end method  
  
    public void handleEndTag(HTML.Tag tag, int position)  
    {  
        if (tag == HTML.Tag.TR )  
            { this.on=2; v.add(this.line1); line1="";}  
        else if (tag == HTML.Tag.BODY || tag == HTML.Tag.TABLE ||  
            tag == HTML.Tag.P || tag == HTML.Tag.UL)  
            v.add(this.line);  
  
        //work around bug in the parser that fails to call flush  
        if (tag == HTML.Tag.HTML) this.flush( );  
  
    public void handleText(char[] text, int position)  
    {  
        String s = new String(text);  
        if (this.level ==1) this.title= s;  
        else{if (this.on==1){v.add(s); this.on = 0;}  
            else this.line1 = this.line1+" "+s; } // end else  
        try {out.flush( );} // end try  
        catch (IOException e) {System.err.println(e);} // end method  
  
    public void flush( ) {  
        try {out.flush( );}  
        catch (IOException e) {System.err.println(e);}  
    } // end method  
}
```

Figure 63 the source code of the Outliner class

3.3.3 Extraction

The implementation process of the extraction modules is shown as below:

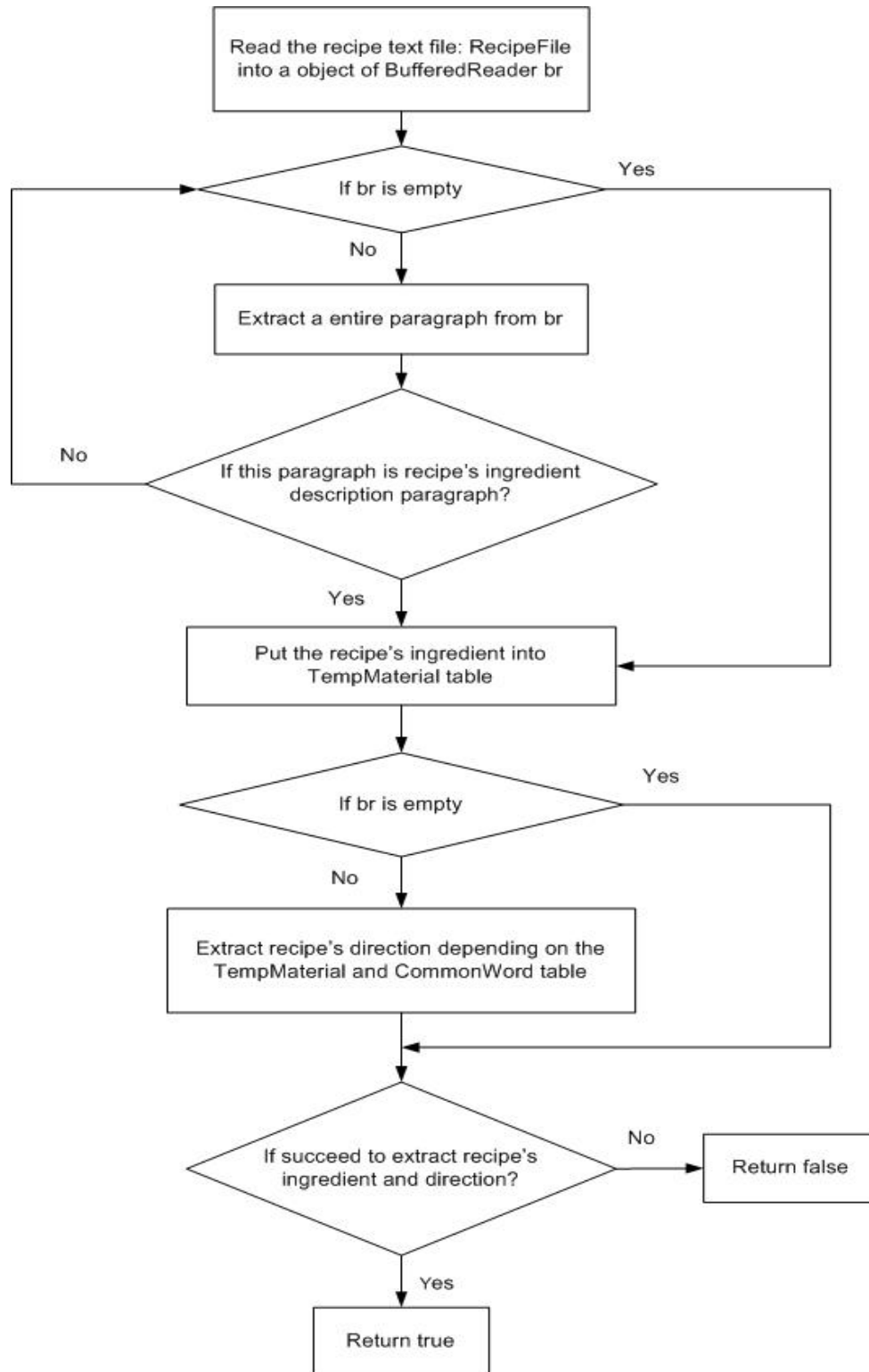


Figure 64 the flow chart for Extraction

The flow chart of the method for extracting an entire paragraph is shown as below:

Extract a entire paragraph

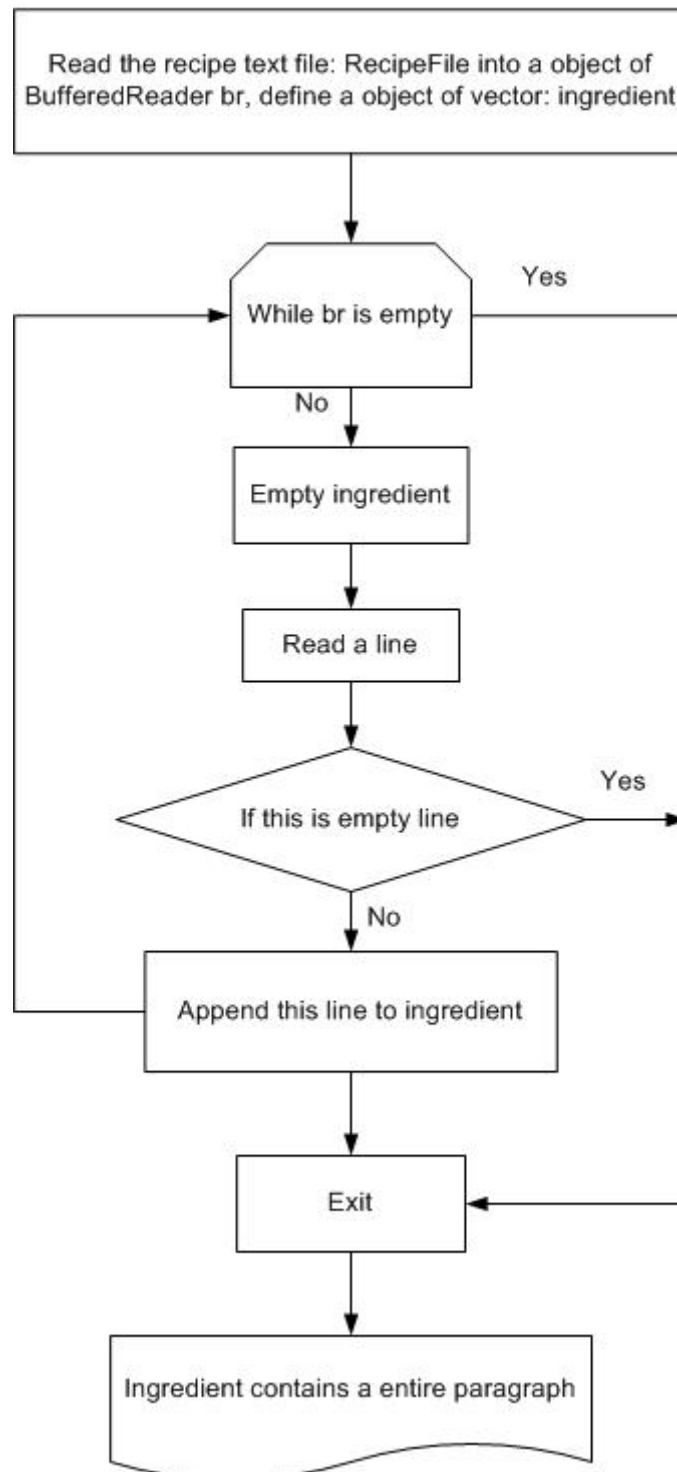


Figure 65 the flow chart for extracting an entire paragraph

The flow chart of the method for checking if the paragraph is the recipe ingredient paragraph is shown as below:

**Check if the paragraph is
recipe's ingredient paragraph**

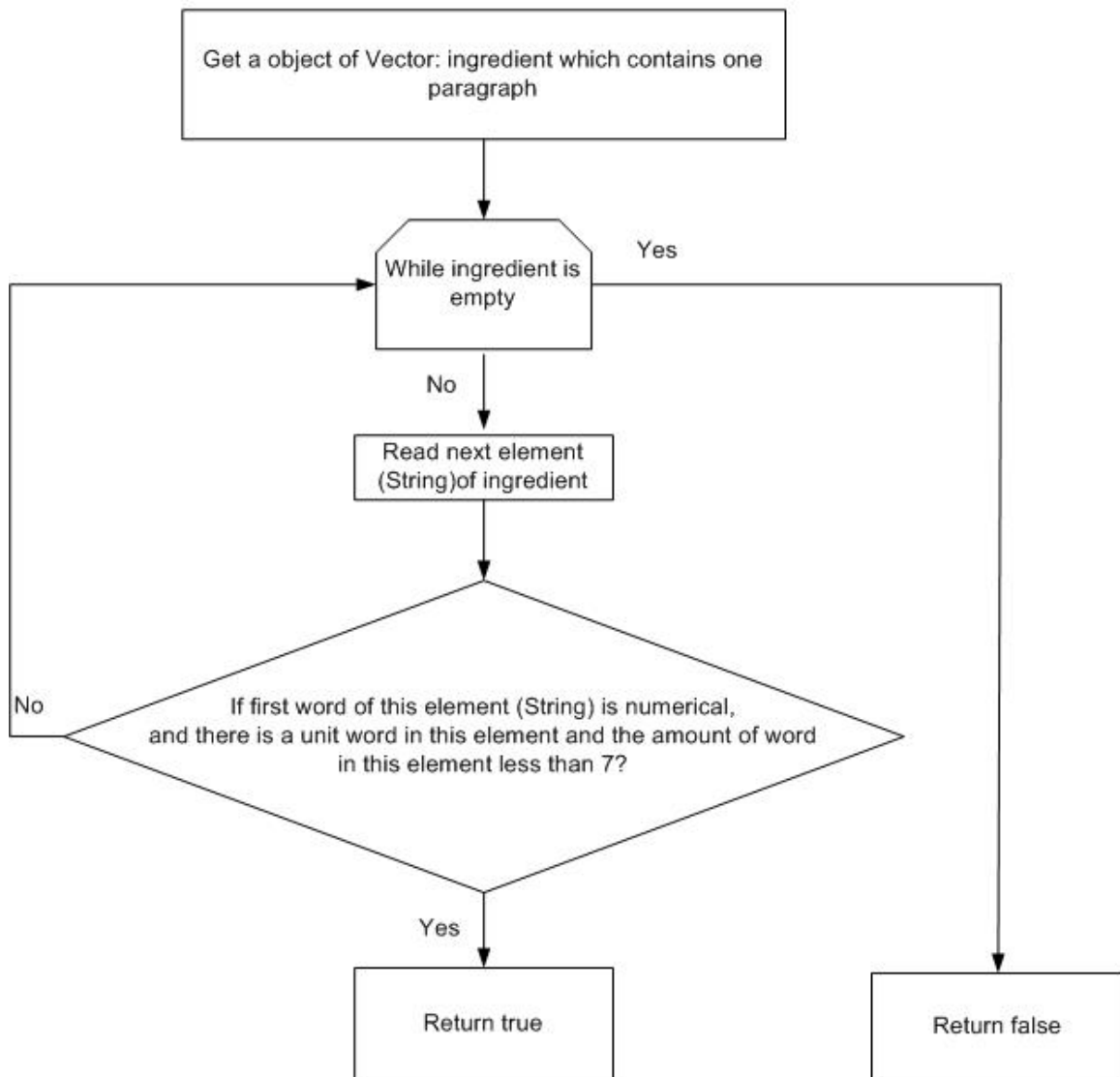


Figure 66 the flow chart for checking the recipe ingredient paragraph

The flow chart of the method for extracting the recipe direction is shown as below:

Extract recipe's direction

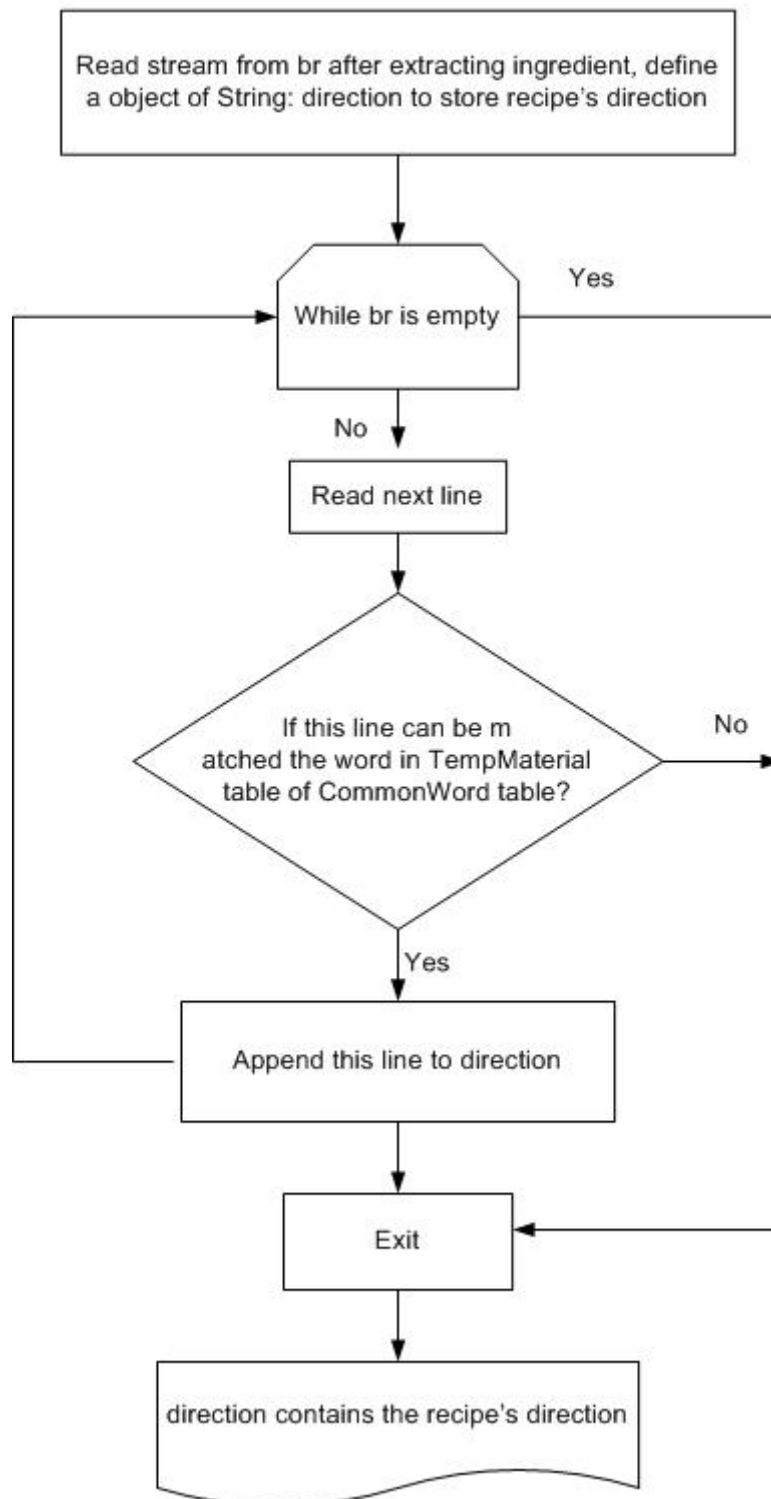


Figure 67 the flow chart for extracting the recipe direction

3.3.4 Inserting the Recipe into the Database

The flow chart of the method for inserting the recipe ingredient into the TempMaterial table is shown as below:

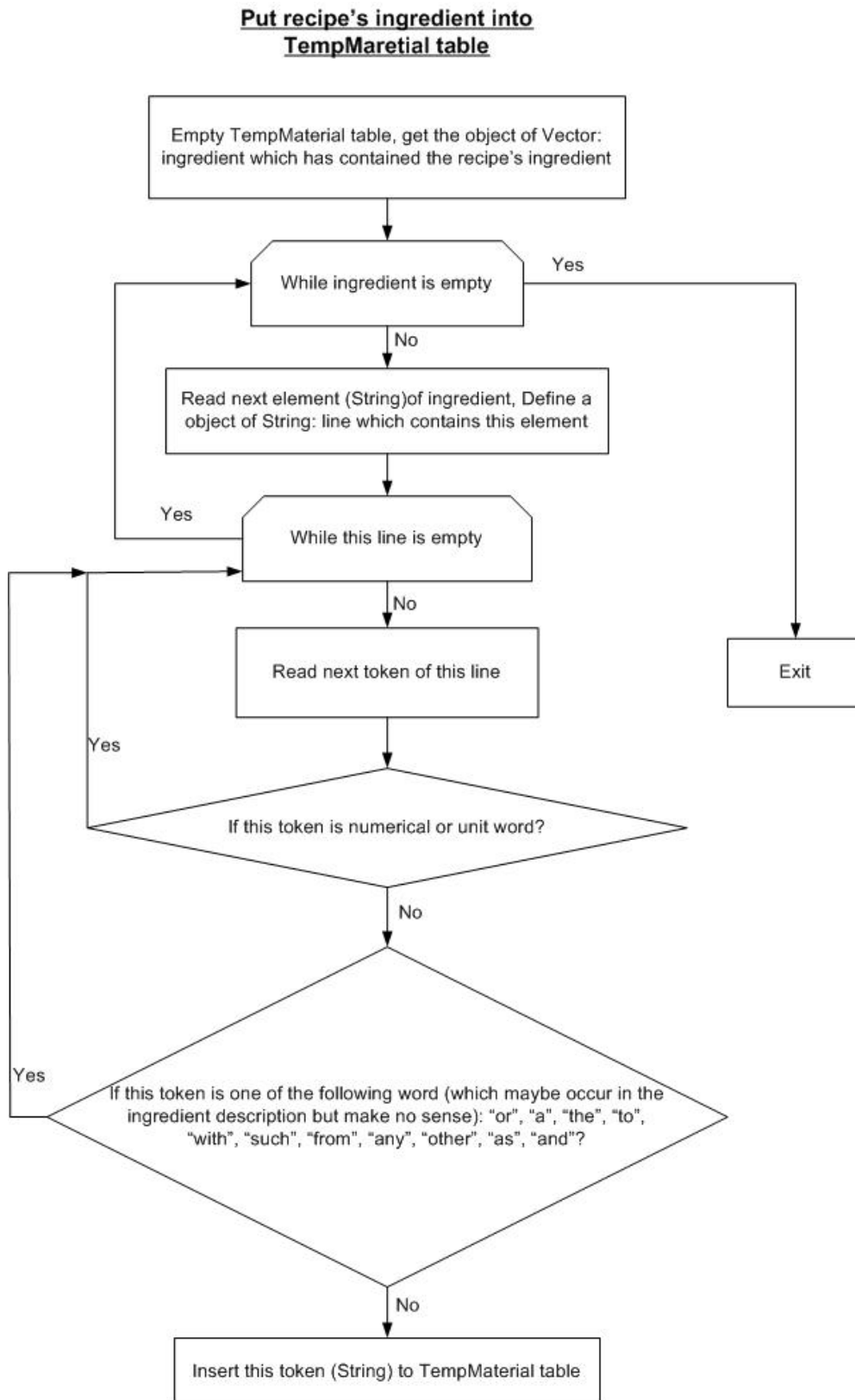


Figure 68 the flow chart for inserting the recipe ingredient into the TempMaterial table

As the algorithm for inserting the recipe data into the database is similar with the one used in solution 1, the flow chart for that method could also refer to the one in solution 1.

3.4 Results and Test

I will test the import function from the following three conditions:

4. The recipe file is invalid.
5. The recipe file is valid and it can be imported successfully.
6. The recipe that will be imported has already existed in the database

3.4.1 Import the Invalid Recipe File

The following web page is the index page of the [CNN](http://www.cnn.com) website, obviously it is not a recipe file.



Figure 69 the example page 'CNN website'

When the Import button is clicked, the error message box is returned:

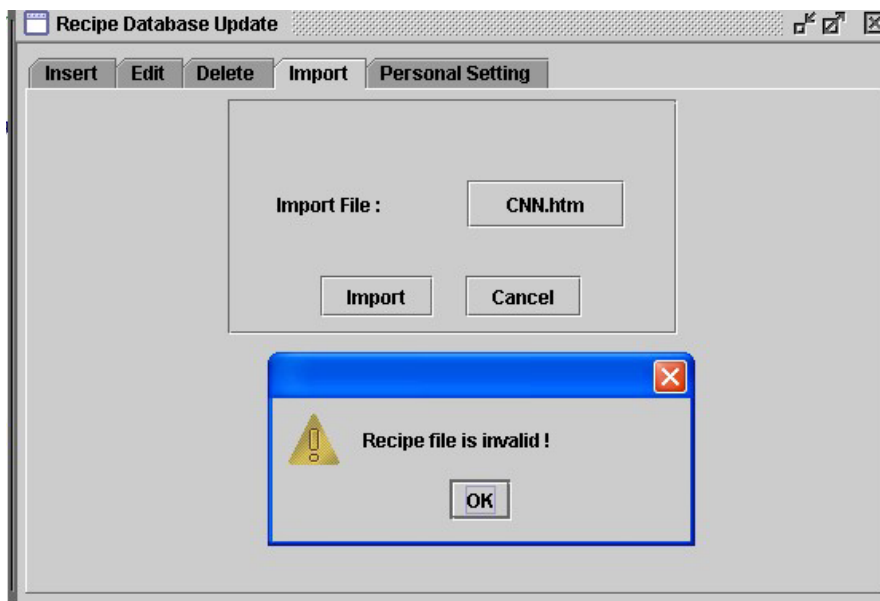


Figure 70 the error message is pop out when dealing with the invalid recipe file

3.4.2 Import the Valid Recipe File

There are two cases for testing the import of the valid recipe file. One is to test on a common valid HTML recipe file; the other is to test on a valid HTML recipe file of which the ingredient description part is written in the table format. Here the tests on both of these two cases have been done.

3.4.2.1 Test 1

The test 1 is to test on the common valid HTML recipe file. The title of the recipe is is 'Brocco Taco Salad', as shown in the following figure.

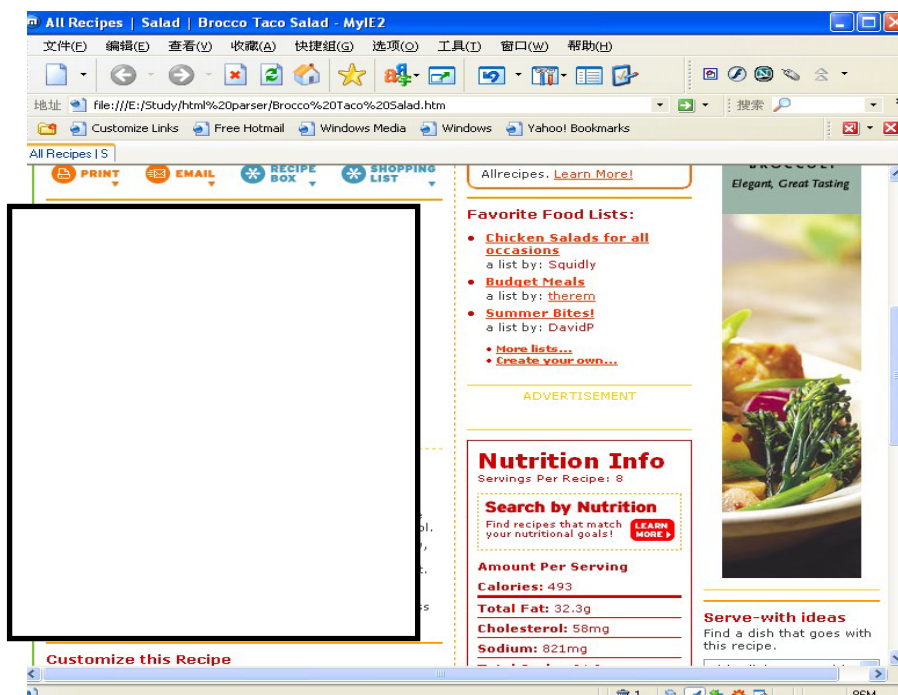
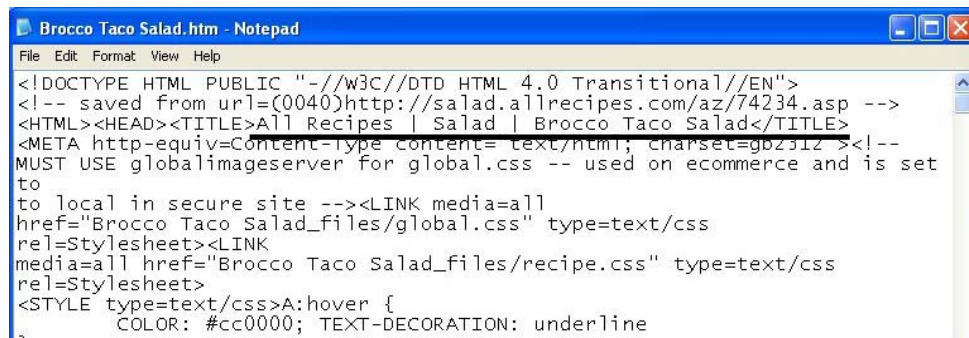


Figure 71 the example HTML page 'Brocco Taco Salad'

The source code for the example HTML page is:

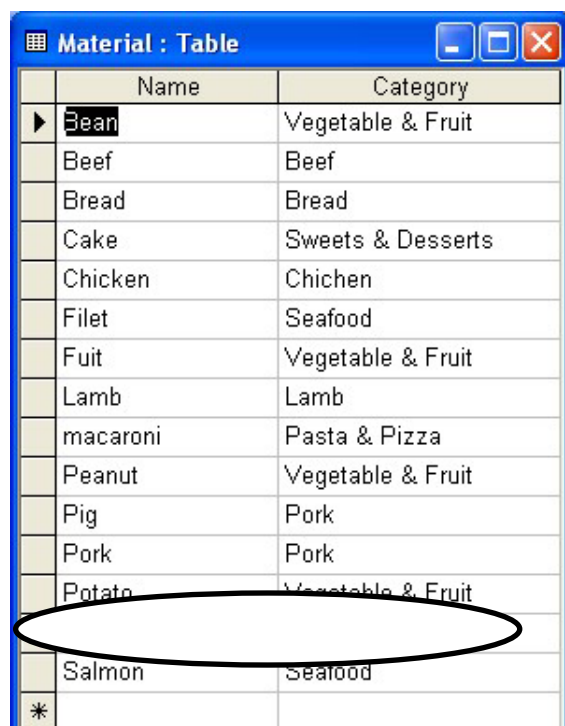


