Even Semester (2024)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Algorithm and Programming

Final Project

Student Information:

Surname	**:** Rahyang
Given Name	**:** Fadhillah Haidar Rahyang
Student ID	**:** 2702337211

Course Code	**:** COMP6699001	**Course Name**	**:** Object Oriented Programming
Class	**:** L2BC	**Lecturer**	**:** Jude Joseph Lamug Martinez, MCS

Type of Assignment **:** Final Project Report

Submission Pattern:

Due Date	**:** 10 June 2024
Submission Date	**:** 09 June 2024

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.

2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.

3. The above information is complete and legible.

4. Compiled pages are firmly stapled.

5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.
6.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been notified and accepted in advance

Signature of Student:

Fadhillah Haidar Rahyang

Table of Contents

Table of Contents

A. Introduction

1. **Background**

   Students are expected to make a program with the purpose of solving a problem. The goal of this project is to apply all the knowledge of object-oriented programming gained from the past lectures, using java as the programming language, and formulate a program that also include materials outside of class.

   Initially, the plan was to make a game however, after giving some thoughts about the problem to solve, I made an application that helps one of user's daily activities, cooking. The app includes the use of java swing which is responsible for creating a window of the application and the use of dat files to save the user's information locally in their device.

2. **Problem Identification**

   From personal experience, most of the people that follow a recipe to cook the dish is following a recipe from a physical book. A physical book can only do so much as to just show us the printed notes it has. Additionally, it has multiple limitations that are all addressed by the internet. Though there are recipes stored online to follow, there is one slight problem that some people still face today.

   One of the requirements that needs to be met when cooking a dish is the ingredients. For a dish to be cooked in the first place, the required ingredients for that dish needs to be available and the quantity of each ingredient should also be sufficient. The problem lies in the requirements of the dish, specifically the ingredients. Sometimes, people tend to be forgetful and when they plan to cook a recipe, they realise after going grocery shopping, they don't have enough ingredients, and making another trip to the market is tedious.

Therefore, I have come up with a solution that not only stores recipes locally on their devices, but also a list of ingredients that keeps track of the quantities of each ingredient. After they cooked up a dish, they will press a button on the application that decrements the quantity of all the ingredients from that recipe. Subsequently, when they are willing to go shopping, they can check the list of ingredients and restock accordingly. Additionally, they can check the recipe they are planning to cook and it will indicate which ingredients are enough and which aren't. This ensures better time management and keeps everything tidy.

B. Project Specification

1. Application Name

- E-Cooking+
- E states that it is a virtual app
- The '+' sign indicates additional features that a normal cookbook would not have

2. Application Concept

- "E-Cooking+" is an application that serves as a virtual cookbook for users to use while cooking. Users will be able to select recipes that are already available in the list and, if needed, can also add their own custom recipes into the list.
- There are 3 sections that split the application into 3 parts, the recipe panel, the main page panel, and the ingredients panel. The recipe panel will be used to store information of every existing recipe and newly added recipes. The main page panel will display the details of a selected recipe. Then the ingredients panel is divided into to sections, the top panel which will be the fridge and the bottom panel which will be the shelf, both storing ingredients.
- When a recipe is selected, it will highlight certain ingredients from the ingredient panels. The ingredient text will be coloured green if it's available and is sufficient according to the recipe, but will be coloured red if it's available and not enough in quantity. When the user finishes

cooking, they can click on a button that decrements the necessary ingredients.

3. **Application Flow Summary**
    - Upon opening the application, the user can either select an existing recipe from the panel, add a completely new recipe or remove a recipe if needed.

    - When a recipe is selected, the details of that recipe will be displayed on the main page panel, which includes the recipe name at the top of the page, the required ingredients as well as the instructions below it.
    - Then the users can check the ingredients panel and see which ingredients are enough, lacking in quantity, and missing.
    - After the user is done cooking their dish, they can click the ‘cook’ button here and this decrements the quantity of each ingredient from the ingredient panels with the quantities from the recipe, keeping track of the ingredients that are available in the household.
    - There are also options to manually increase and decrease the quantity of an ingredient when needed.

4. **Application Objective**
    - To store recipes locally in the user’s device and keep track of the ingredients available in their household.

5. **Application Display**
    - The app utilises java swing utilities to display the whole application. The window is split into 3 parts, the recipe panel, the main page panel, and the ingredients panel.
    - The recipe panel, located at the left of the whole window, has a title named ‘Recipes’ at the top of the panel, a list of recipe names at the center, and the buttons to add, remove, and modify a recipe at the bottom of the panel.
    - The main page panel, located at the center of the whole window, has a title named ‘Main Page’ at the top, an area for texts at the center which is used to display the recipes’ details, and the ‘cook’ button at the bottom.

- Lastly, the ingredients panel which is equally divided into 2 parts, the fridge and shelf panels. At the top of each panel there is a title named 'fridge' and 'shelf' respectively, a list of ingredients at the center of each panel, and buttons to add ingredient, remove ingredient, set quantity to add and remove the ingredient quantity, add quantity, and subtract quantity located at the bottom of the panel.

6. **Application Mechanics**
   - When add and modify recipe buttons are clicked, a pop-up window will appear to allow the user to type in the recipe name, the ingredients required and the instructions to cook it.
   - When remove recipe button is clicked, with the condition that a recipe is already selected, that recipe will be completely removed from the list.
   - If the cook button is clicked, ingredients from the ingredient panels will be decremented according to the selected recipe's ingredient needs.
   - When add ingredient is clicked, a pop-up window will appear to allow the user to type in the name of the ingredient and set its initial quantity.
   - When remove ingredient is clicked, given that an ingredient is selected, it will remove that ingredient.
   - The set quantity there once clicked shows a pop-up window to allow the user enter an integer to set the adding and subtracting quantity functions; default is set to 1.
   - The add and subtract buttons increases and decreases a selected ingredient according to the set value.

7. **Application Physics**
   - Action listeners are added to all buttons to detect mouse clicks and respond accordingly.
   - The way the recipe and ingredients are shown is that each recipe or ingredient is displayed as a rectangular cell in a scrollable pane. Both ingredients and recipes are stored in a JList which is used to display the contents of that JList into the JScrollPane.

8. **Application Input**

- Mouse left button/trackpad: To click every button and select recipe or ingredient.
- Keyboard: To receive user input when typing.

9. Application Output

- Pop-up windows that involve adding or editing something
- Adding, removing, modifying recipe sound effect (Recipe.wav)
- Adding, removing, increasing quantity, decreasing quantity sound effect in fridge (Fridge.wav)
- Adding, removing, increasing quantity, decreasing quantity sound effect in shelf (Shelf.wav)
- Cook sound effect (Cook.wav).
- Error handlers.

10. Application Libraries/Modules

- java.desktop: A java module used to define the AWT and Swing user interface toolkits, and APIs for accessibility, audio, imaging, printing, and JavaBeans. In the application, the packages used from this module include, javax.sound.sampled which is used for playing sound effects, java.awt which is used to set the GUI of the app and its responses, and javax.swing which is used to further customize the GUI of the app and add more functions and features into it.
- java.base: A java module used to define the foundational APIs of the Java SE Platform. The packages used from this module include, java.util which involves using data structures to store data temporarily and for searching purposes, and java.io which is used to store information locally in the user's device.

