Smart Cookbook

Teemu Romo, 829841

Tietotekniikka, 2020

26.4.2021

1. Overview

The users can easily keep a record of recipes and which ingredients they have available and how much. They can add their own recipes and ingredients and delete them. Users can search recipes by name and allergens (e.g. vegetarian food or gluten-free) recipes which fill all the criteria are shown in the GUI. User can click the “Done” button for the selected recipe, and then the program will ask if you have really done that recipe. If you can’t do that and you try to make that recipe, it will show ingredients that you don’t have at all or enough. The program also shows all the recipes that users can make without needing to go to the store when a user clicks a button called “What recipe(s) I can make?”. When you have all needed ingredients and click the “Done” button, the program will delete all the used ingredients from the library. The program will also convert measurements to their correct values. For example flour is stored in kilograms but recipes use it in liters.

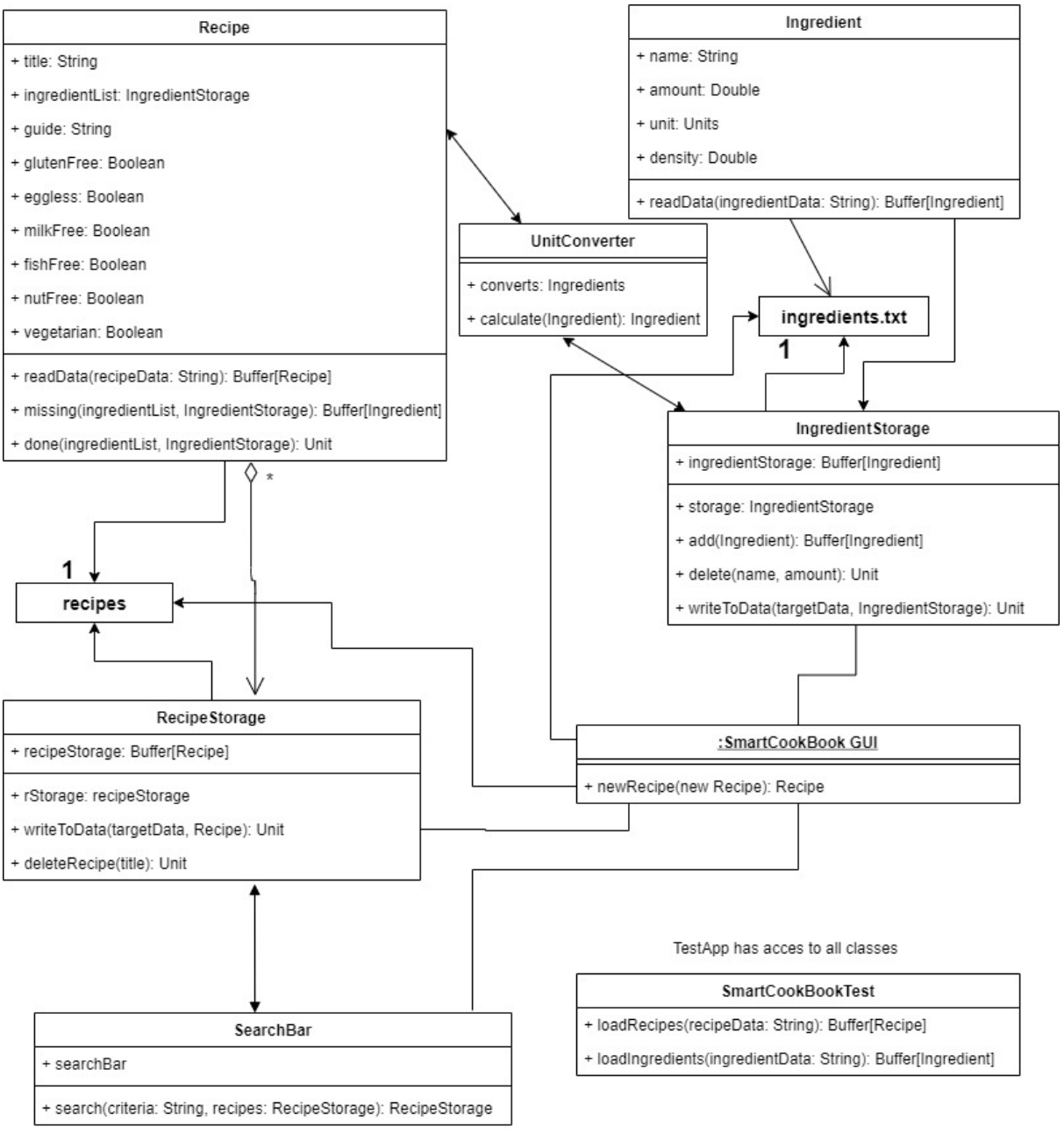
The only major change or shortcoming in the project, I think, was the connection between the recipes and the ingredients. This requirement was difficult to implement and, at a certain stage in the program, it seemed too hard to have that requirement done by the deadline. Also, the editing of recipes is a bit incomplete and in practice it only happens by deleting the old recipe and adding a new recipe with new data / modified data. Of course, the original design of the interface changed a lot when I started making the program, but all the essential components can still be found with a separate implementation.