```
Brocco Taco Salad.htm - Notepad
File Edit Format View Help
<!DOCTYPE HTML PUBLIC "-//w3c//DTD HTML 4.0 Transitional//EN">
<!-- saved from url=(0040)http://salad.allrecipes.com/az/74234.asp -->
<HTML><HEAD><TITLE>All Recipes | Salad | Brocco Taco Salad</TITLE>
<META http-equiv=Content-type content= text/html; charset=gb2312 ><!--
MUST USE globalimageserver for global.css -- used on ecommerce and is set
to
to local in secure site --><LINK media=all
href="Brocco Taco Salad_files/global.css" type=text/css
rel=stylesheet><LINK
media=all href="Brocco Taco Salad_files/recipe.css" type=text/css
rel=stylesheet>
<STYLE type=text/css>A:Hover {
COLOR: #cc0000; TEXT-DECORATION: underline
}
```

Figure 72 the source code for the example HTML page 'Brocco Taco Salad'

The title of the HTML file is: All Recipes | Salad | Brocco Taco Salad, the program should extract the recipe title 'Brocco Taco Salad', which is presented after the last sign '|'.

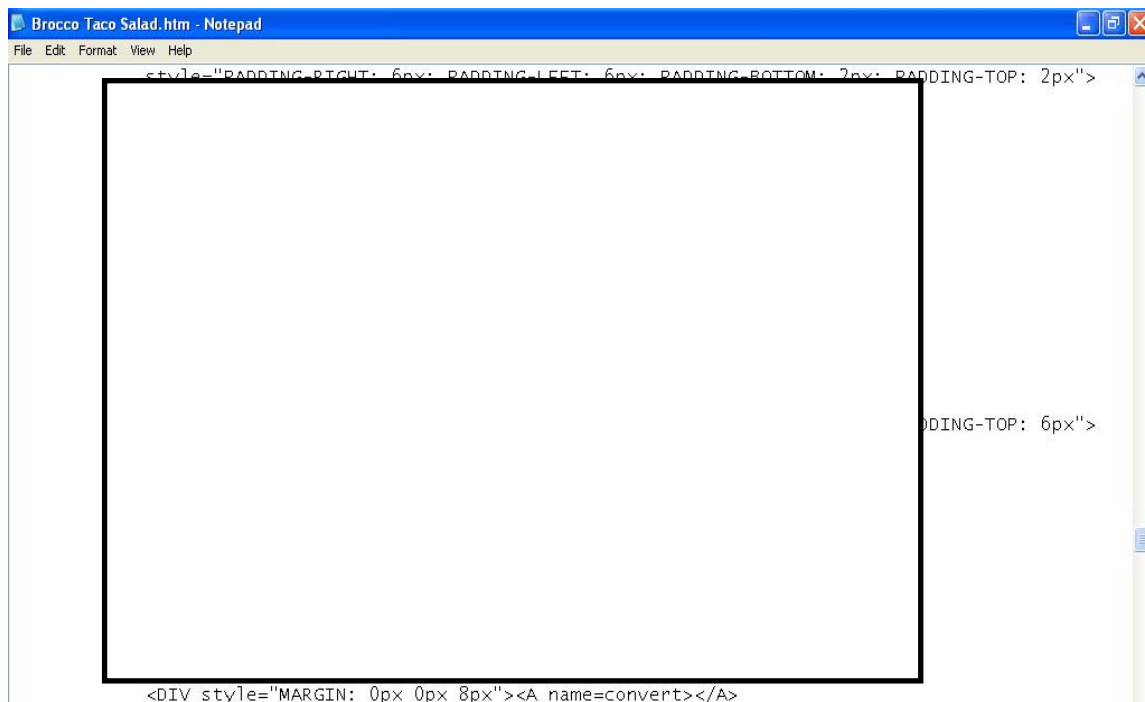
This recipe title contains a material keyword, salad, which can be matched to the one in Material table; the recipe category also can be defined, through the relationship between the Material table and Category table.



	Name	Category
▶	Bean	Vegetable & Fruit
	Beef	Beef
	Bread	Bread
	Cake	Sweets & Desserts
	Chicken	Chicken
	Filet	Seafood
	Fruit	Vegetable & Fruit
	Lamb	Lamb
	macaroni	Pasta & Pizza
	Peanut	Vegetable & Fruit
	Pig	Pork
	Pork	Pork
	Potato	Vegetable & Fruit
	Salmon	Seafood
*		

Figure 73 the example material table

The HTML source code of the recipe ingredient and direction parts is shown as below:

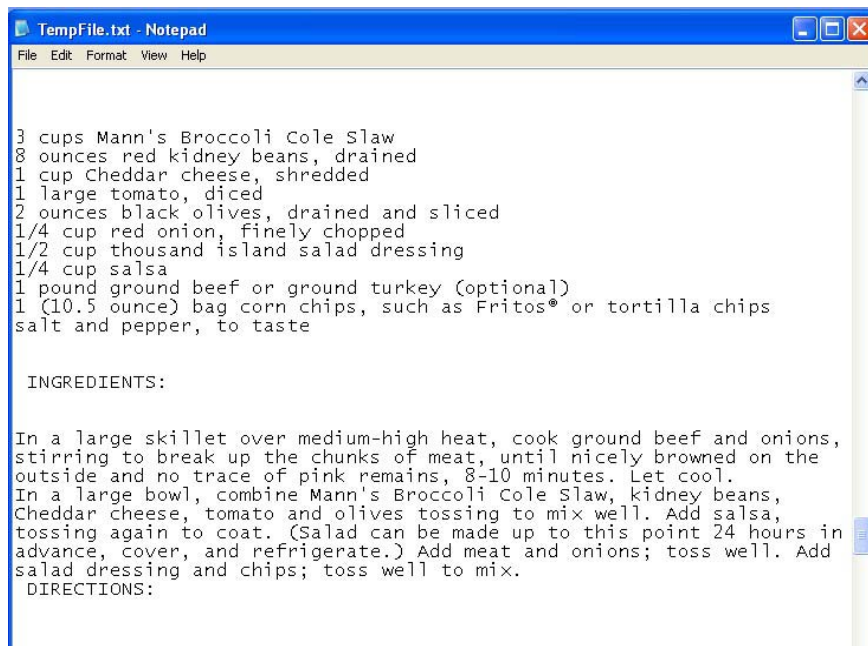