11. Application Files

- Main.java: This the main file with the class, Main, where the program gets executed
- CookBookGUI.java: This file contains the class, CookBookGUI, which contains all the main necessary methods of the application. Sort of a second main file as it has the most line of codes by far.

- Ingredient.java: Contains the class, Ingredient, which will be used to instantiate the ingredient object to be shown in the ingredient panels
- Recipe.java: This file instantiates the recipe details that are displayed in the main page panel and is used to display the recipe list on the recipe panel.
- RecipeManager.java: This file handles the recipe objects instantiated from the class, Recipe, and handles saving of information locally.
- Storage.java: This file contains the class, Storage, which is used to instantiate a fridge (first storage) and a shelf (second storage), both used to store the ingredient objects.
- IngredientListCellRenderer.java: A file used to override the class, DefaultListCellRenderer, which is used to 'draw' the rectangular cell of each pane. Conditional statements also added to change the text colour of ingredients in the ingredient panels.
- RecipeListCellRenderer.java: Used to override the same class, with different attributes set to it. Used in the recipe pane.
- recipe.dat: Used to store the data for recipes.
- fridge.dat: Used to store the date for ingredients in the fridge.
- shelf.dat: Used to store the data for ingredients on the shelf.
- SoundEffects: A folder that contains all the sound effects necessary for the program.

12. Application Sounds

All the audios used were not made by me

- Cook sound effect
  https://youtu.be/iJe4k2AMOk4?si=76VWN3zNbnd3eQSm

- Fridge sound effect
  https://youtu.be/NyFxzDQ0SwI?si=hiTz3JeJ0H8MLPAC

- Shelf sound effect
  https://youtu.be/3mWhWA9tcrc?si=55_JhtqJA0wf1FCg

- Recipe sound effect

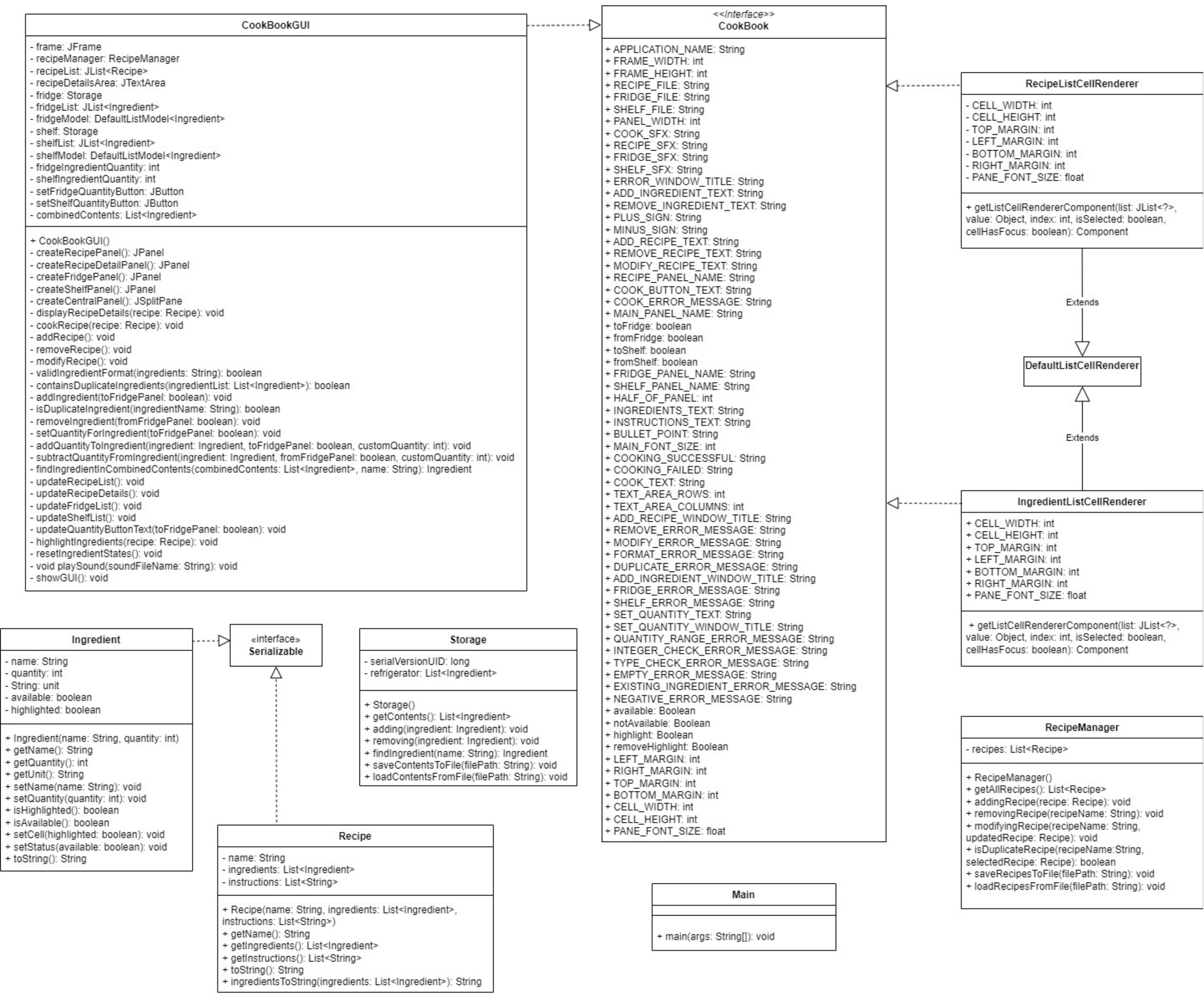
https://youtu.be/k1b2qcJKOTM?si=w1Ca111OnGTKOYK2

13. Application Fonts

- Dialog font used for panel titles, recipe and ingredient lists in each pane
  https://www.myfonts.com/collections/dialog-font-linotype

- Monospace font used in displaying recipe details in the main page panel
  https://www.fontsquirrel.com/fonts/list/classification/monospaced