I’m doing the program on the **Intermediate** level and I think I succeeded on that.

2. User Manual

The program is launched from the SmartCookbook GUI.scala object. The home screen shows the ingredient inventory at the top left. Next down is the addition and removal of ingredients and below this is a recipe store. Recipes can be added from the button in the lower left corner. The gray "area" in the middle is empty when the application starts, but after writing the recipe title to the lowest empty text field on the right and then clicking the "select" button above it makes the recipe visible. The top orange button on the right searches for recipes that can be made with the ingredients in stock. All buttons on the right search, select, delete and done use the text field below them to search, delete, select or make a recipe. When selecting or searching recipes and more than one recipe is the result, only the first recipe on the list is shown. Click the "Add new recipe" button, a window will open where ingredients can be added to the new recipe, the name and instructions for the recipe will be entered, and allergen information will be selected. Then there is a button at the bottom of the window which adds the recipe to the recipe library.

4. Program structure



The structure of the program remained basically the same as in the plan. The only major changes were the move of the writeToData method from the recipe and ingredients to their repositories, the removal of the unit class, and the change from the SmartCookbook test class to the SmartCookbook GUI class.

Recipe class has multiple parameters. It uses recipes to have all those parameters saved. All *Recipes* are in the same data file which can be edited from the UI. *Recipe* has methods **readData** to convert text-file to *Recipe* (and add them straight in the Buffer) and First I thought, UI could have used a text-file for editing the program but I ended up using the program for editing the text-file. **Done** method compares *Recipe*'s parameter ingredientList to class *IngredientStorage* through the *UnitConverter* and after clicking done, it will delete used ingredients from the *IngredientStorage*'s Buffer. **Missing** method will show missing ingredients that are needed to make that recipe.

Ingredient class is using IngredientData text-file. The program only has one file to store all the ingredients. **ReadData** reads the file and forms the *IngredientStorage* from it.

To keep the Ingredient text-file in sync, *IngredientStorage* needs to have a method **wrtiteToData. Storage** method only shows the whole IngredientStorage. After adding (**add**) or deleting (**delete**) the ingredient from the storage the file needs to be written again.

UnitConverter was not simple to implement. I ended up with the conclusion that *UnitConverter* will convert all the ingredients (after clicking done, or deleting ingredients from the storage) which have been inputted in a unit that is different from the one used in the Recipe or storage. I need to use density for making that possible. Users can only choose units from predefined units and I had to configure a "comboBox" for the user interface from which the unit can be selected. Units are all basic units for weight and volume (e.g. kilogram, gram, decilitre, litre, teaspoon, tablespoon, etc.) and pieces will be pieces in the ingredient storage and in the Recipe. To make that happen the “wrong” unit that is used in the recipe is converted using the formula (1) below. If the unit in the recipe is weight and ingredient storage has volume, recipes weight need to convert to the volume. The *UnitConverter* will not do anything if units are the same in recipe and ingredient storage.

RecipeStorage acts similarly to *IngredientStorage*. **Delete** method can delete recipes from the storage, **wrtiteToData** to convert *Recipes* to text-file and **rStorage** for getting the whole recipeStorage to use.