```
style="MARGIN: 0px 0px 8px"><A name=convert></A>
```

Figure 74 shows a Notepad window titled "Brocco Taco Salad.htm - Notepad". The window displays HTML source code. A large rectangular area is highlighted with a black border, representing the recipe ingredient and direction parts. The code includes a style attribute: `style="MARGIN: 0px 0px 8px">`. The text "DDING-TOP: 6px">" is visible on the right side of the highlighted area.

Figure 74 the source code of the recipe ingredient and direction parts

The recipe ingredient description is written in list format. As mentioned above, the HTML 'LI' tag can be handled specially during parsing the HTML document, so the layout of the recipe ingredient paragraph is kept in the newly generated text recipe file. It is show as below:



```
TempFile.txt - Notepad
File Edit Format View Help

3 cups Mann's Broccoli Cole Slaw
8 ounces red kidney beans, drained
1 cup Cheddar cheese, shredded
1 large tomato, diced
2 ounces black olives, drained and sliced
1/4 cup red onion, finely chopped
1/2 cup thousand island salad dressing
1/4 cup salsa
1 pound ground beef or ground turkey (optional)
1 (10.5 ounce) bag corn chips, such as Fritos® or tortilla chips
salt and pepper, to taste

INGREDIENTS:

In a large skillet over medium-high heat, cook ground beef and onions,
stirring to break up the chunks of meat, until nicely browned on the
outside and no trace of pink remains, 8-10 minutes. Let cool.
In a large bowl, combine Mann's Broccoli Cole Slaw, kidney beans,
Cheddar cheese, tomato and olives tossing to mix well. Add salsa,
tossing again to coat. (Salad can be made up to this point 24 hours in
advance, cover, and refrigerate.) Add meat and onions; toss well. Add
salad dressing and chips; toss well to mix.