C. Class Diagram


CookBookGUI
- frame: JFrame
- recipeManager: RecipeManager
- recipeList: JList<Recipe>
- recipeDetailsArea: JTextArea
- fridge: Storage
- fridgeList: JList<Ingredient>
- fridgeModel: DefaultListModel<Ingredient>
- shelf: Storage
- shelfList: JList<Ingredient>
- shelfModel: DefaultListModel<Ingredient>
- fridgeIngredientQuantity: int
- shelfIngredientQuantity: int
- setFridgeQuantityButton: JButton
- setShelfQuantityButton: JButton
- combinedContents: List<Ingredient>
+ CookBookGUI()
- createRecipePanel(): JPanel
- createRecipeDetailPanel(): JPanel
- createFridgePanel(): JPanel
- createShelfPanel(): JPanel
- createCentralPanel(): JSplitPane
- displayRecipeDetails(recipe: Recipe): void
- cookRecipe(recipe: Recipe): void
- addRecipe(): void
- removeRecipe(): void
- modifyRecipe(): void
- validIngredientFormat(ingredients: String): boolean
- containsDuplicateIngredients(ingredientList: List<Ingredient>): boolean
- addIngredient(toFridgePanel: boolean): void
- isDuplicateIngredient(ingredientName: String): boolean
- removeIngredient(fromFridgePanel: boolean): void
- setQuantityForIngredient(toFridgePanel: boolean): void
- addQuantityToIngredient(ingredient: Ingredient, toFridgePanel: boolean, customQuantity: int): void
- subtractQuantityFromIngredient(ingredient: Ingredient, fromFridgePanel: boolean, customQuantity: int): void
- findIngredientInCombinedContents(combinedContents: List<Ingredient>, name: String): Ingredient
- updateRecipeList(): void
- updateRecipeDetails(): void
- updateFridgeList(): void
- updateShelfList(): void
- updateQuantityButtonText(toFridgePanel: boolean): void
- highlightIngredients(recipe: Recipe): void
- resetIngredientStates(): void
- void playSound(soundFileName: String): void
- showGUI(): void
<<Interface>>
CookBook
+ APPLICATION_NAME: String
+ FRAME_WIDTH: int
+ FRAME_HEIGHT: int
+ RECIPE_FILE: String
+ FRIDGE_FILE: String
+ SHELF_FILE: String
+ PANEL_WIDTH: int
+ COOK_SFX: String
+ RECIPE_SFX: String
+ FRIDGE_SFX: String
+ SHELF_SFX: String
+ ERROR_WINDOW_TITLE: String
+ ADD_INGREDIENT_TEXT: String
+ REMOVE_INGREDIENT_TEXT: String
+ PLUS_SIGN: String
+ MINUS_SIGN: String
+ ADD_RECIPE_TEXT: String
+ REMOVE_RECIPE_TEXT: String
+ MODIFY_RECIPE_TEXT: String
+ RECIPE_PANEL_NAME: String
+ COOK_BUTTON_TEXT: String
+ COOK_ERROR_MESSAGE: String
+ MAIN_PANEL_NAME: String
+ toFridge: boolean
+ fromFridge: boolean
+ toShelf: boolean
+ fromShelf: boolean
+ FRIDGE_PANEL_NAME: String
+ SHELF_PANEL_NAME: String
+ HALF_OF_PANEL: int
+ INGREDIENTS_TEXT: String
+ INSTRUCTIONS_TEXT: String
+ BULLET_POINT: String
+ MAIN_FONT_SIZE: int
+ COOKING_SUCCESSFUL: String
+ COOKING_FAILED: String
+ COOK_TEXT: String
+ TEXT_AREA_ROWS: int
+ TEXT_AREA_COLUMNS: int
+ ADD_RECIPE_WINDOW_TITLE: String
+ REMOVE_ERROR_MESSAGE: String
+ MODIFY_ERROR_MESSAGE: String
+ FORMAT_ERROR_MESSAGE: String
+ DUPLICATE_ERROR_MESSAGE: String
+ ADD_INGREDIENT_WINDOW_TITLE: String
+ FRIDGE_ERROR_MESSAGE: String
+ SHELF_ERROR_MESSAGE: String
+ SET_QUANTITY_TEXT: String
+ SET_QUANTITY_WINDOW_TITLE: String
+ QUANTITY_RANGE_ERROR_MESSAGE: String
+ INTEGER_CHECK_ERROR_MESSAGE: String
+ TYPE_CHECK_ERROR_MESSAGE: String
+ EMPTY_ERROR_MESSAGE: String
+ EXISTING_INGREDIENT_ERROR_MESSAGE: String
+ NEGATIVE_ERROR_MESSAGE: String
+ available: Boolean
+ notAvailable: Boolean
+ highlight: Boolean
+ removeHighlight: Boolean
+ LEFT_MARGIN: int
+ RIGHT_MARGIN: int
+ TOP_MARGIN: int
+ BOTTOM_MARGIN: int
+ CELL_WIDTH: int
+ CELL_HEIGHT: int
+ PANE_FONT_SIZE: float
RecipeListCellRenderer
- CELL_WIDTH: int
- CELL_HEIGHT: int
- TOP_MARGIN: int
- LEFT_MARGIN: int
- BOTTOM_MARGIN: int
- RIGHT_MARGIN: int
- PANE_FONT_SIZE: float
+ getListCellRendererComponent(list: JList<?>, value: Object, index: int, isSelected: boolean, cellHasFocus: boolean): Component
Extends
DefaultListCellRenderer
Extends
IngredientListCellRenderer
+ CELL_WIDTH: int
+ CELL_HEIGHT: int
+ TOP_MARGIN: int
+ LEFT_MARGIN: int
+ BOTTOM_MARGIN: int
+ RIGHT_MARGIN: int
+ PANE_FONT_SIZE: float
+ getListCellRendererComponent(list: JList<?>, value: Object, index: int, isSelected: boolean, cellHasFocus: boolean): Component
Ingredient
- name: String
- quantity: int
- String: unit
- available: boolean
- highlighted: boolean
+ Ingredient(name: String, quantity: int)
+ getName(): String
+ getQuantity(): int
+ getUnit(): String
+ setName(name: String): void
+ setQuantity(quantity: int): void
+ isHighlighted(): boolean
+ isAvailable(): boolean
+ setCell(highlighted: boolean): void
+ setStatus(available: boolean): void
+ toString(): String
«interface»
Serializable
Storage
- serialVersionUID: long
- refrigerator: List<Ingredient>
+ Storage()
+ getContents(): List<Ingredient>
+ adding(ingredient: Ingredient): void
+ removing(ingredient: Ingredient): void
+ findIngredient(name: String): Ingredient
+ saveContentsToFile(filePath: String): void
+ loadContentsFromFile(filePath: String): void
Recipe
- name: String
- ingredients: List<Ingredient>
- instructions: List<String>
+ Recipe(name: String, ingredients: List<Ingredient>, instructions: List<String>)
+ getName(): String
+ getIngredients(): List<Ingredient>
+ getInstructions(): List<String>
+ toString(): String
+ ingredientsToString(ingredients: List<Ingredient>): String
Main
+ main(args: String[]): void
RecipeManager
- recipes: List<Recipe>
+ RecipeManager()
+ getAllRecipes(): List<Recipe>
+ addingRecipe(recipe: Recipe): void
+ removingRecipe(recipeName: String): void
+ modifyingRecipe(recipeName: String, updatedRecipe: Recipe): void
+ isDuplicateRecipe(recipeName:String, selectedRecipe: Recipe): boolean
+ saveRecipesToFile(filePath: String): void
+ loadRecipesFromFile(filePath: String): void

D. Essential Algorithms

1. **CookBookConstants.java**

   This file has the necessary constants that are used in multiple other classes in the application. This is also an interface class. The constants are subcategorised into 15 parts, which are as follows,

   a) Application Window

      The constants here are used to define the application name and the screen size upon opening. APPLICATION_NAME is the name of the app, FRAME_WIDTH is the width of the app and FRAME_HEIGHT is the height of the app.

   b) Saving

      Constants here are strings of file paths that are responsible to save data input from users. These involve the use of dat files. RECIPE_FILE stores recipe, FRIDGE_FILE stores ingredients from the fridge panel and SHELF_FILE stores ingredients from the shelf panel.

   c) Panel Width

      PANEL_WIDTH is used to set the width of the recipe panel and the storage panel. The main page panel will adjust accordingly.

   d) Sound Effects

      These are also strings of file paths, but is responsible for the sound effects. There are 4 sound effects which include, COOK_SFX, RECIPE_SFX, FRIDGE_SFX, and SHELF_SFX all responsible for the sound effects according to each naming.