The *SearchBar* class uses method **search** to find recipes that are searched with specific criterias found on the recipe.

SmartCookBook GUI actually includes methods that are missing from other categories, such as adding recipes and how to display recipes that may be made.

5. Algorithms

My program will not basically use any specific algorithms. The *UnitConverter* uses the formula below (1) to calculate unit conversions. At first I thought I would have pre-defined certain ingredients for the user for which I would have pre-defined the densities, but I ended up with a solution where the user enters the ingredient herself, as well as the density.

$$density = \frac{mass}{volume} \quad (1)$$

Formula (1)

6. Data structures

The way that the program will store ingredients and recipes is by using a collection type called Buffer. I think buffers will work well with text files for this purpose as they need to be able to be easily modified. At first I was wondering if I could save the files in code, but the text files became familiar during our course so I ended up with them. At least at this point, I don't think I'm creating my own data structures and I'm using Scala's own. Basically almost all my methods used scala.collection algorithms.

7. Files and Internet access

The program uses normal text files to store ingredients and recipes. The user does not have to edit the text files directly, but the program handles the editing using the user interface and coded methods.

The program does not have internet access.

8. Testing

Contrary to the original testing plan, testing was also done in part using SmartCookbookTest.object. Since making the user interface proved challenging, I used the above object to test certain methods. After completing the user interface, testing was successful and worked like planned. All GUI components are tested with different input values.

9. Known bugs and missing features

Editing text files in a program and changing their structure may cause errors in the program. Also after clicking “Done” and ingredientStorage has an ingredient that does not have a unit called “pcs” but the recipe has, it still deletes pieces from other unit ingredients from the storage. At this moment the program is working well and I don’t find any errors.

The program lacks a connection between ingredients and recipes, as mentioned at the beginning. Editing recipes is also not possible because the user must delete the recipe and make a new one. The third thing that catches the eye a bit, is the rounding that occurs after ingredient conversions. That looks bad in the interface when the ingredient amount is something like “3.9999999991”.

To solve the connection between recipes and ingredients, I thought about the situation that I would have added a Boolean value to the parameter of each recipe, which would have shown whether the ingredient is also a recipe. Thus, I might have been able to add ingredients from the recipe to the ingredient storage.

10. 3 best sides and 3 weaknesses

I think the top three parts of my project are: The user interface. Its coding was new and took considerably the most time. Secondly, how text files were written and read. The third thing in general in the program is the handling of lists and buffers where I succeed well.

Three weaknesses in my program are: First user interface. I don't think the interface will ever be complete. The components work but are not very stylish and for example some pop ups look bad. Another bad thing about the program, of course, is that I was not able to meet all the requirements. Especially the connection between

recipes and ingredients. The third thing I find is the interface code a bit tangled. I could have done the components using a different class structure, but now everything is in the same SmartCookBook Gui object.

11. Deviations from the plan, realized process and schedule

My plan remained more or less the same. I couldn't stay on schedule, but like I told in the Technical plan, making the interface would probably take the most time, which was true. I only got the user interface done about a week before the Project's final return date, so the two-week testing and repair period was a bit short.

12. Final evaluation

I think the project was more successful than I could have expected. For the first four weeks, it seemed like making an interface was a completely impossible task. However, its completion brought hope for the success of the project.

The lack of the above mentioned requirements were missing because the project was already at a good stage until I realized that the feature in fact was missing. At this point, however, there was no time left to change such a large amount of code and I wanted to get the project done on time. However, I thought that if the schedule gave up, I could add those requirements to my work later, but that was not the case.

The program could be improved a lot in the future. The user interface could be made much more user-friendly, and the handling of files would be better with the help of JSON or SQL, for example. I thought the class structure was good. Making changes and fixing program errors could be difficult for the interface, as the interface contains about 800 lines of code, including blank lines and it is a little bit confusing.

If I had to start making a program from scratch, I would immediately start preparing the interface, because no other methods are needed to do it. I realized this later during the project.

13. References

Scala Library https://www.scala-lang.org/api/current/