DIRECTIONS:
```

Figure 75 shows a Notepad window titled "TempFile.txt - Notepad". The window displays the newly generated text recipe file. The text is as follows:

3 cups Mann's Broccoli Cole Slaw
8 ounces red kidney beans, drained
1 cup Cheddar cheese, shredded
1 large tomato, diced
2 ounces black olives, drained and sliced
1/4 cup red onion, finely chopped
1/2 cup thousand island salad dressing
1/4 cup salsa
1 pound ground beef or ground turkey (optional)
1 (10.5 ounce) bag corn chips, such as Fritos® or tortilla chips
salt and pepper, to taste

INGREDIENTS:

In a large skillet over medium-high heat, cook ground beef and onions, stirring to break up the chunks of meat, until nicely browned on the outside and no trace of pink remains, 8-10 minutes. Let cool. In a large bowl, combine Mann's Broccoli Cole Slaw, kidney beans, Cheddar cheese, tomato and olives tossing to mix well. Add salsa, tossing again to coat. (Salad can be made up to this point 24 hours in advance, cover, and refrigerate.) Add meat and onions; toss well. Add salad dressing and chips; toss well to mix.

DIRECTIONS:

Figure 75 the newl generated txt paragraph

The recipe title is: 'Brocco Taco Salad' and its category should be Vegetable & Fruit.

The imported recipe display result is shown as below:

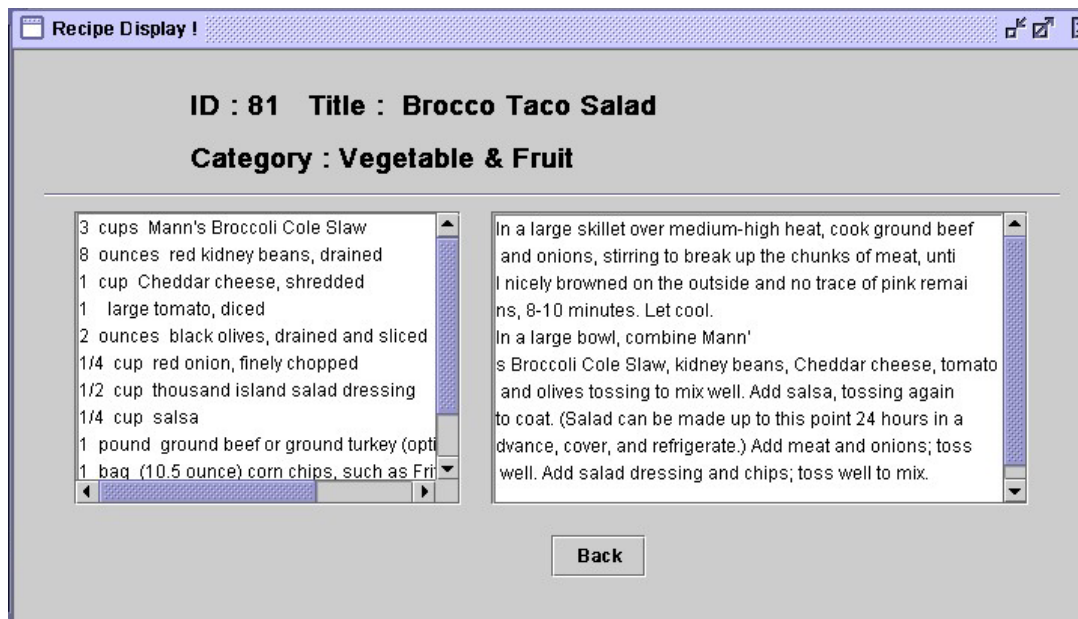


Figure 76 the display of the imported recipe

3.4.2.2 Test 02

Here is another valid HTML recipe file (as shown in the following figure), of which the recipe ingredients are written in the 'table' format.

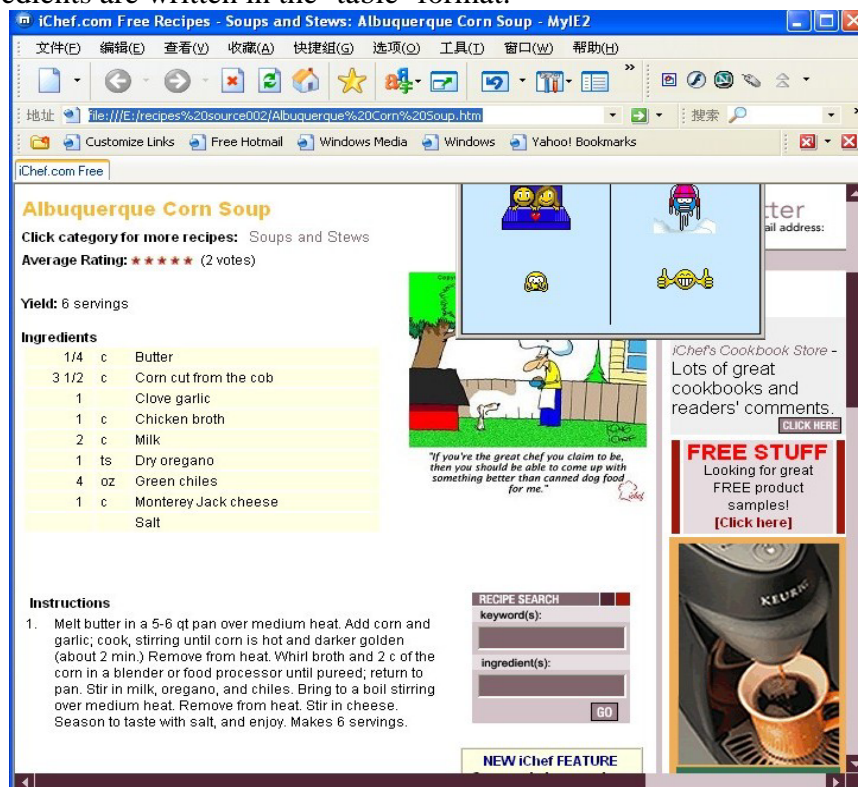


Figure 77 the HTML page, of which the recipe ingredients are written in the table format

The ingredient description is written in the table format.

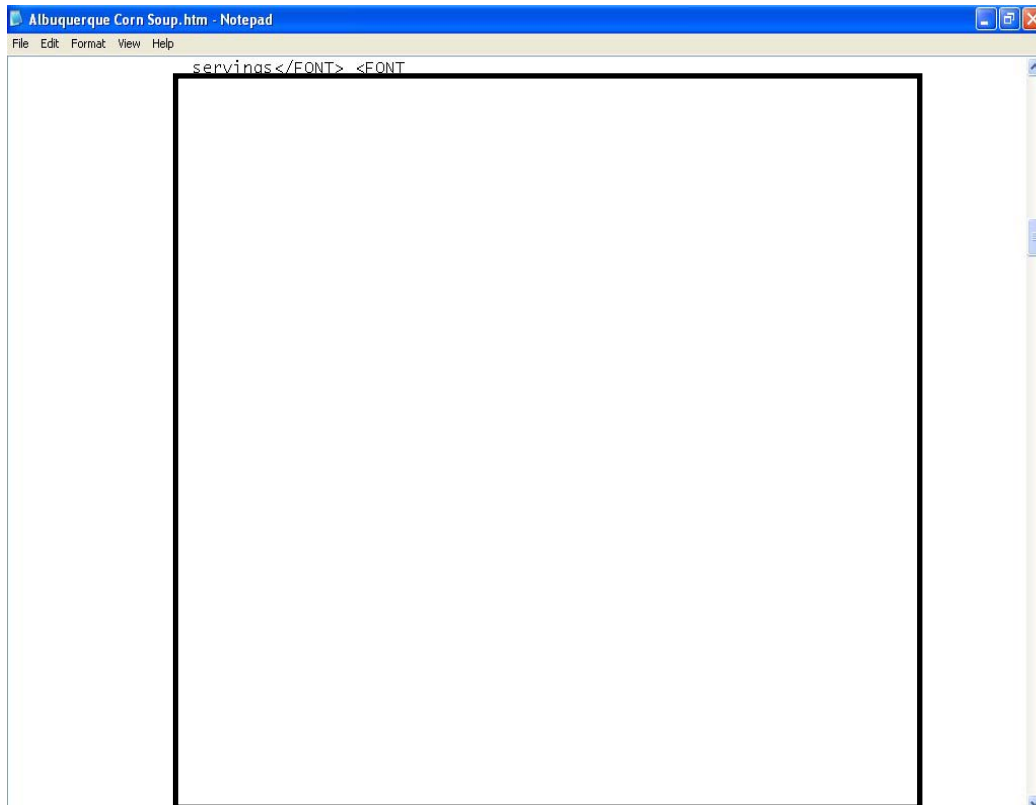


Figure 78 the HTML source code of the ingredient description

We can find that the quantity, unit and ingredient are separated in each cell within one pair of “TR” tags. And there are some other pairs of tags, such as font style and size, in between the pair of “TR” tags. Through handling the ‘TR’ tag in special way, each piece of the ingredient description is kept in one line (Otherwise, the quantity, unit and ingredient will be separated in different lines). The new generated recipe text file is shown as below:

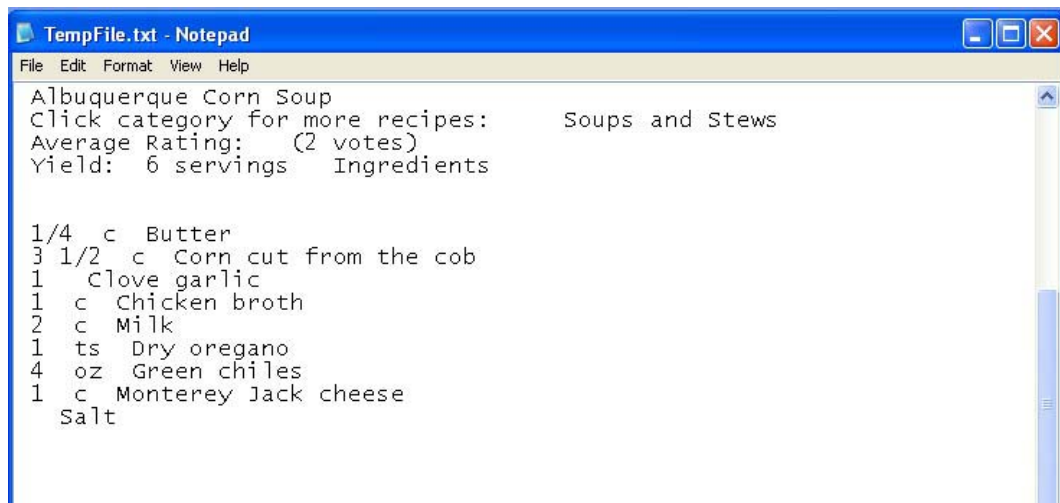


Figure 79 the extracted txt paragraph of the ingredient description

The display result of the imported recipe is:

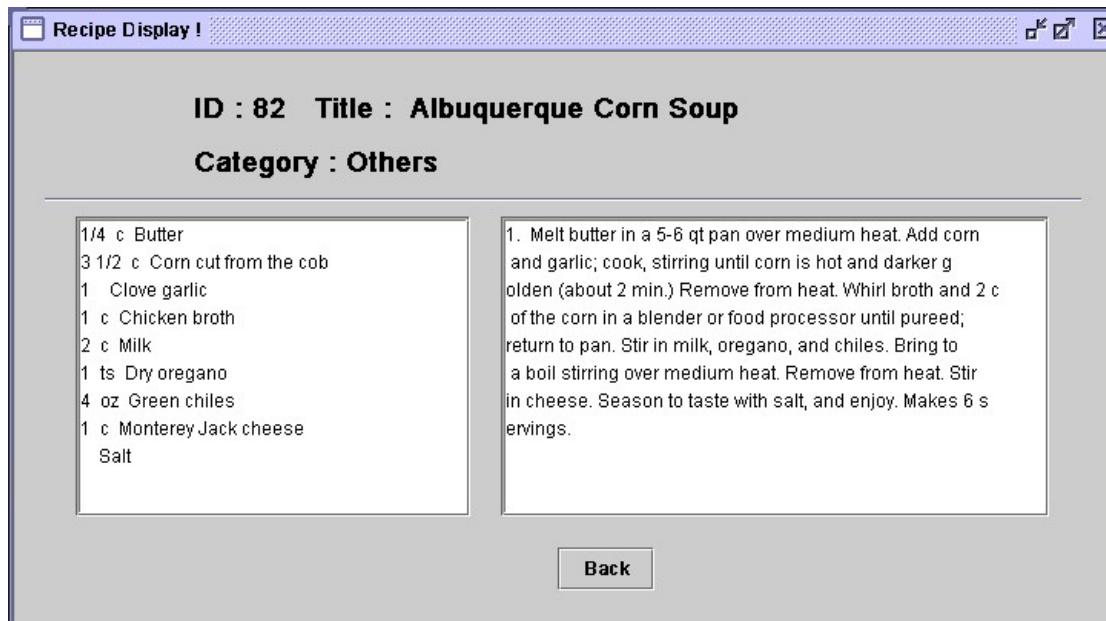


Figure 80 the display result of the imported recipe (test 2)

3.5 Summary

The solution 2 is a kind of upgrade version for solution 1. Compared to solution 1, the major improvement of solution 2 is to introduce a new import function. A new algorithm is adopted to make the system capable of importing the external HTML recipe files directly.

Generally speaking, solution 2 indeed conquers some limitations existing in solution 1 and enhances the flexibility and compatibility of the system.

The process of the importing is implemented by the following three steps:

1. Parse the HTML document into a text file called: RecipeFile.txt,
2. Extract the recipe data from RecipeFile.txt
3. Insert the extracted recipe data into the database.

In solution 2, the system can recognize and extract the recipe title by locating the pair of HTML tags: <Title> and </Title>. Compared to the title extraction method used in solution 1, this way is more intelligent and can increase the accuracy.

The extraction of the recipe ingredients follows the rules that: there must be at least one standard ingredient descriptive item (line) which consists of quantity, unit and ingredient description in the recipe ingredient description paragraph, for example:
1 cup hot water

When a standard ingredient descriptive item (line) is found out, the program will treat the paragraph which this item (line) belong to as the paragraph describing the recipe ingredient.

For the direction part, the system extracts it according to the recipe feature ingredients (such as beef, pork, flour and etc.) and some typical words which may appear in the direction such as mix, stir, grease and etc. By using this method, the extraction of the direction part is elaborated to the line based level. Compared to the 'paragraph' based extraction used in solution 1, this method make the system more semantic oriented and increase the accuracy.

Although solution 2 has been able to import more kinds of recipe files and achieve higher accuracy on the recipe extraction than solution 1 does, there are still some weaknesses in it.

As mentioned above, the HTML specification is loosely structured and there are many different ways (by using different tag pairs) to describe the content but display in the same way. Therefore, in some cases when the HTML recipe files are encoded in some informal or uncommon ways, the system cannot recognize and import them.

In addition, due to the time reason, right now the system can only handle the HTML recipe file with a single frame. The development of handling the HTML recipe file with multiple frames inside would be part of the future work.

4. Conclusion

In this report, I introduced the development procedure of a recipe system, which includes four major steps: system analysis, design, implementation, and test. The development process strictly follows the principle of the software engineering.

There are two solutions implemented in this project. The main difference between solution 1 and solution 2 is the type of the recipe files and the import algorithm. In solution 1, the program can only import the external recipe file which is saved in *.txt format and must contain some special signature key words such as 'ingredient' and 'direction'; In solution 2, the system can automatically import the HTML recipe file which is downloaded from the web site. Moreover, the HTML recipe file doesn't need to contain the special signature key words, which makes the system be able to handle more kinds of recipe files.

The improvement from solution 1 to solution 2 actually reflects the effects of the semantic oriented programming. In solution 1, the algorithm of the recipe extraction mechanically relies on some artificial marks, which are unstable and only existed in some kinds of recipe files. Solution 2 addressed on the discipline of the recipe content and tried to find out the semantic essence of it. Therefore, compared to the former, solution 2 provides a more general method, which can handle most of the recipe files.

In a word, the objectives of this project have been fully achieved.

4.1 Future Work

Strictly speaking, the recipe system is still in the prototype version. There are lots of supplementary works to do. In order to increase the accuracy of the recipe extraction and make the system more flexible and compatible, the following aspects should be addressed as the future work:

Firstly, the system may add as more as possible ingredient names into the Material table in order to make the program be able to accurately recognize the categories for more recipes. In addition, such a completed Material table can be used in the extraction of the ingredients as well, which in turn increase the accuracy of the extraction.

Secondly, the system can add a new table in the database to store the descriptive words which are served as the adjectives of the ingredients. This will accelerate the positioning of the ingredients and facilitate the ingredients extraction.

Thirdly, as mentioned before, the system should be able to handle the HTML recipe files with multiple frames in side. The main challenge behind that is how to locate the recipe frame.

4.2 Personal Conclusion

Through this project, I have gained lots of practical experience and knowledge about the object-oriented programming, the design of the relational database, and HTML

specification. This helps me well understand the development procedure of the software engineering and accumulate rich experience for doing further R&D work in computer science field.

In addition, my ability on how to seek, collect, analyze and utilize the information was enhanced from this project and I better understand the importance of communicating and sharing ideas with others.

In a word, as an intending engineer of the computer science, I should say this project definitely gave me the most valuable practical trainings on both the specific techniques and the methodology of thinking.

Reference

- [1] Gonghe Chen & Hanxin Wang & Linrui Liu, '*The Basic of Database and Access Application Tutorial*', Higher Education Publishing Company Beijing, 2003
- [2] Paul Fischer, '*Introduction to Graphics with JAVA-Swing using the Model-View-Control concept*', IMM, Denmark Technical University Version 1.2, October 2003
- [4] O'Reilly, '*Java Networking Programming*', 2nd Edition
- [5] Thomas Connolly & Carolyn Begg, '*Database Systems, Third Edition*', ADDISON WESLEY, 2002
- [6] Harvery Deitel, Paul Deitel, '*Java How to Program*', *Fifth Edition*, Pearson Prentice Hall, 2003
- [7] Walter Savitch, '*JAVA, An Introduction to Computer Science & Programming*', International Edition, Pearson Prentice Hall, 2004
- [8] Jeffery D. Ullman & Jennifer Widom, '*A First Course in Database System*', Pearson Prentice Hall, 1997
- [9] Ian Sommerville, '*Software Engineering*', Sixth Edition, ADDISON WESLEY, 2001
- [10] H. Garcia-molina, Jeffery D. Ullman & Jennifer Widom, '*Database Systems, The Complete Book*', Pearson Prentice Hall, 2002
- [11] Bob Villareal, '*Access 2002 Programming by Example*', Queen, 2002
- [12] Philip M. Lewis, Arthur Bernstein & Michael Kifer, '*Databases and Transaction Processing, An application-Oriented Approach*', ADDISON WESLEY, 2002
- [13] Khawar Zaman Ahmed & Cary E. Umrysh, '*Developing Enterprise Java Application with J2EE and UML*', Addison Wesley, 2001
- [14] Krik Knoernschid, '*Java Design: Objects, UML, and Process*', Addison Wesley, 2001
- [15] David C. Hay, '*Requirements Analysis: From Business Views to Architecture*' Pearson Prince Hall, 2002

Web Recourses:

- [16] The page of J2SE 1.4.2 API from Sun Java Official web-site
<http://java.sun.com/j2se/1.4.2/docs/api/>
- [17] The page of JAVA Swing Components

<http://java.sun.com/docs/books/tutorial/uiswing/components/components.html>

[18] The page of JDBC Techniques from Sun Java official web-site

<http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html>

[19] The page of Software Engineering

<http://www.sawin.com.cn/satech.asp?class=UML>

[20] The page of Microsoft Office official web-site

<http://office.microsoft.com/home/>

[21] Database Technologies

<http://www.develop.com/us/technology/>

List of Figures

Figure 1 Normal Text Recipe File	14
Figure 2 E-R Diagram.....	25
Figure 3 Use Case Diagram	27
Figure 4The MVC model.....	37
Figure 5 UML Class Diagram.....	38
Figure 6 the procedure of the design and implementation of the database.....	39
Figure 7 the Code for the method 'Insert the Recipe'	47
Figure 8 The code for the method 'edit the recipe'	48
Figure 9 The code for the method 'delete the recipe'	49
Figure 10 The code for the method 'search the recipe'.....	50
Figure 11 Flow01	52
Figure 12 Flow02	53
Figure 13 Flow03-a.....	54
Figure 14 Flow03-b.....	55
Figure 15 Flow04.....	56
Figure 16 Flow05	57
Figure 17 Flow06-a.....	58
Figure 18 Flow06-b.....	59
Figure 19 Flow06-c.....	60
Figure 20 GUI Component Hierarchy Tree	61
Figure 21 Diagram 01: System Entrance Frame Sequence Diagram	63
Figure 22 Diagram 02: Insert New Recipe Sequence Diagram	64
Figure 23 Diagram 03: Edit Recipe Sequence Diagram	65
Figure 24 Diagram 04: Delete Recipe Sequence Diagram	66
Figure 25 Diagram 05: Import Recipe Sequence Diagram.....	67
Figure 26 Diagram 06: Search Recipe Frame Sequence Diagram.....	68
Figure 27 Diagram 07: Change Password Panel Sequence Diagram	69
Figure 28 the System Entrance Interface	70
Figure 29 the General User Interface.....	71
Figure 30 the Administrator Interface-Insert Panel	71
Figure 31 the Administrator Interface-edit panel.....	72
Figure 32 the Administrator Interface-delete panel	72
Figure 33 the Administrator Interface-import panel.....	73
Figure 34 the Administrator Interface-personal setting panel	73
Figure 35 the example recipe table	74
Figure 36 input for the test 1	74
Figure 37 result of the test 1	75
Figure 38 the input for the test 2.....	75
Figure 39 the result of the test 2.....	76
Figure 40 the example admin table	76
Figure 41 the input for the test 'admin. login'	76
Figure 42 the import panel.....	77
Figure 43 the window for selecting files.....	78
Figure 44 a valid recipe file is selected.....	78
Figure 45 the text content of the file 'Whipping Cream Pound Cake.txt'	79
Figure 46 successfully insert the recipe 'Whipping Cream Pound Cake'	80
Figure 47 the display of the recipe "Whipping Cream Pound Cake"	80
Figure 48 the example material table.....	81

Figure 49 the temporary paragraph generated during the import	81
Figure 50 the target recipe has been inserted into the recipe table	82
Figure 51 the ingredients of the targeted recipe has been inserted into the ingredient table	82
Figure 52 the recipe 'Whipping Cream Pound Cake.txt' has already stored in the database	82
Figure 53 the example invalid file which loses the ingredient description.....	83
Figure 54 the response indicates that the file is invalid	83
Figure 55 the nested tag pairs	86
Figure 56 the example HTML page.....	87
Figure 57 the HTML source code (a) for the example page.....	87
Figure 58 the source code (b) for the example HTML page.....	88
Figure 59 the source code (c) for the example HTML page.....	88
Figure 60 the UML class diagram for the extraction class	94
Figure 61 the procedure of the import process	95
Figure 62 the flow chart for parsing the HTML Document.....	97
Figure 64 the flow chart for Extraction.....	100
Figure 65 the flow chart for extracting an entire paragraph	101
Figure 66 the flow chart for checking the recipe ingredient paragraph.....	102
Figure 67 the flow chart for extracting the recipe direction	103
Figure 68 the flow chart for inserting the recipe ingredient into the TempMaterial table	104
Figure 69 the example page 'CNN website'	106
Figure 70 the error message is pop out when dealing with the invalid recipe file.....	107
Figure 72 the source code for the example HTML page 'Brocco Taco Salad'	108
Figure 73 the example material table.....	108
Figure 74 the source code of the recipe ingredient and direction parts	109
Figure 75 the new generated txt paragraph.....	109
Figure 76 the display of the imported recipe	110
Figure 77 the HTML page, of which the recipe ingredients are written in the table format	110
Figure 78 the HTML source code of the ingredient description.....	111
Figure 79 the extracted txt paragraph of the ingredient description	111
Figure 80 the display result of the imported recipe (test 2)	112

List of Tables

Table 1 Login of the general user	29
Table 2 Login of the administrator	29
Table 3 Search the recipe	30
Table 4 Search the Recipe by the Title	30
Table 5 Search the Recipe by the Ingredient	31
Table 6 Search the Recipe by the Category	31
Table 7 Modify the Recipe Database	31
Table 8 Insert the Recipe	32
Table 9 Edit the Recipe	32
Table 10 Delete the Recipe	33
Table 11 Import the Recipe	33
Table 12 Modify the Password	34
Table 13 Modify the Password	34
Table 14 the Recipe Table	40
Table 15 The Ingredient Table	40
Table 16 The Category Table	41
Table 17 The Material Table	41
Table 18 The Unit Table	42
Table 19 The Admin Table	42
Table 20 Select	43
Table 21 UPDATE	43
Table 22 Delete	44
Table 23 Insert Into	44
Table 24 Load the Drivers	45
Table 25 Establish the Connection	45
Table 26 Execute the Update	45
Table 27 Execute the Delete	46
Table 28 Execute the Insert	46
Table 29 Execute the Query	46

Appendix I Installation Guide

Execution Requirements

Hardware requirements:

- Intel Pentium III, 600 MHz or equivalent CPU
- 128 Mb of RAM
- 10/100Mb Network Card

Software requirements:

- J2EE 1.4.2 SDK (Java 2 Platform, Enterprise Edition, Software Development Kit)
- HyperText Markup Language (HTML) version 4.01
- Microsoft Access 2000
- Java Virtual Machine

Execution Indication

1. Double click the soluton1.jar (or solution2.jar)
2. The database system entrance interface appears. When click on the ‘Administrator’ button, a dialog window appears and indicate the user to input name and password.
3. Input the following Administrator name and password
Name1: linlinwang Password: 19781130
Or
Name2: bojiang Password: 19790702
4. Import the example recipe files. All the recipe files are stored in the folder: External Recipe Files. The example files of external recipe for solution 1 and solution 2 are stored in the two child folders of External Recipe Files: Recipes for Solution1 and Recipes for Solution2. The detail information of the recipe files is show in the following table:

Examples of External Recipe File for Solution 1

	File Name	Location	Property
1	Beef Pepper Steak.txt	.\External Recipe Files \Recipes for Solution 1	Valid recipe files Can be imported successfully
2	Dutch Oven Buttermilk Cornbread.txt	.\External Recipe Files \Recipes for Solution 1	Valid recipe files Can be imported successfully
3	Apple Ple Parfaits	.\External Recipe Files	Valid recipe files

	Recipe.txt	\Recipes for Solution 1	Can be imported successfully
4	Golden Harvest Beef Recipe.txt	.\External Recipe Files \Recipes for Solution 1	Valid recipe files Can be imported successfully
5	Soups and Stews.txt	.\External Recipe Files \Recipes for Solution 1	Valid recipe files Can be imported successfully
6	Whipping Cream Pound Cake.txt	.\External Recipe Files \Recipes for Solution 1	Valid recipe files Can be imported successfully

Examples of External Recipe File for Solution 2

	File Name	Location	Property
1	Arizona Brown Rice Pilaf.htm	.\External Recipe Files \Recipes for Solution 2	Valid recipe files Can be imported successfully
2	Brocco Taco Salad.htm	.\External Recipe Files \Recipes for Solution 2	Valid recipe files Can be imported successfully
3	Frank's Famous Spaghe tti Sauce.htm	.\External Recipe Files \Recipes for Solution 2	Valid recipe files Can be imported successfully
4	Meals For You Peach Pan Dowdy II.htm	.\External Recipe Files \Recipes for Solution 2	Valid recipe files Can be imported successfully
5	Mom's Best Peanut Brittle.htm	.\External Recipe Files \Recipes for Solution 2	Valid recipe files Can be imported successfully

6	Salad Chutney Chicken Salad.htm	.\External Recipe Files \Recipes for Solution 2	Valid recipe files Can be imported successfully
---	------------------------------------	--	---

Appendix II Configuration of Source Code

The recipe database system consists of two parts :

1. Access Database

There are 8 tables in the database:

- Recipe: To save the recipe ID, title, category and direction
- Ingredient: To save the ingredient quantity, unit and ingredient description
- Category: To save the recipe category
- Unit: To save the unit words could appear in the recipe ingredient description
- Material: To save some key materials may be used for matching recipe category
- CommonWord(only in Solution 2): To save some common words may appear in the recipe direction
- MaterialTemp(Just in Solution 2): To save the temporary materials of the imported recipe
- Admin : To save the administrator name and password

2. Java Program

• Solution 1

There are 6 Classes defined in this solution:

- RecipeQuerySystem.java: The main driver which can call the RecipeQueryFrame class.
- RecipeQueryFrame.java: The system entrance interface which can call the UserFrame and AdminFrame class.
- UserFrame.java: The general user interface which includes an inner class-RecipeFrame.
- AdminFrame.java: The administrator interface which includes the following inner class: InsertPanel, EditPanel, DeletePanle, ImportPanle , PerPanel and RecipeDisplay.
- ModifyRecipe.java: It is used to handle the common operations such as inserting the new recipe into the database. .
- ExtractInformation.java: It is used to handle the importing operations from the users.

• Solution 2

There in one function package and 7 classes defined in solution 2:

The Package – MyUtils includes two classes:

- MyUtil.class: It includes the complementary methods such as: isElement(), similarString() and etc.
- ParserGetter.class: It is used to get the parser to parse the HTML document.

The rest of classes defined in solution 2 are:

- RecipeQuerySystem.java: The main driver which can call the RecipeQueryFrame class.
- RecipeQueryFrame.java: The system entrance interface which can call the UserFrame and AdminFrame class.
- UserFrame.java: The general user interface which includes an inner class, RecipeFrame.
- AdminFrame.java: The administrator interface which includes the following inner class: InsertPanel, EditPanel, DeletePanle, ImportPanle , PerPanel and RecipeDisplay.
- ModifyRecipe.java: It is used to handle the common operations such as inserting the new recipe into the database.
- ExtractInformation02.java: It is used to handle the importing operations from the users.

Appendix II Test Results

ModifyRecipe.java