   e) Error Handling Title

      This sets the pop-up window title that appears for every error a user does.

   f) Recipe Panel

      Constants used in the recipe panel. This includes all the texts for the buttons placed in this panel as well as the panel title.

   g) Main Page Panel

      Constants used in the main page panel. There is one constant used for the button text, one for the panel title.

   h) Storage Panel

Constants used in the storage panel, which includes fridge and shelf panels. The constants here set the text for all the buttons in the storage panel, the panel titles for fridge and shelf, and HALF_OF_PANEL is used to split the fridge and shelf panels equally.

i) Storage Separator

The constants here are all boolean values that are used to decide whether the added ingredient is towards the fridge or towards the shelf.

j) Display Recipe Details

These are the constants that will be used to display the recipe details on the main page panel.

k) Cooking

These are the constants used when the cook button is clicked. All of them are used in error handling, COOKING_SUCCESSFUL if the selected recipe is cookable, COOKING_FAILED if it isn't, and COOK_ERROR_MESSAGE if a recipe hasn't been selected.

l) Recipe Manipulation

Here lies the constants that are used when buttons from the recipe panels are clicked which also includes error handling. The first two integers are used to define the area of the input text when adding or modifying an ingredient.

m) Ingredient Manipulation

Used when changes are made to ingredients which includes a lot of error handling, including units of each ingredient.

n) Highlight Ingredients

The boolean values for highlight and removeHighlight are used to decide whether an ingredient should be highlighted. The boolean values for available and notAvailable are used to decide whether the ingredient should be highlighted green or red.

o) Cell Rendering

These constants are used in overriding the DefaultListCellRenderer class which includes the margins, cell area, and cell font size.

2. CookBookGUI.java

This file is sort of the second main file as it contains all of the main methods required for this application to properly function. This class inherits the interface class, CookBookConstants.

a) Constructor

   In the constructor objects are instantiated and used when the program is initially started.
   A new frame object is made to set the application's window,
   A recipe manager object is made to be able to manage the recipes,
   A fridge and shelf object are made to store ingredients,
   setFrame() method to set the frame's attributes,
   File objects are made to allow saving of data,
   loadAllFiles() method to load the files when the program opens,
   resetIngredientStates() method to ensure all ingredients are black,
   - Updating methods to display information once the files are loaded.

b) Creating Panels

   Every panel are instantiated with a new JPanel using the layout, BorderLayout, which will allow panels to placed in specific places of the frame. There is an exception to the storage however, as it uses JSplitPane which splits the JPanels of fridge and shelf into two. The method that includes these locations is the setFrame() method where the recipe panel is placed on the west of the frame, the storage panel on the east of the frame, and the main page panel on the center. The placement of each panel titles also uses the same concept, BorderLayout and placing it north. Every panel has a DefaultListModel which allows management of items displayed by a JList. The JList is in a JScrollPane object that allows vertical and horizontal scrolling when necessary. The JList cell rendering is overridden with each panel's respective cell rendering classes. Buttons are instantiated and added at the bottom of the panel. Lastly, every button is given an action listener to detect mouse clicks and respond accordingly.

c) Displaying Text on Main Page

This mainly utilises string builder which allows appending strings easily into the string builder. The font is also set here to monospaced and bold with a set font size. Then the setText() method is used to set whatever contents are in the string builder, with the condition that this string builder is converted into a string which is why toString() is used.

d) The Cook Button

To put it in short, the way the method works is that it will compare two lists of ingredients and retrieve certain ingredients. The first list is all the ingredients from the storage panel. The second list is the ingredients from the recipe details. A boolean value, hasAllIngredients, is set to decide whether the ingredients are available and proper (the units need to be same). Iterating through the ingredient list of the storage panel, the combinedIngredient object will be used to store the ingredient that are needed. The method findIngredientInCombinedContents() handles the logic in retrieving the necessary ingredients, this is the part where lists are compared. The boolean value unitsMatch will check if the units from the ingredient in the recipe and the storage panels are the same. If either the ingredients aren’t enough or the units are different, hasAllIngredients is set to false and the logic to execute the decrementing of ingredients is not applied. If applied however, it will iterate through the ingredients required, get the quantity of each ingredient and decrement the ingredients from the fridge first, then the shelf.

e) Making Changes to the Recipes

This includes adding a new recipe, removing an existing recipe, and modifying an existing recipe. For adding and modifying recipe methods, they are quite similar; only difference is the initial values of the variables, the addRecipe() variables are set to empty because when adding a recipe, the text fields should be empty. The modifyRecipe() variables are set to whatever was contained in those recipes, saving the user’s input. JOptionPane() is utilised here to receive display a pop-up window and retrieve user input. Once ‘OK’ is clicked, first it checks the formatting, which involves the use of regular expression. The

regular expression used is translated to,
(WS)(W)(WS)(C)(WS)(I)(WS)(C)(WS)(W)(WS)

Legend:	WS = Whitespace
	W = Word
	C = Colon
	I = Integer.

Then it checks for recipe duplication in case an existing recipe of the same name is present. To do this, the names of the recipes in the current pane are iterated and compared with the current name. There was a bug where modifying the recipe didn't work with the same name used because the current name is also iterated and the program thinks of it as a duplicate. A fix to this is simply modifying the conditional statement which will disregard the selected recipe name. Lastly, it checks for duplicate ingredients that are added in the ingredients text field. This utilises the use of Set as Set only allows unique elements.

For removeRecipe(), getsSelectedValue() allows the selection of cells within a JScrollPane(). The conditional statement here checks if an ingredient has already been selected or not, if it is, the element (cell) is removed from the model. Then the text area for the recipe details is set to empty. All methods have a method to save the recipes into the dat file at the end.

f) Making Changes to the Ingredients

This include adding, removing, setting a quantity to modify ingredient quantity, adding quantity, and subtracting quantity. The addIngredient() is similar to how addRecipe() and modifyRecipe() is executed, but with different logic. First it checks if the ingredient name text field is empty. Then checks if the ingredient added is a duplicate of an existing ingredient from the storage panels. Basically, it iterates through the ingredient names from the storage panels and compares it with the current user input. After that, using the try-catch error handling, it checks whether the number input for quantity is an integer. Then it checks if the number is a negative value. Lastly, it checks if the unit text field is empty. Once all the conditions are met, an ingredient object is instantiated and added into one of the storage panels.

For removeIngredient(), similar to how removeRecipe() works, it utilises getSelectedValue() to enable selection of elements. A boolean value is utilised to separate the button response and the selection of fridge and shelf panels. The setQuantityForIngredient() is executed to set a quantity that will be used to either add or decrement the quantity of a selected ingredient. A conditional statement is used to limit the quantity from 1 to 100, 0 doesn't make sense and above 100 is unnecessary. Number format is also checked using try-catch and the method updateQuantityButtonText() updates the text displayed on the quantity button to whatever integer the user inputs.
For addQuantityToIngredient(), the way the ingredient quantity is increased is that the quantity of a selected ingredient is first retrieved, and then added with customQuantity. CustomQuantity is a value passed from the setQuantityForIngredient(). Two variables responsible for setting the numbers are placed outside as global variables. These will be changed according to what was executed in setQuantityForIngredient(). Then for subtractQuantityToIngredient() it's quite similar. The difference is that first it checks if the selected ingredient quantity is more than 0, and when the value of the set quantity is more than the ingredient quantity, the ingredient quantity is subtracted by itself instead of customQuantity value.

g) The Updating Methods
This includes updateRecipeList(), updateRecipeDetails(), updateFridgeList(), updateShelfList(), updateQuantityButtonText(), highlightIngredients() and resetIngredientStates().

Starting off with the first four methods, the first thing that is done is that the panel are cleared completely, removing every information that was displayed. Then the whole information is loaded once again, but with the newly edited one.

In updateQuantityButtonText however, this method utilises the method setText() to change the texts displayed on these buttons.

For highlightIngredients(), the same concept of comparing two lists is also done here, but with the addition of conditional statements that allow colouring of ingredient text. The boolean unitsMatch here also checks for the units. The method setCell() decides if an ingredient should be highlighted or not. Then if the ingredient is equal or more than needed, it is highlighted green, otherwise red. If the ingredient is not needed however, that ingredient's 'setcell()' is false meaning it will not execute the statement, if enough: green else: red. The resetIngredientStates() method is like a the little brother of the previous one as it serves as a reset. Whenever the program loads, to ensure all ingredient text is coloured black this method is used.

h) Others

The rest include the playSound() method as well as the showGUI() method. The playSound() allows for audio playback and is used to give most panel buttons a sound effect. The showGUI() method is just there to set the frame visible, the smallest method here in CookBookGUI.java.

3. **Main.java**

This is the main file of the cookbook application. This is where the program is supposed to run. The CookBookGUI object is instantiated, basically like taking a physical book from a real life library, and the showGUI() method makes the app visible, like opening a physical book.

4. **Ingredient.java**

This file is used to instantiate the ingredient objects that are used throughout the app. The class in this file inherits the interface class, Serializable, which is necessary to allow of saving data locally on the device.

a) Constructor

   There are three parameters in the constructor, the first is name of type string which is used to set the name of an ingredient, then it’s the quantity of type int which is used to set the ingredient quantity, and lastly, it is the unit of type string which sets the unit of the quantity.
   this.name, which defines the ingredient name,
   this.quantity, defines the quantity,
   this.unit, defines the unit,
   this.available, checks the ingredient availability,
   this.highlighted, for highlighting the ingredient text,

b) Getter Methods

   This includes getName() which returns a String, getQuantity() which returns an int, and getUnit() which returns a String. All of these are used to get the corresponding attributes of an ingredient.

c) Setter Methods

   Theres three setter method, setQuantity(), setCell() and setStatus(). The setter method setName() and setUnit() is not used as modifying ingredients does not exist in this app. I forgot to remove setName() and if I did remove it, it will break the serializable class, preventing the data from each panel to load properly. The setQuantity() is used to give changes to ingredient quantity. The setCell() and setStatus() is there to allow highlighting of certain ingredients

d) ToString Method

   This method overrides the toString() method of an object class to prevent it printing the memory location. The method here returns the string, name: quantity unit. For instance, eggs: 10 pieces.

5. **Recipe.java**

This file is responsible for instantiating the recipe objects of this application. Just like the Ingredient class, this also inherits the class, Serializable, to allow saving.

a) Constructor

   It has three parameters, first is the name of type string, then it's the ingredients in the form a list of ingredient objects, last but not least is the instructions in the form of a list of string objects.
   this.name, defines the recipe name,
   this.ingredients, defines the list of ingredients of a recipe,
   this.instructions, defines the steps of cooking a recipe.

b) Getter Methods

   There are three getter methods, all of which are used to retrieve each corresponding attribute of a recipe object. The getName() for recipe name, getIngredients() for the ingredients of a recipe, and getInstructions() to receive the instructions of a recipe.

c) ToString Method

   Overrides the toString() method and simply returns the name of the recipe. To avoid confusion, the getName() method is used in CookBookGUI.java while the toString() method is to prevent printing of memory location.

d) IngredientsToString Method

   This is necessary to actually display the ingredients of a recipe on the main page panel. Everytime an ingredient, with its attributes, are appended into the string builder, \n is appended to set the next ingredient text into a new line.

6. **Storage.java**

   This file is responsible for instantiating the space for the ingredient objects which inherits the interface class,Serializable. The storages made in this application is a fridge and a shelf.

   a) Constructor

There are no parameters in this constructor.

this.contents, used to store the ingredients in an array list.

b) Getter Method

There's only 1 getter method, the getContents() which is used to return all of the ingredients from one of the storage, either shelf or fridge.

c) Modifying The List

The adding() method to add ingredients into the array list and the removing() method to remove an ingredient from the array list.

d) Finding an Ingredient

To find a specific ingredient in the list, the array list is iterated and checks a match with the selected ingredient. If there are not matches, return null, meaning no ingredient is found in the list.

e) Filing Methods

Here lies the method of allowing the function to save data locally. This include saveContentsToFile() and loadContentsFromFile(). FileOutputStream() allows writing of files and combining it with ObjectOutputStream, it makes it possible to write objects into the file.

7. **RecipeManager.java**

This is basically the storage for the recipe objects.

a) Constructor

Same as the storage constructor, there are no parameters here.

this.recipes, used to store recipe objects

b) Getter Method

There's only 1 getter method, the getAllRecipes() which returns the list of all the recipes.

c) Modifying The List

These include 3 methods, the addRecipe(), which is used to add a new recipe, the removeRecipe(), used to remove a selected recipe, and modifyRecipe(), used to modify a selected recipe.

d) Parsing and Duplication Check

For the parsing method, parseIngredients(), the ingredients are first split into parts (single ingredient) and stored in an array. Then for every ingredient pair that exist in the recipe details, it is split once again and this time it splits the ingredient attributes into three parts, the name, quantity and unit. These are used in instantiating the ingredient object.

Then for the duplication check which was aforementioned before, it will check every recipe in the list and compare it with the current user input, with the condition of disregarding the name of the selected recipe.

e) Filing Methods

The way this works is exactly the same like how the storage class executes it. The only difference is that this will lead to a different file, a file for recipe data only.