```
//*****  
// Final Thesis Project Author: s020953 LinLin Wang  
// ModifyRecipe.java 16.August 2004  
// Implements operation functions of changing database  
//*****
```

```
import java.util.*;  
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
import javax.swing.border.*;  
import javax.swing.table.*;  
import java.sql.*;
```

```
public class ModifyRecipe  
{  
    private ResultSet rs, rs1, rs2, rs3, rs4;  
    private Connection con;  
    private ResultSetMetaData rsmd;  
    private DatabaseMetaData dma;  
    private Statement stmt;  
    private String titleQuery = ("SELECT Title FROM Recipe");
```

```
public ModifyRecipe ()  
{  
    try{  
        String url="jdbc:odbc:driver={Microsoft Access Driver (*.mdb)}; DBQ =  
            + Recipes.mdb";  
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
        con = DriverManager.getConnection(url, "linlin", "19781130");  
    }//end try  
    catch (Exception e){} // end catch  
}
```

```
//-----  
// Insert New Recipe Into the Database  
//-----
```

```
public boolean insert (String title, String categ, String direction, String[][] ingredient )  
{  
    int num, num1, num2;  
    try{  
        stmt = con.createStatement();  
        String directionQuery = ("SELECT Direction FROM Recipe ");  
        rs4 = stmt.executeQuery(directionQuery);  
        rs4.next();  
        do{  
                                                    # 1
```

```

    num2=rs4.getRow();
}while(rs4.next());
String directions[]= new String[num2];
rs4 = stmt.executeQuery(directionQuery);
for(int i=0; i<num2; i++) # 2
{
    rs4.next();
    directions[i]=rs4.getString("Direction");
} //end for

// new recipe doesn't exist in database
if (ExtractInformation.SameRecipe(direction, directions)) # 3
    return false;
else
{
    String title1 = ExtractInformation.correctString(title);
    String categ1 = ExtractInformation.correctString(categ);
    String direction1 = ExtractInformation.correctString(direction);
    String insert1 = "INSERT INTO Recipe (Title, Category, Direction) VALUES ("
        + title1 + ", "+categ1+", "+direction1+"");
    stmt.executeUpdate(insert1);
    // get new recipe's ID
    String IDquery = ("SELECT Rec_ID FROM Recipe");
    rs2 = stmt.executeQuery(IDquery);
    int Rec_ID;
    rs2.next();
    do{ # 4
        String Rec_ID1=rs2.getString("Rec_ID");
        Rec_ID =Integer.parseInt(Rec_ID1);
    } while(rs2.next());
    String s1="", s2="", s3="";
    // get new recipe's ingredien
    boolean b= true;
    for(int i=0; i<20 && b; i++) # 5
    {
        s1 = ingredient[i][0];
        s2 = ingredient[i][1];
        s3 = ingredient[i][2];
        if(s3 == null) # 6
            b = false;
        else {
            String s11 = ExtractInformation.correctString(s1);
            String s22 = ExtractInformation.correctString(s2);
            String s33 = ExtractInformation.correctString(s3);
            String insert = "INSERT INTO Ingredient ( Rec_ID, Quantity, Unit, Ingredient)"
                + "VALUES (" + Rec_ID+", "+s11 + ", "+ s22 + ", "+ s33 + ")";
            stmt.executeUpdate(insert);
        } // end else
    } // end for i
    return true;

```



```

    } // end else
  } // end try
  catch (Exception e) return false;
} // end insert method

// -----
// Edit recipe data in the database
// -----

public boolean edit (int id, String title, String categ, String direction, String[][] ingredient )
{
  String s=ingredient[0][0];
  if((title.compareToIgnoreCase(""))==0 || (direction.compareToIgnoreCase(""))==0
      title==null || direction == null) # 1
    return false;
  else{
    try {
      stmt = con.createStatement();
      String title1 = ExtractInformation.correctString(title);
      String categ1 = ExtractInformation.correctString(categ);
      String direction1 = ExtractInformation.correctString(direction);
      String update1 = "UPDATE Recipe SET Title = '"+title1+"', "+ "Category = '"
          +categ1+"', "+ "Direction = '"+direction1+"'"+"WHERE Rec_ID =
          "+id;

      stmt.executeUpdate(update1);
      // get new recipe's ingredient
      String s1="", s2="", s3="";
      boolean b= true;
      String del = "DELETE FROM Ingredient "+ "WHERE Rec_ID = "+id;
      stmt.executeUpdate(del);
      for(int i=0; i<20 && b; i++) # 2
      {
        s1 = ingredient[i][0];
        s2 = ingredient[i][1];
        s3 = ingredient[i][2];
        if(s1==null) # 3
          b = false;
        else {
          String s11 = ExtractInformation.correctString(s1);
          String s22 = ExtractInformation.correctString(s2);
          String s33 = ExtractInformation.correctString(s3);
          String insert = "INSERT INTO Ingredient ( Rec_ID, Quantity, Unit, Ingredient)"
              + "VALUES (" + id+", "+s11 +", "+ s22 +", "+ s33 +")";
          stmt.executeUpdate(insert);
        } // end else
      } // end for i
      return true;
    } // end try
    catch (Exception e) return false;
  }
}

```

```

    } // end else
} // end edit method

// -----
// Search Recipe
// -----

public void searchRecipe(String title, String ingredient, String categ, Vector id, Vector
names)
{
    try{
        if (categ!=null) # 1
        {
            String get_ID= ("SELECT Rec_ID, Title, Direction FROM Recipe "+"WHERE
                Category =" "+categ+ " ");
            if(title==null ) # 2
            {
                if (ingredient == null) # 3
                {
                    stmt = con.createStatement();
                    rs = stmt.executeQuery(get_ID);
                    while( rs.next()) # 4
                    {
                        String rtitle= rs.getString("Title");
                        String ID= rs.getString("Rec_ID");
                        id.add(ID);
                        names.add(rtitle);
                    } // end while( rs.next())
                } // if (ingredient == null)
            } // if (ingredient!=null) # 5
            {
                stmt = con.createStatement();
                rs = stmt.executeQuery(get_ID);
                while( rs.next()) # 6
                {
                    String rtitle= rs.getString("Title");
                    String ring = rs.getString("Direction");
                    if (ExtractInformation.isElement(ingredient, ring)) # 7
                    {
                        String ID= rs.getString("Rec_ID");
                        id.add(ID);
                        names.add(rtitle);
                    }
                } // end while( rs.next())
            } // end if (ingredient!=null)
        } // end if(title==null )
        if(title!=null ) # 8
        {
            if (ingredient == null) # 9
            {

```

```

    stmt = con.createStatement();
    rs = stmt.executeQuery(get_ID);
}
while( rs.next()) # 10
    String rtitle= rs.getString("Title");
if (ExtractInformation.isElement(title, rtitle)) # 11
{
    String ID= rs.getString("Rec_ID");
    id.add(ID);
    names.add(rtitle);
}
} // end while( rs.next())
} // if (ingredient != null)
if(ingredient != null ) # 12
{
    stmt = con.createStatement();
    rs = stmt.executeQuery(get_ID);
while( rs.next()) # 13
{
    String rtitle= rs.getString("Title");
    String ring = rs.getString("Direction");
if (ExtractInformation.isElement(title, rtitle) &&
    ExtractInformation.isElement(ingredient, ring)) # 14
{
    String ID= rs.getString("Rec_ID");
    id.add(ID);
    names.add(rtitle);
    }
    } // end while( rs.next())
    } // end if(ingredient != null )
    } // end if(title!=null )
} // end if (categ!=null)
else {
String get_ID= ("SELECT Rec_ID, Title, Direction FROM Recipe ");
if(title==null ) # 15
{
    if (ingredient == null) # 16
    {
        stmt = con.createStatement();
        rs = stmt.executeQuery(get_ID);
while( rs.next()) # 17
        {
            String rtitle= rs.getString("Title");
            String ID= rs.getString("Rec_ID");
            id.add(ID);
            names.add(rtitle);
        } // end while( rs.next())
    } // if (ingredient == null)
if (ingredient!=null) # 18
    {

```

```

stmt = con.createStatement();
rs = stmt.executeQuery(get_ID);
while( rs.next())
{
String rtitle= rs.getString("Title");
String ring = rs.getString("Direction");
if ( ExtractInformation.isElement(ingredient, ring))
{
String ID= rs.getString("Rec_ID");
id.add(ID);
names.add(rtitle);
}
} // end while( rs.next())
} // end if (ingredient!=null)
} // end if(title==null )
if(title!=null )
{
if (ingredient == null)
{
stmt = con.createStatement();
rs = stmt.executeQuery(get_ID);
while( rs.next())
{
String rtitle= rs.getString("Title");
if ( ExtractInformation.isElement(title, rtitle))
{
String ID= rs.getString("Rec_ID");
id.add(ID);
names.add(rtitle);
}
} // end while( rs.next())
} // if (ingredient != null)
if(ingredient != null )
{
stmt = con.createStatement();
rs = stmt.executeQuery(get_ID);
while( rs.next())
{
String rtitle= rs.getString("Title");
String ring = rs.getString("Direction");
if (ExtractInformation.isElement(title, rtitle) &&
ExtractInformation.isElement(ingredient, ring))
{
String ID= rs.getString("Rec_ID");
id.add(ID);
names.add(rtitle);
}
} // end while( rs.next())
} // end if(ingredient != null )

```

```

        } // end if(title!=null )
    } // end else
} // end try
catch (Exception e) {}
} // end method

// -----
// Delete old recipe from the database
// -----

public void delete (int id)
{
    try{
        stmt = con.createStatement();
        String del = "DELETE FROM Recipe "+"WHERE Rec_ID = "+id;
        stmt.executeUpdate(del);
        String del2 = "DELETE FROM Ingredient "+"WHERE Rec_ID = "+id;
        stmt.executeUpdate(del2);
    } // end try
    catch (Exception e) {}
} // end delete method

} // end class

```

ExtractInformation.java

```
//*****
// Final Thesis Project                               Author: s020953 LinLin Wang
// ExtractInformation.java                             16.August 2004
// Extract recipe data from external file and put them into database
//*****

import java.sql.*;
import java.util.*;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.StringTokenizer;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.BufferedWriter;

public class ExtractInformation
{
    private int Rec_ID=0 ;
    private final static int RecipeExist = -1;
    private final static int InvalidRecipe = -2;
    private final static int FileNotFound = -3;
    private final static int IOExcep = -4;
    private final static int Excep = -5;
    private Connection con;
    private String url=
        "jdbc:odbc:driver={MicrosoftAccessDriver (*.mdb)};DBQ=Recipes.mdb";
public ExtractInformation()
{
    try{
        // load JDBC-ODBC bridge driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        // connect to database
        con = DriverManager.getConnection(url, "linlin", "19781130");
    } // end try
    catch (Exception e)
        System.out.println(e+ " Exception From ExtractInformation constructor!");
}

// -----
// Get the imported recipe ID
// -----

public int getID()
{
    return Rec_ID;
}
}
```

```

// -----
// Extract recipe ingredient and direction and put them into database
// -----

public int Extract(FileReader fr, String title)
{
    Vector v1=new Vector(), v2= new Vector(), v3=new Vector();
    String cat;
    try{
        File paragraph = new File("paragraph.txt");
        FileWriter fw = new FileWriter(paragraph );
        ExtractParagraph(fr, fw);
        // check if the recipe file valid
        if( !ValidRecipe(paragraph))                # 1
            // recipe file is invalid
            return InvalidRecipe;
        else{
            FileReader fr1 = new FileReader(paragraph);
            FileReader fr2 = new FileReader(paragraph);
            BufferedReader br = new BufferedReader(fr1);
            String kw1 = "ingredient";
            String direction=ExtractDirection(br);
            // store recipe's title and direction into database
            ResultSet rs1, rs2, rs3, rs4;
            int num, num1, num2;
            Statement stmt = con.createStatement();
            String directionQuery = ("SELECT Direction FROM Recipe ");
            rs4 = stmt.executeQuery(directionQuery);
            rs4.next();
            do{                # 2
                num2=rs4.getRow();
            }while(rs4.next());
            String directions[]= new String[num2];
            rs4 = stmt.executeQuery(directionQuery);
            for(int i=0; i<num2; i++)                # 3
            {
                rs4.next();
                directions[i]=rs4.getString("Direction");
            } //end for
            if(SameRecipe(direction, directions))    # 4
                // Recipe has exist
                return RecipeExist;
            else
            {
                // set recipe category
                cat = ExtractCategory(title, direction);
                String title1 = correctString(title);
                String cat1 = correctString(cat);
                String direction1 = correctString(direction);
            }
        }
    }
}

```

```

stmt.executeUpdate("INSERT INTO Recipe (Title,Category, Direction) VALUES
    (" + title1 + ", " + cat1 + ", " + direction1 + ")");
String IDquery = ("SELECT Rec_ID FROM Recipe");
rs1 = stmt.executeQuery(IDquery);
rs1.next();
do{
    String Rec_ID1=rs1.getString("Rec_ID");
    Rec_ID =Integer.parseInt(Rec_ID1);
} while(rs1.next());
String line1;
StringTokenizer token1, token2;
String unit = ("SELECT Name FROM Unit");
// judge if this line contains exist units, if it exist, extract it.
// Read data from database
rs2 = stmt.executeQuery(unit);
rs2.next();
do{
    num=rs2.getRow();
}while(rs2.next());
String units[]= new String[num];
rs2 = stmt.executeQuery(unit);
for(int i=0; i<num; i++)
{
    rs2.next();
    units[i]=rs2.getString("Name");
} //end for
// Extract Ingredient
br = new BufferedReader(fr2);
String [] words;
while(br.ready() )
{
    line1= br.readLine();
while(!isElement(kw1,line1) && br.ready())
    line1= br.readLine();
    if(br.ready())
    line1= br.readLine();
    line1.trim();
token2 = new StringTokenizer(line1);
int cou =token2.countTokens() ;
while(cou==0 && br.ready())
{
    line1= br.readLine();
    token2 = new StringTokenizer(line1);
    cou =token2.countTokens() ;
    }
    line1.trim();
    token1 = new StringTokenizer(line1);
token2 = new StringTokenizer(line1);
while(token2.hasMoreTokens() )
{

```



```

String first2=token2.nextToken();
    while(br.ready())                                # 13
    {
        line1=br.readLine();
        token1=new StringTokenizer(line1);
        boolean a=true;
        if(token1.hasMoreTokens() && a)              # 14
    {
        v1=new Vector(); v2=new Vector(); v3=new Vector();
        int count = token1.countTokens();
        words = new String[count];

        // store the contents in tokens into string array "words"
        for(int i=0; i<count; i++)                    # 15
            words[i] = token1.nextToken();
        // suppose the first word always numerical
        v1.add(words[0]);
        boolean b=true;
        for(int i=0; i<words[0].length() && b; i++)    # 16
        {
            // first token contain "(", we will find another part ")"
            if(words[0].charAt(i)==40)                # 17
                b=false;
        } // end for
        //first token contain "("
        if (!b)                                       # 18
        {
            boolean c=true;
            for(int m=1; (m<words.length) && c; m++)  # 19
            {
                for(int n=0; (n<words[m].length()) && c; n++) # 20
                {
                    // first token contain "(", we have to find another part ")"
                    if(words[m].charAt(n)==41)        # 21
                    {
                        v1.add(words[m]);
                        c=false;
                        boolean d=true;
                        // judge whether unit word exist in the line
                        for(int i=m+1; ((i<words.length+1) && d); i++) # 22
                        {
                            for(int j=0; (j<units.length)&& d; j++) # 23
                            {
                                if(words[i].compareToIgnoreCase(units[j])==0) # 24
                                {
                                    // judge if adjective word exist before unit word
                                    for(int x=0; (x<words[i-1].length())&& d; x++) # 25
                                    {
                                        // when it's numeric or exist ")", we ignore it
                                        if( ((words[i-1].charAt(x)>=48) && # 26

```