8. **IngredientListCellRenderer.java**

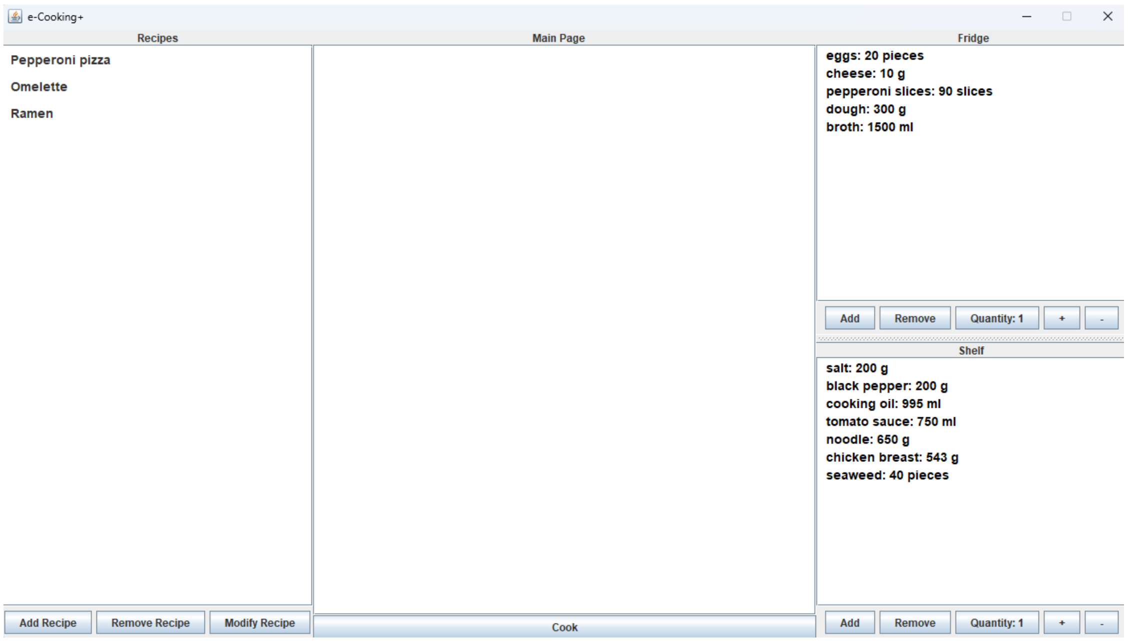
   This class overrides the DefaultListCellRenderer by inheriting the object class., DefaultListCellRenderer. The attributes that are overridden include, the cell area, the cell font size, the cell margin, and the colour of the cell text.

9. **RecipeListCellRenderer.java**

   This also overrides the same class, but the height of the cell is slightly bigger than the ingredient one, and the font size is also slightly bigger. Both cell rendering classes inherits the interface class, CookBookConstants.

E. Screenshots of the Application

1. Application Screen (Upon opening)

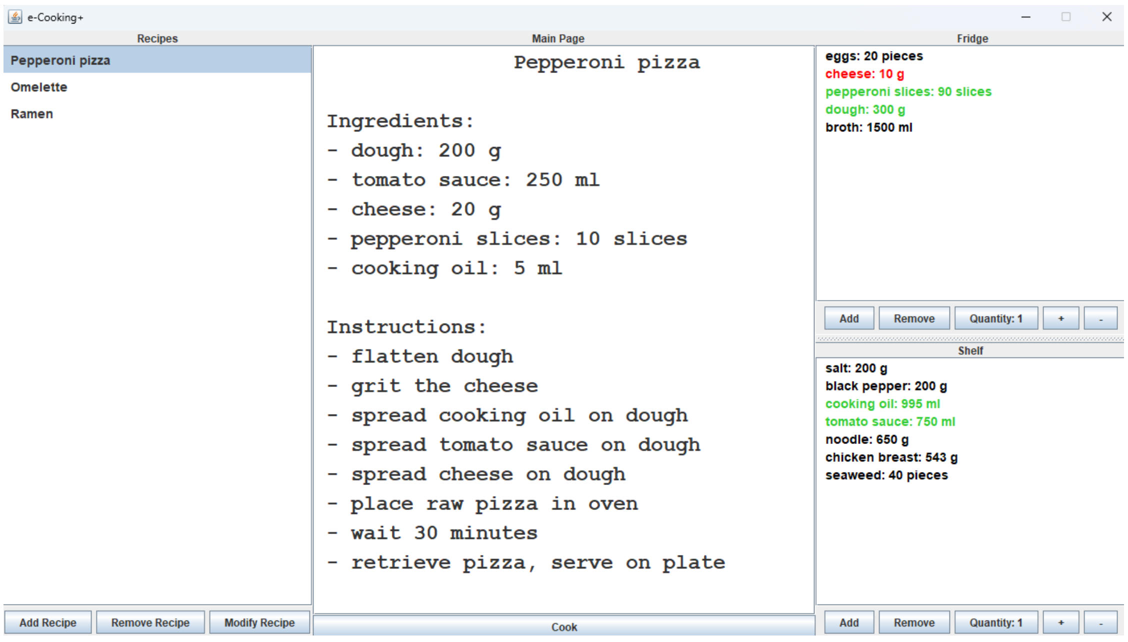


Pepperoni pizza
Omelette
Ramen

eggs: 20 pieces
cheese: 10 g
pepperoni slices: 90 slices
dough: 300 g
broth: 1500 ml

salt: 200 g
black pepper: 200 g
cooking oil: 995 ml
tomato sauce: 750 ml
noodle: 650 g
chicken breast: 543 g
seaweed: 40 pieces

2. Application Screen (When a recipe is selected)



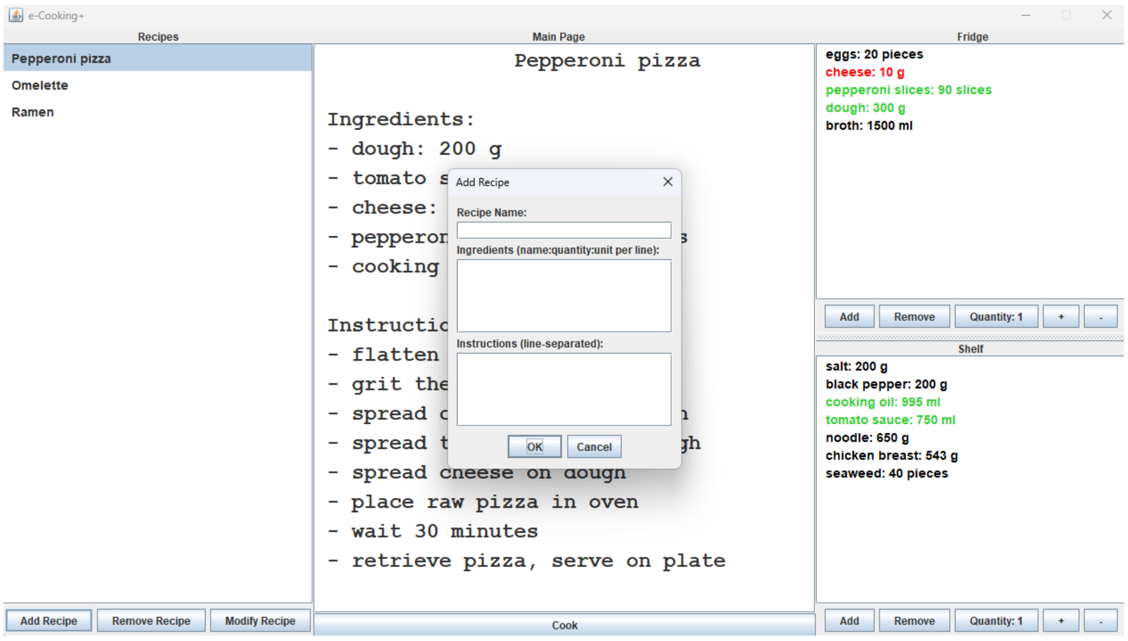
Pepperoni pizza
Omelette
Ramen

```
Pepperoni pizza

Ingredients:
- dough: 200 g
- tomato sauce: 250 ml
- cheese: 20 g
- pepperoni slices: 10 slices
- cooking oil: 5 ml

Instructions:
- flatten dough
- grit the cheese
- spread cooking oil on dough
- spread tomato sauce on dough
- spread cheese on dough
- place raw pizza in oven
- wait 30 minutes
- retrieve pizza, serve on plate
```

eggs: 20 pieces
cheese: 10 g
pepperoni slices: 90 slices
dough: 300 g
broth: 1500 ml

salt: 200 g
black pepper: 200 g
cooking oil: 995 ml
tomato sauce: 750 ml
noodle: 650 g
chicken breast: 543 g
seaweed: 40 pieces