```

(words[i-1].charAt(x) <= 57)) ||
((words[i-1].charAt(x)==41))
{
    v2.add(words[i]);
    d=false;
} //end if
} // for x
if(d) # 27
{
    v2.add(words[i-1]);
    v2.add(words[i]);
    d=false;
}
} //if
} // for j
} //for i
} // if
} // end for n
if (c) # 28
    v1.add(words[m]);
} // for m
} // if !b
// first token doesn't contain "("
else
{
    if (words.length>1) # 29
    {
        for(int e=0; (e<words[1].length() && a); e++) # 30
        {
            // second token contain "("
            if(words[1].charAt(e)==40) # 31
            {
                if(words[1].charAt(words[1].length()-1)!=41) # 32
                {
                    a=false;
                    boolean c=true;
                    for(int m=1; (m<words.length) && c; m++) # 33
                    {
                        for(int n=0; (n<words[m].length()) && c; n++) # 34
                        {
                            // first token contain "(", we have to find another part ")"
                            if(words[m].charAt(n)==41) # 35
                            {
                                c=false;
                                boolean d=true;
                                // judege whether unit word exist in the line
                                for(int i=m+1; ((i<words.length+1) && d); i++) # 36
                                {
                                    for(int j=0; (j<units.length)&& d; j++) # 37
                                    {

```

```

        if(words[i].compareToIgnoreCase(units[j])==0) # 38
        {
// judge if express word exist before unit word
        for(int x=0;x<words[i-1].length(); x++) # 39
        {
// when it's numeric or exist ")", we ignore it
        if( ((words[i-1].charAt(x)>=48) &&
            (words[i-1].charAt(x)<=57)) ||
            ((words[i-1].charAt(x)==41)) ) # 40
        {
            v2.add(words[i]);
            d=false;
        } //end if
        } // for x
        if(d) # 41
        {
            v2.add(words[i-1]);
            v2.add(words[i]);
            d=false;
        }
        } //if
    } // for j
} //for i
} // if(words[m].charAt(n)==41)
} // end for n
} // for m
} // if(words[1].charAt(words[1].length()-1)!=41)
else
{
    a=false;
    boolean d=true;
// judge whether unit word exist in the line
    for(int i=1; (i<words.length)&& d; i++) # 42
    {
        for(int j=0; (j<units.length)&& d; j++) # 43
        {
            if(words[i].compareToIgnoreCase(units[j])==0) # 44
            {
// judge if express word exist before unit word
                for(int x=0; (x<words[i-1].length())&& d; x++) # 45
                {
// when it's numeric or exist ")", we ignore it
                    if( (words[i-1].charAt(x)>=48) &&
                        (words[i-1].charAt(x)<=57) ||
                        (words[i-1].charAt(x)==41)) # 46
                    {
                        v2.add(words[i]);
                        d=false;
                    } //end if
                }
            }
        }
    }
}

```

```

        {
            v2.add(words[i-1]);
            v2.add(words[i]);
            d=false;
        } // else
    } // for x
} //if
} // for j
} //for i
} // end else
} // end if(words[1].charAt(e)==40)
// second token doesn't contain "("
else
{
    // next token is numeric
if( (words[1].charAt(e)>=48) && (words[1].charAt(e)<=57)) # 47
    {
        v1.add(words[1]);
        a=false;
        boolean d=true;
        // judge whether unit word exist in the line
for(int i=1; (i<words.length)&& d; i++) # 48
    {
        for(int j=0; (j<units.length)&& d; j++) # 49
            {
                if(words[i].compareToIgnoreCase(units[j])==0) # 50
                    {
                        // judge if express word exist before unit word
for(int x=0; (x<words[i-1].length())&& d; x++) # 51
                            {
                                // when it's numeric or exist ")", we ignore it
if( (words[i-1].charAt(x)>=48) &&
                                (words[i-1].charAt(x)<=57) ||
                                (words[i-1].charAt(x)==41)) # 52
                                    {
                                        v2.add(words[i]);
                                        d=false;
                                    } //end if
                                else
                                    {
                                        v2.add(words[i-1]);
                                        v2.add(words[i]);
                                        d=false;
                                    } // else
                                } // for x
                            } //if
                        } // for j
                    } //for i
                } // end if( (words[1].charAt(e)>=48) &&(words[1].charAt(e)<=57))
                // second token is nether numeric nor "("

```

```

else
{
    boolean g=true;
// judge whether unit word exist in the line
    for(int i=1; (i<words.length && g && a); i++)      # 53
    {
        for(int j=0; (j<units.length && g); j++)      # 54
        {
            if(words[i].compareToIgnoreCase(units[j])==0 ) # 55
            {
                // judge if express word exist before unit word
                for(int y=0; (y<words[i-1].length() && g); y++) # 56
                {
                    // when it's numeric or exist ")", we ignore it
                    if( ( (words[i-1].charAt(y)>=48) &&
                        (words[i-1].charAt(y)<=57)) ||
                        (words[i-1].charAt(y)==41))      # 57
                    {
                        v2.add(words[i]);
                        g = false;
                        a = false;
                    } //end if
                } // for y
                if(g) # 58
                {
                    v2.add(words[i-1]);
                    v2.add(words[i]);
                    g=false;
                    a=false;
                } // end if (g)
            } //if
        } // for j
    } //for i
    } // end else
    } // end else
    } // for e
    } // end else
// store ingredient description into v3
    for(int i=0; i<words.length ; i++)      # 59
    {
        boolean o=true;
        for(int j=0; j<v1.size() && o; j++)      # 60
        {
            if(words[i].compareToIgnoreCase((v1.elementAt(j)).toString())==0) # 61
            {
                o=false;
            } // for j
            if(o) # 62
            {
                for(int x=0; x<v2.size() && o ; x++)      # 63
                {

```

```

if(words[i].compareToIgnoreCase(v2.elementAt(x)).toString()==0) # 64
    o=false;
    } // for x
    if(o) # 65
    {
    v3.add(words[i]);
    o=false;
    } // if o
    } // if o
    } // for i
    // store the data to database
    // first, convert the contents of v1, v2, v3 to string format
    String s1="", s2="", s3="";
    for(int i=0 ;i<v1.size(); i++) # 66
        s1=s1+v1.elementAt(i)+" ";
    for(int i=0 ;i<v2.size(); i++) # 67
        s2=s2+v2.elementAt(i)+" ";
    for(int i=0 ;i<v3.size(); i++) # 68
        s3=s3+v3.elementAt(i)+" ";
    // store the new ingredient description into database , update database
    String s11 = correctString(s1);
    String s22 = correctString(s2);
    String s33 = correctString(s3);
    String query = "INSERT INTO Ingredient ( Rec_ID, Quantity,
Unit, Ingredient)" + "VALUES (" + Rec_ID+", "+s11 +", "+ s22 +", "+ s33 +")";
    stmt.executeUpdate(query);
    rs1 = stmt.executeQuery(unit);
    } // if (words.length>1)
    } // if(token1.hasMoreTokens() && a)
    else
    return 0 ;
    } // while(br.ready())
    } // while(token2.hasMoreTokens())
    } // while(br.ready())
    } // end else
    } // end else
    return InvalidRecipe;
    } // end try
    catch(FileNotFoundException e)
        return FileNotFound;
    catch(IOException ioe)
        return IOExcep;
    catch (Exception e)
} // end Extract method

// -----
// Extract the specific paragraph which contain Ingredient and Direction description
// -----

private static void ExtractParagraph(FileReader fr, FileWriter fw)

```

```

{
try{
StringTokenizer token1, token2;
boolean a = true;
BufferedWriter bw = new BufferedWriter(fw);
BufferedReader br = new BufferedReader(fr);
String line1="";
String kw1 = "ingredient", kw2 = "direction", kw3 = "procedure", kw4 =
"instruction";
Vector v1=new Vector(), v2 = new Vector();
if(br.ready()) # 1
line1= br.readLine();
while( !isElement(kw1,line1) && br.ready()) # 2
line1= br.readLine();
while(br.ready() && a) # 3
{
v1.add(line1);
line1 = br.readLine();
if(isElement(kw1,line1)) # 4
{
v1.removeAllElements();
} // end if
if(isElement(kw2,line1) || isElement(kw3,line1) || isElement(kw4,line1) &&
br.ready()) # 5
{
v2.add(line1);
line1= br.readLine();
token2 = new StringTokenizer(line1);
while(token2.countTokens()==0&& br.ready()) # 6
{
line1=(br.readLine());
token2=new StringTokenizer(line1);
}
v2.add(line1);
if(br.ready())
{
line1=(br.readLine()); }
token1 = new StringTokenizer(line1);
while(token2.countTokens()!=0&& br.ready()) # 7
{
v2.add(line1);
line1=(br.readLine());
token2=new StringTokenizer(line1);}
a =false;
} // end if
} // end while (br.ready)
}
}
for(int i=0; i<v1.size(); i++) # 8
{

```

```

        bw.write((String)v1.elementAt(i));
            bw.newLine();
        }
for(int i=0; i<v2.size(); i++)           # 9
    {
        bw.write((String)v2.elementAt(i));
        bw.newLine();
    }
    bw.close();
}
catch(IOException ioe) {}
} // end method

// -----
// Check Whether Recipe File Valid
// -----

private boolean ValidRecipe(File paragraph) throws IOException
{
    StringTokenizer token1;
    int line_count=0;
    boolean valid = false;
    try{
        FileReader fr = new FileReader(paragraph);
        BufferedReader br = new BufferedReader (fr);
        String line1="";
        String kw1 = "ingredient", kw2 = "direction", kw3 = "procedure", kw4 =
"instruction";
        StringTokenizer token2;
        if(br.ready())                   # 1
            line1 = br.readLine();
        while( !isElement(kw1,line1) && br.ready()) # 2
            line1 = br.readLine();
        if(br.ready())                   # 3
            line1 = br.readLine();
            token2 = new StringTokenizer(line1);
            while(token2.countTokens()>0 && br.ready()) # 4
            {
                line1= br.readLine();
                token2 = new StringTokenizer(line1);
            }
        while( !isElement(kw2,line1) &&!isElement(kw3,line1)&& !isElement(kw4,line1) ) # 5
        {
            if (br.ready())               # 6
            {
                line_count ++;
                line1=br.readLine();
            }
        }
        if( line_count>1 )                # 7

```



```

        {
            if (br.ready())
                valid = true;
        } // end if (2)
        return valid;
    } // end try
    catch(IOException ioe) {};
    return valid;
} // end ValidRecipe method

// -----
// Extract Direction part from the file
// -----

private static String ExtractDirection(BufferedReader br) throws IOException
{
    String kw2 = "direction", kw3 = "procedure", kw4 = "instruction";
    StringTokenizer token1, token2;
    String sum="", line1, line2;
    while(br.ready())
    {
        line1= br.readLine();
        while( !isElement(kw2,line1) && ! isElement(kw3,line1) &&!isElement(kw4,line1) )
        {
            line1= br.readLine();
            while(br.ready())
            {
                line1= br.readLine();
                line1.trim();
                token1 = new StringTokenizer(line1);
                token2 = new StringTokenizer(line1);
                if (token2.countTokens() != 0)
                {
                    sum=sum+line1+"\n";
                    while(token2.hasMoreTokens())
                    {
                        String first2=token2.nextToken();
                        while(br.ready())
                        {
                            line1=(br.readLine()).trim();
                            token1=new StringTokenizer(line1);
                            if(token1.hasMoreTokens() )
                            {
                                sum=sum+line1+"\n";
                                else return sum;
                            } //while(br.ready())
                        } // while(token2.hasMoreTokens())
                    }
                }
            } // while(br.ready())
        }
        return sum;
    } // end ExtractDirection
}

```

```

// -----
// Define Category Depending on Recipe's Title or Direction
// -----

private String ExtractCategory(String title, String direction)
{
    String get_keywords = ("SELECT Name, Category FROM Material");
    String category = "Others", line = "";
    ResultSet rs1;
    try{
        Statement stmt = con.createStatement();
        rs1 = stmt.executeQuery(get_keywords);
        while (rs1.next())                # 1
        {
            String material = rs1.getString("Name");
            if ( isElement(material, title) )                # 2
            {
                category = rs1.getString("Category");
                return category;
            } // end if
        } // end while
        rs1 = stmt.executeQuery(get_keywords);
        while (rs1.next())                # 3
        {
            String material = rs1.getString("Name");
            if ( isElement(material, direction) )                # 4
            {
                category = rs1.getString("Category");
                return category;
            } // end if
        } // end while
        return category;
    } // end try
    catch (Exception e) return category;
} // end ExtractCategory method

// -----
// Check whether the two recipe is the same recipe
// -----

public static boolean SameRecipe( String direction, String[] directions)
{
    boolean a2 = false;
    for(int j=0; j<directions.length && !a2; j++)
    {
        if( SameDirection(direction, directions[j]) )
            a2 = true; // recipe has exist in database
    } // end for(int j=0; j<directions.length && a2; j++)
    return a2;
} // end method

```


ExtractInformation02.java

```
//*****
// Final Thesis Project                               Author: s020953 LinLin Wang
// ExtractInformation02.java                           16.August 2004
// Extract recipe data from external file and put them into database
//*****

import java.sql.*;
import java.util.*;
import java.io.*;
import javax.swing.text.*;
import javax.swing.text.html.*;
import javax.swing.text.html.parser.*;
import MyUtils.*;

public class ExtractInformation02 extends Observable
{
    private FileReader fr;
    private File f;
    private Connection con;
    private Vector ingredient;
    private String direction, title, category;
    private String[] units, direct, material, categories, materialTemp, commonWord;
    private int Rec_ID ;
    private final static String url="jdbc:odbc:driver={Microsoft Access Driver +
                                   (*.mdb)};DBQ=Recipes.mdb";
    private final static int RecipeExist = -1;
    private final static int InvalidRecipe = -2;
    private final static int IOExcep = -3;

    public ExtractInformation02()
    {
        try{
            this.f = new File ("RecipeFile.txt");
            this.fr = null;
            this.Rec_ID = 0;
            direction = "";
            title = "";
            category = "Others";
            ingredient = new Vector();
            units = null;
            materialTemp = null;
            commonWord = null;
            direct = null;
            material = null;
            categories = null;
            connectToDatabase();
            getUnitWord();
            getDirect();
            getMaterial();
        }
    }
}
```

```

    } // end try
    catch (Exception e) {}
}
public int getID()
{
    return Rec_ID;
} // end method

// -----
// Extract ingredient and direction
// -----

private boolean extract()
{
    boolean check = false, valid = false;
    try{
        FileReader fr = new FileReader(this.f);
        BufferedReader br = new BufferedReader(fr);

        // search ingredient paragraph
        while (br.ready() && !check) # 1
        {
            ingredient.removeAllElements();
            String line = br.readLine().trim();
            StringTokenizer token1 = new StringTokenizer(line);
            int count = token1.countTokens();
            while(count>0) # 2
            {
                ingredient.add(line);
                if(br.ready()) # 3
            {
                line = br.readLine().trim();
                token1 = new StringTokenizer(line);
                count = token1.countTokens();
            }
            else count=0;
            } // end while
            check = ingredientParagraph(ingredient);
        } // end while br.ready
        if (!check) # 4
        {
            ingredient.removeAllElements();
            return valid;
        }
        else{
            clearMaterialTemp();
            Statement stmt = con.createStatement();
            for(int i=0; i<ingredient.size(); i++) # 5
            {
                String line =(String) ingredient.elementAt(i);

```