3. Window Pop-up (When add recipe button is clicked)



Pepperoni pizza
Omelette
Ramen

Pepperoni pizza

Ingredients:
- dough: 200 g
- tomato s
- cheese:
- pepperon s
- cooking

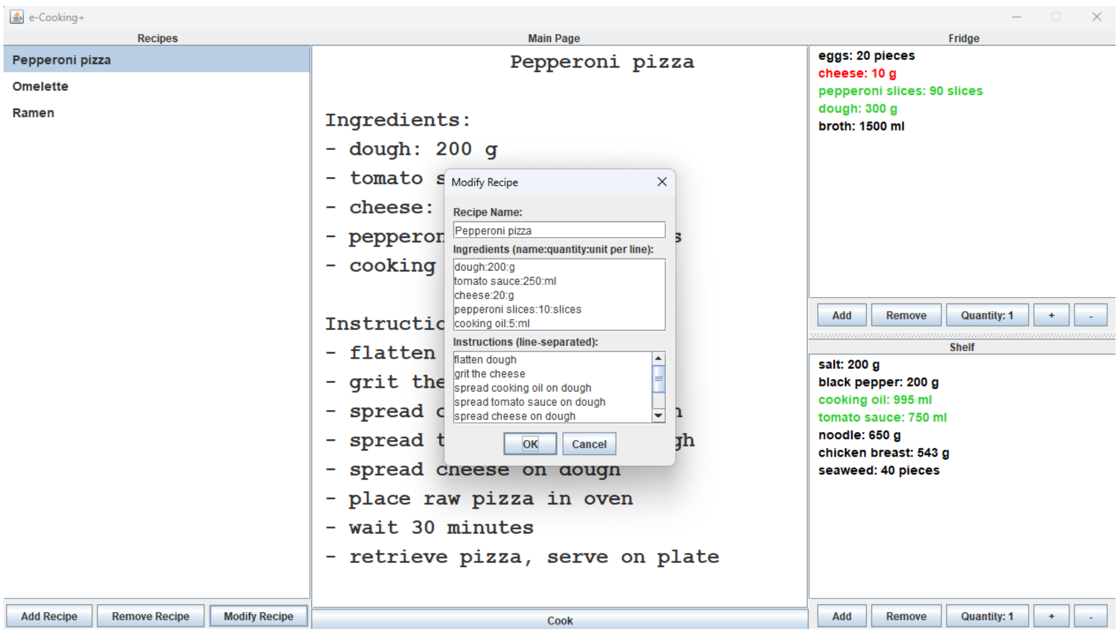
Instructio
- flatten
- grit the
- spread c n
- spread t gh
- spread cheese on dough
- place raw pizza in oven
- wait 30 minutes
- retrieve pizza, serve on plate

Add Recipe
Recipe Name:
Ingredients (name:quantity:unit per line):
Instructions (line-separated):

eggs: 20 pieces
cheese: 10 g
pepperoni slices: 90 slices
dough: 300 g
broth: 1500 ml

salt: 200 g
black pepper: 200 g
cooking oil: 995 ml
tomato sauce: 750 ml
noodle: 650 g
chicken breast: 543 g
seaweed: 40 pieces

4. Window Pop-up (When modify recipe button is clicked)



Pepperoni pizza
Omelette
Ramen

Pepperoni pizza

Ingredients:
- dough: 200 g
- tomato s
- cheese:
- pepperon s
- cooking

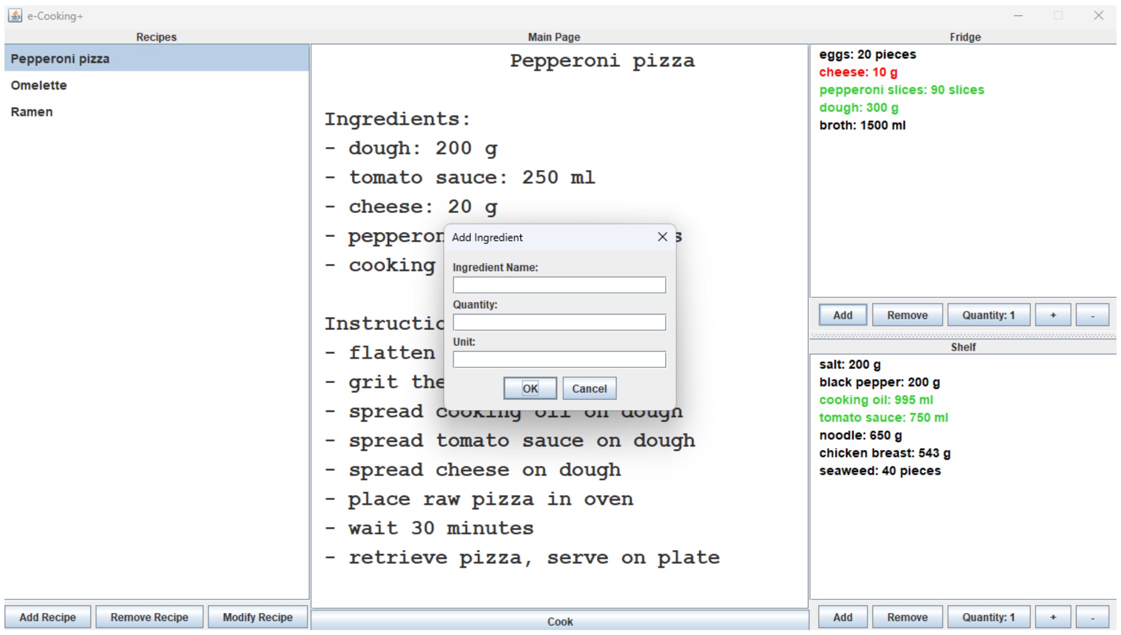
Instructio
- flatten
- grit the
- spread c n
- spread t gh
- spread cheese on dough
- place raw pizza in oven
- wait 30 minutes
- retrieve pizza, serve on plate

Modify Recipe
Recipe Name:
Pepperoni pizza
Ingredients (name:quantity:unit per line):
dough:200:g
tomato sauce:250:ml
cheese:20:g
pepperoni slices:10:slices
cooking oil:5:ml
Instructions (line-separated):
flatten dough
grit the cheese
spread cooking oil on dough
spread tomato sauce on dough
spread cheese on dough

eggs: 20 pieces
cheese: 10 g
pepperoni slices: 90 slices
dough: 300 g
broth: 1500 ml

salt: 200 g
black pepper: 200 g
cooking oil: 995 ml
tomato sauce: 750 ml
noodle: 650 g
chicken breast: 543 g
seaweed: 40 pieces

5. Window Pop-up (When add ingredient button is clicked)



e-Cooking+

Recipes

- Pepperoni pizza
- Omelette
- Ramen

Main Page

```
Pepperoni pizza

Ingredients:
- dough: 200 g
- tomato sauce: 250 ml
- cheese: 20 g
- pepperon                                  s
- cooking

Instructic
- flatten
- grit the
- spread cooking oil on dough
- spread tomato sauce on dough
- spread cheese on dough
- place raw pizza in oven
- wait 30 minutes
- retrieve pizza, serve on plate
```

Add Ingredient

Ingredient Name:

Quantity:

Unit:

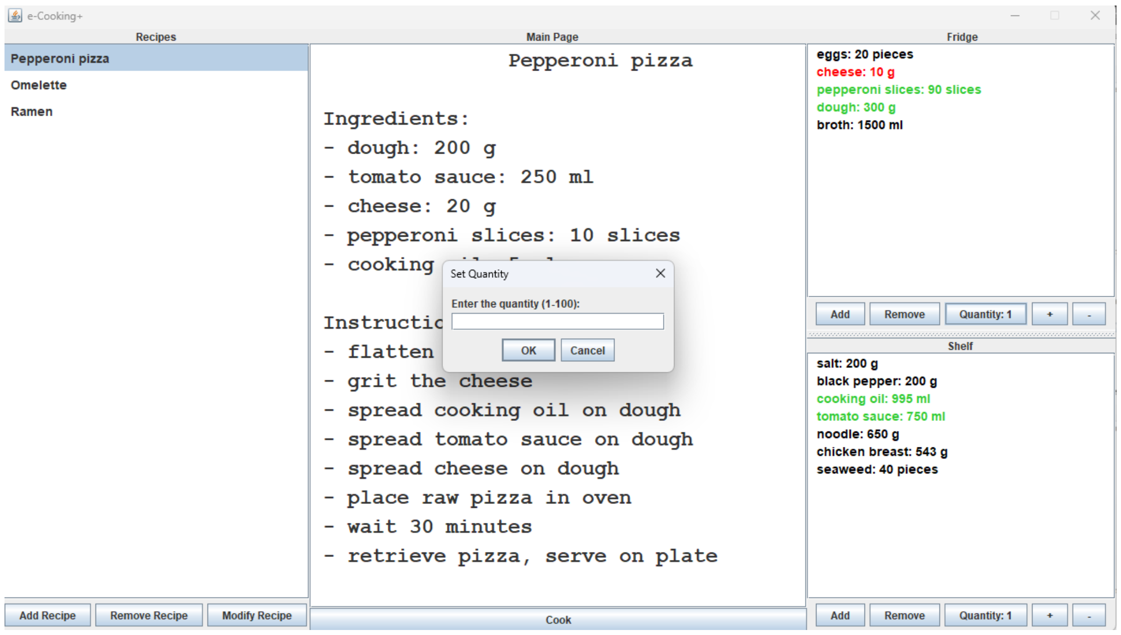
Fridge

- eggs: 20 pieces
- cheese: 10 g
- pepperoni slices: 90 slices
- dough: 300 g
- broth: 1500 ml

Shelf

- salt: 200 g
- black pepper: 200 g
- cooking oil: 995 ml
- tomato sauce: 750 ml
- noodle: 650 g
- chicken breast: 543 g
- seaweed: 40 pieces

6. Window Pop-up (When set quantity button is clicked)



e-Cooking+

Recipes

- Pepperoni pizza
- Omelette
- Ramen

Main Page

```
Pepperoni pizza

Ingredients:
- dough: 200 g
- tomato sauce: 250 ml
- cheese: 20 g
- pepperoni slices: 10 slices
- cooking

Instructic
- flatten
- grit the cheese
- spread cooking oil on dough
- spread tomato sauce on dough
- spread cheese on dough
- place raw pizza in oven
- wait 30 minutes
- retrieve pizza, serve on plate
```

Set Quantity

Enter the quantity (1-100):

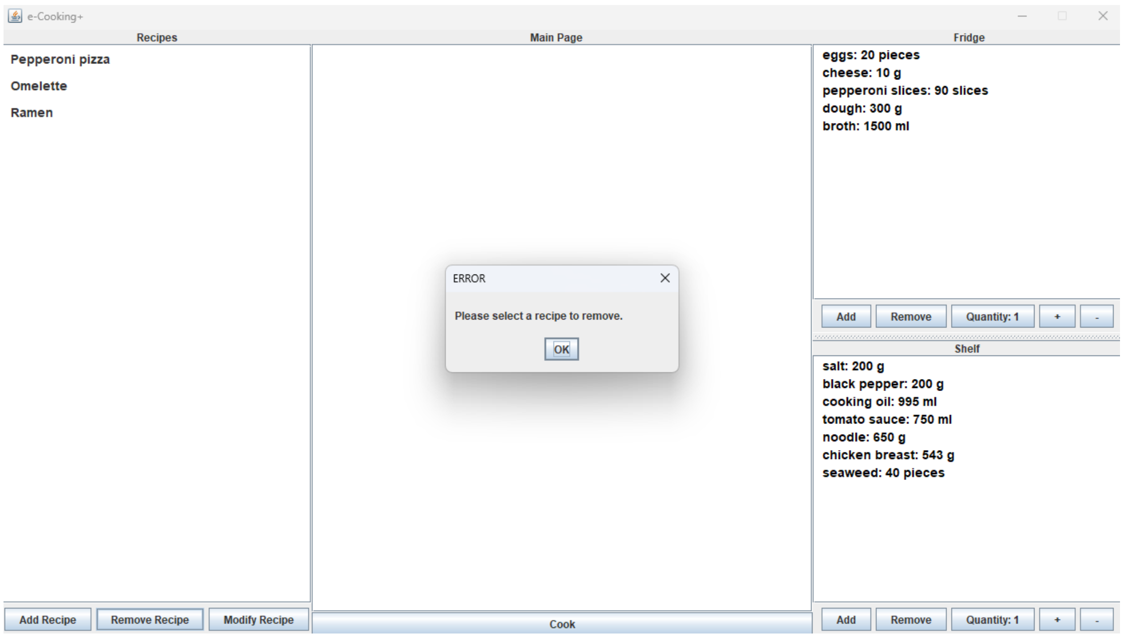
Fridge

- eggs: 20 pieces
- cheese: 10 g
- pepperoni slices: 90 slices
- dough: 300 g
- broth: 1500 ml

Shelf

- salt: 200 g
- black pepper: 200 g
- cooking oil: 995 ml
- tomato sauce: 750 ml
- noodle: 650 g
- chicken breast: 543 g
- seaweed: 40 pieces

7. Error Handling



Pepperoni pizza
Omelette
Ramen

eggs: 20 pieces
cheese: 10 g
pepperoni slices: 90 slices
dough: 300 g
broth: 1500 ml

Please select a recipe to remove.

salt: 200 g
black pepper: 200 g
cooking oil: 995 ml
tomato sauce: 750 ml
noodle: 650 g
chicken breast: 543 g
seaweed: 40 pieces

F. Lessons Learnt/Reflection

For starters, I'd say object-oriented programming specifically is easier in java rather than in python. Despite having much more complex syntax, it is easier to do OOP here than it is in python. Something I also learned is that an application doesn't need to be very complicated, as long as it executes well, achieves the objective, and has sufficient functionalities, it can be much more useful than making a tower defence game (cough, cough) my final project in algorithm and programming course. Not to mention that creating a graphical user interface here is easier than it is in python.

G. Resources

1. Java learning
   https://devdocs.io/openjdk~8_gui/
2. File saving learning
   https://stackoverflow.com/questions/11593725/sending-file-through-objectoutputstream-and-then-saving-it-in-java

3. Music and sound effects
   https://www.y2meta.com/en163