```

StringTokenizer token2 = new StringTokenizer(line);
int count = token2.countTokens();
for(int j=0; j<count; j++)
    {
    String word =token2.nextToken();
    if (! (MyUtil.checkFirstChar(word)) && ! (MyUtil.contain(this.units, word)) # 7
        && ! (word.compareToIgnoreCase("or") == 0)
        &&! (word.compareToIgnoreCase("a") == 0)
        &&! (word.compareToIgnoreCase("the")== 0)
        &&! (word.compareToIgnoreCase("to") == 0)
        &&! (word.compareToIgnoreCase("with") == 0)
        &&! (word.compareToIgnoreCase("such") == 0)
        &&! (word.compareToIgnoreCase("any") ==0)
        &&! (word.compareToIgnoreCase("other") == 0)
        &&! (word.compareToIgnoreCase("as") == 0)
        &&! (word.compareToIgnoreCase("from") == 0)
        && !(word.compareToIgnoreCase("and") == 0 ))
        {
            String word01 = MyUtil.correctString02(word);
            String insert1 = "INSERT INTO MaterialTemp (Name) VALUES (" + word01
                + ")";
            stmt.executeUpdate(insert1);
        } // end if
    } // end for j
} // end for i
// extract direction
getMaterialTemp();
getCommonWord();
while (br.ready() && this.direction == "") # 8
    {
String line = br.readLine().trim();
if ( MyUtil.contain (line, this.materialTemp) ||
    MyUtil.contain(line, this.commonWord)) # 9
    {
        this.direction = this.direction+line+"\n";
        this.direction.trim();
    } // end if
} // end while
boolean b = false;
while (br.ready() && !b) # 10
    {
String line = br.readLine().trim();
if ( MyUtil.contain (line, this.materialTemp) ||
    MyUtil.contain(line, this.commonWord)) # 11
    this.direction = this.direction+line+"\n";
else b = true;
    } // end while
    this.direction.trim();
    get_category();
} // end else

```

```

        if (this.direction!="")
            valid = true;
        return valid;
    } // end try
    catch (Exception e) {return valid; }
} // end method

// -----
// Check if this paragraph is ingredient description
// -----

public boolean ingredientParagraph(Vector paragraph)
{
    boolean valid=false;
    try{
        StringTokenizer token ;
        for(int i=0; i<paragraph.size() && !valid; i++)
            # 1
            {
                String line =(String)paragraph.elementAt(i);
                token = new StringTokenizer(line);
                int count = token.countTokens();
                if (MyUtil.checkFirstChar(line)&& count<7) // first char is numerical    # 2
                    {
                        boolean check=false;
                        token = new StringTokenizer(line);
                        while( token.hasMoreTokens() && !check)
                            # 3
                            {
                                String s2 = token.nextToken();
                                check =MyUtil.contain(this.units, s2);
                                if (check)
                                    # 4
                                    valid = true;
                            } // end while
                    } // end if
            } // end for i
        return valid;
    }
    catch (Exception e) {return valid;}
} // end method

// -----
// Extract recipe's category
// -----

private void get_category()
{
    boolean b=false;
    for(int i=0; i<this.material.length && !b; i++)
        # 1
        {
            if ( MyUtil.isElement(material[i],this.title) )
                # 2
                {

```

```

        this.category = this.categories[i];
        b = true;
    } // end if
} // end for i
if (!b) # 3
{
for (int i=0; i< this.material.length && !b; i++) # 4
{
if ( MyUtil.isElement(material[i], this.direction) ) # 5
{
    this.category = this.categories[i];
    b = true;
    } // end if
} // end for i
} // end if
} // end method

// -----
// Put recipe's ingredient data into Ingredient Table
// -----

private void storeIngredient()
{
    try{
        Statement stmt = con.createStatement();
        String query = ("SELECT Unit FROM Ingredient");
        for (int i=0; i<this.ingredient.size(); i++) # 1
        {
            Vector v1 = new Vector();
            StringTokenizer token = new StringTokenizer (
                ((String)this.ingredient.elementAt(i)).trim());
            while (token.hasMoreTokens()) # 2
                v1.add(token.nextToken());
            String w1="", w2 = "";
            if (! (MyUtil.checkFirstChar( (String)v1.elementAt(0)))) # 3
            {
                boolean b = true;
                for (int j=0; j<v1.size() &&b ; j++) # 4
                {
                    if (MyUtil.contain(this.units, ((String)v1.elementAt(j)) )) # 5
                    {
                        w2 = (String)v1.remove(j);
                        b = false;
                    } // end if
                } // end for j
            } // end if
        }
        else {
            w1 = (String)v1.remove(0) ;
            if (MyUtil.checkFirstChar((String)v1.elementAt(0))) # 6
                w1 = w1 + " " +(String)v1.remove(0);
        }
    }
}

```



```

        boolean b = true;
        for (int j =0; j<v1.size() && b; j++) # 7
        {
            if (MyUtil.contain (this.units, (String)v1.elementAt(j))) # 8
            {
                w2 = (String)v1.remove(j);
                b = false;
            } // end if
        } // end for j
    } // end else
    String s="";
    for(int a=0 ;a<v1.size(); a++) # 9
        s=s+v1.elementAt(a)+" ";
    String word1 = MyUtil.correctString02(w1);
    String word2 = MyUtil.correctString02(w2);
    String word3 = MyUtil.correctString02(s);
    String query1 = "INSERT INTO Ingredient ( Rec_ID, Quantity, Unit,
Ingredient)"
        + "VALUES (" + this.Rec_ID+", ""+word1 +", ""+ word2 +", ""+ word3
+""");
    stmt.executeUpdate(query1);
    ResultSet rs1= stmt.executeQuery(query);
    } // end for i
    }
    catch (Exception e) {}
} // end method

// -----
// Connect the program to Database
// -----

private void connectToDatabase()
{
    try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection(url, "linlin", "19781130");
    } // end try
    catch (Exception e) {}
} // end method

// -----
// Initialize MaterialTemp table
// -----

public void clearMaterialTemp()
{
    try{
        String del = "DELETE FROM MaterialTemp ";
        Statement stmt = con.createStatement();
        stmt.executeUpdate(del);
    }
}

```

```

    } // end try
    catch (Exception e) { }
} // end method

// -----
// Get Unit word from the database
// -----

private void getUnitWord()
{
    try{
        String get_unit= ("SELECT Name FROM Unit");
        Statement stmt = con.createStatement();
        ResultSet rs1 = stmt.executeQuery(get_unit);
        StringTokenizer token ;
        boolean valid=false;
        int num;
        rs1.next();
        do{
            num=rs1.getRow();
        } while(rs1.next());
        this.units= new String[num];
        rs1 = stmt.executeQuery(get_unit);
        for(int i=0; i<num; i++)
        {
            rs1.next();
            this.units[i]=rs1.getString("Name");
        } //end for
    }
    catch (Exception e) { }
} // end method

// -----
// Get common used word in recipe's direction from the database
// -----

private void getCommonWord()
{
    try{
        String get_commonWord= ("SELECT Name FROM CommonWord");
        Statement stmt = con.createStatement();
        ResultSet rs1 = stmt.executeQuery(get_commonWord);
        StringTokenizer token ;
        boolean valid=false;
        int num;
        rs1.next();
        do{
            num=rs1.getRow();
        } while(rs1.next());
        this.commonWord= new String[num];
    }
}

```

```

        rs1 = stmt.executeQuery(get_commonWord);
        for(int i=0; i<num; i++)
        {
            rs1.next();
            this.commonWord[i]=rs1.getString("Name");
        } //end for
    }
    catch (Exception e) { }
} // end method

// -----
// Get all the recipe's direction from the database
// -----

public void getDirect()
{
    try{
        String get_direct= ("SELECT Direction FROM Recipe");
        Statement stmt = con.createStatement();
        ResultSet rs1 = stmt.executeQuery(get_direct);
        StringTokenizer token ;
        boolean valid=false;
        int num;
        rs1.next();
        do{
            num=rs1.getRow();
        } while(rs1.next());
        this.direct= new String[num];
        rs1 = stmt.executeQuery(get_direct);
        for(int i=0; i<num; i++)
        {
            rs1.next();
            this.direct[i]=rs1.getString("Direction");
        } //end for
    }
    catch (Exception e) { }
} // end method

// -----
// Get the material keyword from Material table
// -----

public void getMaterial()
{
    try{
        String get_mate= ("SELECT Name, Category FROM Material");
        Statement stmt = con.createStatement();
        ResultSet rs1 = stmt.executeQuery(get_mate);
        StringTokenizer token ;
        boolean valid=false;

```

```

        int num;
        rs1.next();
        do{
            num=rs1.getRow();
        } while(rs1.next());
        this.material= new String[num];
        this.categories = new String[num];
        rs1 = stmt.executeQuery(get_mate);
        for(int i=0; i<num; i++)
        {
            rs1.next();
            this.material[i]=rs1.getString("Name");
            this.categories[i] = rs1.getString("Category");
        } //end for
    }
    catch (Exception e) {}
} // end method

// -----
// Get the temprary material word from the MaterialTemp table
// -----

public void getMaterialTemp()
{
    try{
        String get_m= ("SELECT Name FROM MaterialTemp");
        Statement stmt = con.createStatement();
        ResultSet rs1 = stmt.executeQuery(get_m);
        StringTokenizer token ;
        boolean valid=false;
        int num;
        rs1.next();
        do{
            num=rs1.getRow();
        } while(rs1.next());
        this.materialTemp= new String[num];
        rs1 = stmt.executeQuery(get_m);
        for(int i=0; i<num; i++)
        {
            rs1.next();
            this.materialTemp[i]=rs1.getString("Name");
        } //end for
    }
    catch (Exception e) {}
} // end method

// -----
// Return true if this recipe has already existed in database
// -----

private boolean sameRecipe()

```

```

{
    boolean exist = false;
    for(int i=0; i<this.direct.length && !exist; i++)
    {
        if (MyUtil.similarString(this.direction, direct[i]))
            exist = true;
    } // end for i
    return exist;
} // end method

// -----
// Parse the html file to text file
// -----

private void parserHtml()
{
    ParserGetter kit = new ParserGetter( );
    HTMLEditorKit.Parser parser = kit.getParser( );
    try {
        Outliner callback = new Outliner (new OutputStreamWriter(System.out));
        parser.parse(this.fr, callback, true);
        this.title = callback.get_title();
        Vector v1 = callback.get_v();
        FileWriter fw1 = new FileWriter ("TempFile.txt");
        BufferedWriter bw1 = new BufferedWriter (fw1);
        for(int i=0; i<v1.size(); i++)
        {
            bw1.write((String)v1.elementAt(i));
            bw1.newLine();
        }
        bw1.close();
        FileReader fr1 = new FileReader ("TempFile.txt");
        MyUtil.correctTextFile(fr1, this.f);
    } // end try
    catch (IOException e) {}
} // end main

// -----
// put the recipe's data into the Recipe Table
// -----

private void storeRecipeTable()
{
    try{
        Statement stmt = con.createStatement();
        String tt = MyUtil.correctString02(this.title);
        String dd = MyUtil.correctString02(this.direction);
        stmt.executeUpdate("INSERT INTO Recipe (Title,Category, Direction) VALUES
("
            + tt +", '"+ this.category +", '"+dd+"'");
    }
}

```

```

String IDquery = ("SELECT Rec_ID FROM Recipe");
    ResultSet rs1 = stmt.executeQuery(IDquery);
    rs1.next();
    do{
        String Rec_ID1=rs1.getString("Rec_ID");
        this.Rec_ID =Integer.parseInt(Rec_ID1);
    } while(rs1.next());
    }
    catch (Exception e) {}
} // end method

// -----
// Import a html recipe file into database , return 0 if import successfully
// -----

public int import_file(FileReader fr)
{
    int i =0;
    this.fr=fr;

// * Firstly, parse the html file into text file , this.f "RecipeFile.txt" was overwritten by the
text recipe file * //
    parserHtml();
    if (!extract())
        return InvalidRecipe;
    if (sameRecipe())
        return RecipeExist;
    storeRecipeTable();
    storeIngredient();
    setChanged();
    notifyObservers();
    return i;
} // end method

// -----
// Inner class Outliner inherit HTMLEditorKit.ParserCallback
// override handleStartTag(), handleEndTag() and handleText() methods
// -----

private class Outliner extends HTMLEditorKit.ParserCallback
{
    private Writer out;
    private String title=null;
    private int level =0;
    private int on = 0;
    private Vector v=new Vector();
    private String line1="";
    public String line = System.getProperty("line.separator", "\r\n");
    public Outliner(Writer out)
    {

```

```

        this.out = out;
    }
    public String get_title()
    {
        return title;
    }
    public Vector get_v()
    {
        return v;
    }

public void handleStartTag(HTML.Tag tag,MutableAttributeSet attributes, int position)
{
    this.level =0;
    if (tag == HTML.Tag.TITLE)
        level = 1;
    if (tag == HTML.Tag.BODY || tag == HTML.Tag.TABLE || tag == HTML.Tag.P
        || tag == HTML.Tag.HR || tag == HTML.Tag.DIV)
    {
        v.add (this.line1);
        v.add(this.line);
        this.line1="";
    }
    else if ( tag == HTML.Tag.BR || tag == HTML.Tag.LI)
        this.on = 1;
    else if (tag == HTML.Tag.TR)
        this.line1="";
        try{out.flush();}
        catch (IOException e) {}
} // end method

public void handleEndTag(HTML.Tag tag, int position)
{
    if (tag == HTML.Tag.TR )
    {
        this.on=2;
        v.add(this.line1);
        line1="";
    }
    else if (tag == HTML.Tag.BODY || tag == HTML.Tag.TABLE ||
        tag == HTML.Tag.P || tag == HTML.Tag.UL)
        v.add(this.line);
        // work around bug in the parser that fails to call flush
        if (tag == HTML.Tag.HTML) this.flush( );
    }

public void handleText(char[] text, int position)
{
    String s = new String(text);
    if (this.level ==1)

```

```

        this.title= MyUtil.correctTitle(s);
    else{
        if (this.on==1)
            v.add(s); this.on = 0;
        else this.line1 = this.line1+" "+s;
        } // end else
    try {out.flush( );} // end try
    catch (IOException e) {}
} // end method

public void flush( )
{
    try {out.flush( );}
    catch (IOException e) {}
} // end method
} // end class
} // end all

```


Structural Test Table

Structural Test -- ModifyRecipe class: insert method

Table of test cases

Choice	Input data set	Input property
1 once	A	there isn't any recipe record in the Recipe table
1 more than once	B	there is at least one recipe record in the Recipe table
2 zero time	A	there isn't any recipe record in the Recipe table
2 once	C	there is only one recipe record in the Recipe table
2 more than once	B	there are more than one recipe records in the Recipe table
3 true	D	the insert recipe has already existed in the Recipe table
3 false	E	the insert recipe hasn't existed in the Recipe table
4 once	A	there isn't any recipe record in the Recipe table
4 more than once	B	there is at least one recipe record in the Recipe table
5 zero time	no available	there isn't any item of recipe ingredient
5 once	F	there is only one item of recipe ingredient
5 more than once	G	there are more than one items of recipe ingredient
6 true	H	the ingredient description is empty
6 false	I	the ingredient description isn't empty

Table of input data sets

Input data set	Contents	Output
A	Recipe table is empty	return 0
B	insert one or more records into Recipe table	return 1 or 2, 3 ...
C	insert one record into Recipe table	return one direction of recipe
D	repeat insert one same records into Recipe table	return false
E	insert new record into Recipe table	return true
F	insert one item of ingredient	add one ingredient record into Ingredient table
G	insert more than one item of ingredient	add more than one ingredients into Ingredient table
H	insert empty ingredient description	exit from the loop
I	insert some ingredient description	run in the loop

Structural Test -- ModifyRecipe class: edit method

Table of test cases

Choice	Input data set	Input property
1 true	A	uncompleted recipe data
1 false	B	completed recipe data
2 zero time	no available	there isn't any item of recipe ingredient
2 once	C	there is only one item of recipe ingredient
2 more than once	D	there are more than one items of recipe ingredient
3 true	E	the ingredient description is empty
3 false	F	the ingredient description isn't empty

Table of input data sets

Input data set	Contents	Output
A	insert empty direction or title	return false
B	insert a completed recipe data	return true
C	insert one item of ingredient	add one ingredient record into Ingredient table
D	insert more than one item of ingredient	add more than one ingredients into Ingredient table
E	insert empty ingredient description	exit from the loop
F	insert some ingredient description	run in the loop

Structural Test -- ModifyRecipe class: searchRecipe method

Table of test cases

Choice	Input data set	Input property
1 true	A	category seleted
1 false	B	category unseleted
2 true	C	title uninserted
2 false	D	title inserted
3 true	E	ingredient uninserted
3 false	F	ingredient inserted
4 zero time	G	no record in Recipe table
4 once	H	only one record in Recipe table
4 more than once	I	more than one record in Recipe table
5 zero time	G	no record in Recipe table
5 once	H	only one record in Recipe table
5 more than once	I	more than one record in Recipe table
6 true	J	input ingredient is matched
6 false	K	input ingredient isn't matched
7 true	E	ingredient uninserted
7 false	F	ingredient inserted
8 zero time	G	no record in Recipe table
8 once	H	only one record in Recipe table
8 more than once	I	more than one record in Recipe table
9 true	L	input title is matched
9 false	M	input title isn't matched
10 zero time	G	no record in Recipe table
10 once	H	only one record in Recipe table
10 more than once	I	more than one record in Recipe table
11 true	N	both title and ingredient is matched
11 false	O	either title or ingredient isn't matched
12 true	C	title uninserted
12 false	D	title inserted
13 true	E	ingredient uninserted
13 false	F	ingredient inserted
14 zero time	G	no record in Recipe table
14 once	H	only one record in Recipe table
14 more than once	I	more than one record in Recipe table
15 zero time	G	no record in Recipe table
15 once	H	only one record in Recipe table
15 more than once	I	more than one record in Recipe table
16 true	J	input ingredient is matched
16 false	F	input ingredient isn't matched
17 true	E	ingredient uninserted

17 false	F	ingredient inserted
18 zero time	G	no record in Recipe table
18 once	H	only one record in Recipe table
18 more than once	I	more than one record in Recipe table
19 true	J	input title is matched
19 false	D	input title isn't matched
20 zero time	G	no record in Recipe table
20 once	H	only one record in Recipe table
20 more than once	I	more than one record in Recipe table
21 true	D	both title and ingredient is matched
21 false	F	either title or ingredient isn't matched

Table of input data sets

Input data set	Contents	Output
A	category = c title = null ingredient = null no record in Recipe table	no recipe is matched
B	category = null title = null ingredient = null no record in Recipe table	no recipe is matched
C	category = c title = t ingredient = null no record in Recipe table	no recipe is matched
D	category = c title = null ingredient = i no record in Recipe table	no recipe is matched
E	category = c title = t ingredient = i no record in Recipe table	no recipe is matched
F	category = c title = null ingredient = null many records in Recipe table	recipe is matched with category = c
G	category = c title = t ingredient = i many records in Recipe table	recipe is matched with category = c title contains 't', ingredient contains 'i'
H	category = null title = t ingredient = i many records in Recipe table	recipe is matched with title contains 't', ingredient contains 'i'
I	category = null	recipe is matched with

	title = null ingredient = i many records in Recipe table	ingredient contains 'l'
J	category = null title = t ingredient = null many records in Recipe table	recipe is matched with title contains 't'
K	category = c title = null ingredient = i many records in Recipe table	recipe is matched with category = c ingredient contains 'l'
J	category = c title = t ingredient = null many records in Recipe table	recipe is matched with category = c title contains 't'

Structural Test -- ExtractInformation class: **ExtractCategory** method

Table of test cases

Choice	Input data set	Input property
1 true	A	invalid recipe file
1 false	B	valid recipe file
2 zero time	C	no record in Recipe table
2 once	D	only one record in Recipe table
2 more than once	D	more than one record in Recipe table
3 zero time	E	no record in Recipe table
3 once	F	only one record in Recipe table
3 more than once	G	more than one record in Recipe table
4 true	A	recipe already exists
4 false	B	recipe not exists
5 zero time	C	no record in Recipe table
5 once	D	only one record in Recipe table
5 more than once	E	more than one record in Recipe table
6 zero time	F	no record in Recipe table
6 once	G	only one record in Recipe table
6 more than once	E	more than one record in Recipe table
7 zero time	F	no record in Recipe table
7 once	H	only one record in Recipe table
7 more than once	I	more than one record in Recipe table
8 zero time	J	buffer reader empty
8 once	A	only one line in buffer reader
8 more than once	B	more than one line in buffer reader
9 zero time	C	no empty line
9 once	D	only one empty line
9 more than once	E	more than one empty line
10 true	F	buffer reader not empty
10 false	G	buffer reader empty
11 zero time	H	no empty line
11 once	I	only one empty line
11 more than once	J	more than one empty line
12 zero time	K	no empty line
12 once	L	only one empty line
12 more than once	A	more than one empty line
13 zero time	B	no empty line

13 once	C	only one empty line
13 more than once	D	more than one empty line
14 true	E	no empty line
14 false	E	empty line
15 zero time	F	no empty line
15 once	F	only one empty line
15 more than once	H	more than one empty line
16 zero time	I	empty string
16 once	J	only one character
16 more than once	K	more than one characters
17 true	L	character '('
17 false	G	not character '('
18 true	A	character '('
18 false	B	not character '('
19 zero time	C	one token
19 once	E	two tokens
19 more than once	D	more than two tokens
20 zero time	D	empty string
20 once	E	only one character
20 more than once	F	more than one characters
21 true	G	character ')'
21 false	H	not character ')'
22 zero time	I	two tokens
22 once	J	three tokens
22 more than one	K	more than three tokens
23 zero time	E	no unit word in database
23 once	F	only unit word in database
23 more than once	G	more than one unid word in database
24 true	H	unit word
24 false	I	not unit word
25 zero time	J	one token
25 once	K	two tokens
25 more than once	L	more than two tokens
26 true	G	number or character ')'
26 false	H	neither number nor character ')'
27 true	G	neither number nor character ')'
27 false	H	number or character ')'
28 true	E	character ')'

28 false	F	not character ')'
29 true	G	more than one characters
29 false	H	no or only one character
30 zero time	J	empty string
30 once	K	only one character
30 more than once	E	more than one characters
31 true	F	character ')'
31 false	G	not character ')'
32 true	H	not character ')'
32 false	I	character ')'
33 zero time	J	one token
33 once	K	two tokens
33 more than once	L	more than two tokens
34 zero time	G	empty string
34 once	H	only one character
34 more than once	G	more than one characters
35 true	H	character ')'
35 false	E	not character ')'
36 zero time	F	one token
36 once	J	two tokens
36 more than once	K	more than two tokens
37 zero time	E	no unit word in database
37 once	F	only unit word in database
37 more than once	G	more than one unid word in database
38 zero time	H	empty string
38 once	I	only one character
38 more than once	J	more than one characters
39 true	K	number or character ')'
39 false	L	neither number nor character ')'
40 true	G	neither number nor character ')'
40 false	H	number or character ')'
41 zero time	G	one token
41 once	H	two tokens
41 more than once	E	more than two tokens
42 zero time	F	empty string
42 once	J	only one character
42 more than once	K	more than one characters
43 true	E	unit word

43 false	F	not unit word
44 zero time	G	one token
44 once	H	two tokens
44 more than once	I	more than two tokens
45 true	J	number or character ')''
45 false	K	neither number nor character ')''
46 true	L	number
46 false	G	not number
47 zero time	H	one token
47 once	G	two tokens
47 more than once	H	more than two tokens
48 zero time	E	no unit word in database
48 once	F	only unit word in database
48 more than once	H	more than one unid word in database
49 true	G	unit word
49 false	H	not unit word
50 zero time	E	one token
50 once	F	two tokens
50 more than once	J	more than two tokens
51 true	K	number or character ')''
51 false	E	neither number nor character ')''
52 zero time	F	one token
52 once	G	two tokens
52 more than once	H	more than two tokens
53 zero time	H	no unit word in database
53 once	G	only unit word in database
53 more than once	H	more than one unid word in database
54 true	E	unit word
54 false	F	not unit word
55 zero time	J	one token
55 once	K	two tokens
55 more than once	E	more than two tokens
56 true	F	number or character ')''
56 false	G	neither number nor character ')''
57 true	H	neither number nor character ')''
57 false	H	number or character ')''
58 zero time	G	one token
58 once	H	two tokens

58 more than once	E	more than two tokens
59 zero time	F	no number
59 once	J	one number
59 more than once	K	two numbers
60 true	E	number
60 false	F	not number
61 true	G	not number
61 false	H	number
62 zero time	H	no unit word
62 once	G	one unit word
62 more than once	H	two unit words
63 true	E	unit word
63 false	F	not unit word
64 true	J	not unit word
64 false	K	unit word
65 zero time	E	no number
65 once	F	one number
65 more than once	G	two numbers
66 zero time	H	no unit word
66 once	J	one unit word
66 more than once	K	two unit words
67 zero time	E	no ingredient description
67 once	F	one ingredient description
67 more than once	G	more than one ingredient description

Table of input data sets

Input data set	Contents	Output
A	empty file	return InvalidRecipe (-2)
B	one record in Recipe table , imported recipe is same as this one record	return RecipeExist (-1)
C	first word is number second word is unit the rest word is ingredient description	return 1
D	first word isn't number the rest word is	return 1

	ingredient description	
E	first word is number second word is number third word is unit word the rest word is ingredient description	return 1
F	first word is number second word is ajective word, third word is unit word, the rest word is ingredient description	return 1
G	first word is number second word is number third word is ajective word, forth word is unit rest word is ingredient description	return 1
H	first word contains (', second word contains ') rest word is ingredient description	return 1
I	first word is number and with '(', second word contains ')', the rest word is ingredient description	return 1
J	first word is number and with '(', second word contains ')',third word is unit word, the rest word is ingredient description	return 1
K	first word is number and with '(', second word contains ')', third word is ajective word forth word is unit word rest word is ingredient	return 1

	description	
L	first word is number rest word is ingredient description	return 1

Structural Test -- ExtractInformation class: **ExtractCategory** method

Table of test cases

Choice	Input data set	Input property
1 zero time	A	no record in Recipe table
1 once	B	only one record in Recipe table
1 more than once	B	more than one record in Recipe table
2 true	B	title contains key material
2 false	C	title not contains key material
3 zero time	A	no record in Recipe table
3 once	B	only one record in Recipe table
3 more than once	B	more than one record in Recipe table
4 true	D	direction contains key material
4 false	E	direction not contains key material

Table of input data sets

Input data set	Contents	Output
A	empty Recipe table	return 'Others'
B	title = 'Onion Beef' (Beef is key material)	return 'Beef'
C	title = 'mom's best' direction contains chichen'	return 'Chicken'
D	title = 'mom's best' direction not contains any key materials	return 'Others'

Structural Test -- ExtractInformation class: **ExtractDirection** method

Table of test cases

Choice	Input data set	Input property
1 zero time	A	buffer reader empty
1 once	B	only one line in buffer reader
1 more than once	B	more than one line in buffer reader
2 zero time	C	no empty line
2 once	D	only one empty line
2 more than once	D	more than one empty line
3 zero time	A	buffer reader empty
3 once	B	only one line in buffer reader
3 more than once	B	more than one line in buffer reader
4 true	C	no empty line
4 false	D	empty line exists
5 zero time	D	empty line exists
5 once	D	only one empty line
5 more than once	D	more than one empty line
6 zero time	A	buffer reader empty
6 once	B	only one line in buffer reader
6 more than once	B	more than one line in buffer reader
7 true	D	empty line exists
7 false	C	no empty line

Table of input data sets

Input data set	Contents	Output
A	empty file	return null
B	file isn't empty but without keyword	return null
C	file isn't empty and with keyword but without direction	return null
D	file isn't empty and with keyword and direction	return direction

Structural Test -- ExtractInformation class: **ValidRecipe** method

Table of test cases

Choice	Input data set	Input property
1 true	A	buffer reader not empty
1 false	B	buffer reader empty
2 zero time	C	keyword1 appear
2 once	D	keyword1 not appear once
2 more than once	D	keyword1 not appear more than once
3 zero time	B	buffer reader empty
3 once	A	buffer reader has only one line
3 more than once	A	buffer reader has more than one line
4 true	C	keyword1 appear
4 false	D	keyword1 not appear
5 true	E	keyword2 appear
5 false	F	keyword2 not appear
6 zero time	A	no empty line
6 once	B	only one empty line
6 more than once	B	more than one empty line
7 true	A	buffer reader not empty
7 false	B	buffer reader empty
8 zero time	B	more than one empty line
8 once	B	only one empty line
8 more than once	C	no empty line
9 zero time	D	no ingredient
9 once	C	only one item of ingredient
9 more than once	D	more than one item of ingredient
10 zero time	E	no direction
10 once	C	only one line of direction
10 more than once	D	more than one line of direction

Table of input data sets

Input data set	Contents	Output
A	empty file	Invalid Recipe File
B	file without ingredient	Invalid Recipe File

C	file with keyword1 but without ingredient	Invalid Recipe File
D	file with keyword1 and ingredient but without keyword2	Invalid Recipe File
E	file with keyword1 and ingredient and key word2 but without direction	Invalid Recipe File
F	file with keyword1 and ingredient and key word2 and direction	Valid Recipe File

Structural Test -- ExtractInformation02 class: **Extract** method

Table of test cases

Choice	Input data set	Input property
1 zero time	A	empty buffer
1 once	B	one line in buffer reader
1 more than once	B	more than one line in reader
2 zero time	B	empty line
2 once	B	one token
2 more than once	B	more than one token
3 true	A	empty buffer
3 false	B	not empty buffer
4 true	B	invalid paragraph
4 false	C	valid paragraph
5 zero time	no available	empty ingredient
5 once	C	one item in ingredient
5 more than once	C	more than one item in ingredient
6 zero time	A	empty line
6 once	B	one token
6 more than once	B	more than one token
7 true	B	ingredient word
7 false	C	not ingredient word
8 zero time	A	empty buffer
8 once	B	one line in buffer reader
8 more than once	B	more than one line in reader
9 true	C	contain keyword
9 false	D	not contain keyword
10 zero time	A	empty buffer
10 once	B	one line in buffer reader
10 more than once	B	more than one line in reader
11 true	B	contain keyword
11 false	C	not contain keyword
12 true	D	empty direction
12 false	D	not empty direction

Table of input data sets

Input data set	Contents	Output
----------------	----------	--------

A	empty file	return false
B	file without ingredient	return false
C	file with ingredient but without direction	return false
D	file with ingredient and direction	return true

Structural Test -- ExtractInformation02 class:ingredientParagraph method

Table of test cases

Choice	Input data set	Input property
1 zero time	no available	empty line
1 once	A	one line
1 more than once	B	more than one line
2 true	B	ingredient line
2 false	A	not ingredient line
3 zero time	A	empty line
3 once	B	ingredient line
3 more than once	A	not ingredient line
4 true	B	ingaredient paragraph
4 false	A	not ingredient paragraph

Table of input data sets

Input data set	Contents	Output
A	paragraph without ingredient	return false
B	paragraph with ingredient	return true

Structural Test -- ExtractInformation02 class:get_category method

Table of test cases

Choice	Input data set	Input property
1 zero time	A	empty Material table
1 once	B	only one record Material table
1 more than once	B	more than one record in Material table
2 true	B	title contains keyword of material
2 false	C	title not contains keyword
3 true	C	title not contains keyword
3 false	B	title contains keyword of material
4 zero time	C	empty Material table
4 once	A	only one record Material table
4 more than once	B	more than one record in Material table
5 true	C	direction contains keyword of material
5 false	C	direction not contain keyword

Table of input data sets

Input data set	Contents	Output
A	title not contain keyword of material, direction not contain keyword of material	category = 'Others'
B	title = "Onion Beef"	category = 'Beef'
C	title not contain keyword of material, direction contains "chicken"	category = 'Chichek'

Structural Test -- ExtractInformation02 class:storeIngredient method

Table of test cases

Choice	Input data set	Input property
1 zero time	A	empty line
1 once	B	one line
1 more than once	B	more than one line
2 zero time	A	empty line
2 once	B	one token
2 more than once	B	more than one token
3 true	C	number
3 false	D	not number
4 zero time	D	empty
4 once	D	one element
4 more than once	D	two elements
5 true	E	unit word
5 false	F	not unit word
6 true	D	number
6 false	E	not number
7 zero time	A	empty
7 once	B	one element
7 more than once	B	two elements
8 true	D	unit word
8 false	E	not unit word
9 zero time	A	empty
9 once	B	one element
9 more than once	B	two elements

Table of input data sets

Input data set	Contents	Output
A	only one item of ingredient	add one record in Ingredient table
B	more than one item of ingredient	add more than one record in Ingredient table
C	first token is number rest is ingredient	add number in Quantity field, add rest ingredient in Ingredient field
D	first token is number	add number in Quantity field,

	second is unit word rest is ingredient	add unit in Unit field, add rest ingredient in Ingredient field
E	first token is number second is number third is unit word rest is ingredient	add first and second number in Quantity field, add unit in Unit field, add rest ingredient in Ingredient field,
F	first token is number second token is number third is ajective word fouth is unit word rest is ingredient	add first and second number in Quantity field, add ajective and unit word in Unit field, add rest ingredient in Ingredient field