Technical plan: Smart Cookbook

Teemu Romo, 829841
Computer Science 2020
13.2.2021

Designing the class structure:

I have done a basic UML class diagram to represent the program's class structure. I found from my Projectplan classes: *Recipe, Ingredient, IngredientStorage, RecipeStorage, UnitConverter,* and *SearchBar.* I've also modelled the object, *SmartCookBook* and text-files for recipes and ingredients in the UML.

Recipe class has multiple parameters. It uses RecipeData.txt to have all those parameters saved. Every single *Recipe* has its own data file which can be edited from the UI. Method **edit** allows recipes to be edited. *Recipe* has methods **readData** to convert text-file to *Recipe* (and add them straight in the Buffer) and **wrtiteToData** to convert *Recipe* to text-file. 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.** 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) which have been inputted in a unit that is different from the one used in the Recipe. I need to use density for making that possible. Users can only choose units from predefined units and I needed to make class *Units* for that. 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*. *RecipeStorage* has the **newRecipe** method that creates a new empty recipe which is shown in the UI. **Delete** method can delete recipes from the storage. **Select** method will show the recipe in the UI.

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

Data structures & Algorithms:

My program will not basically use any specific algorithms. The *UnitConverter* uses the formula below (1) to calculate unit conversions.

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

Formula (1)

I think that is the only way to compare mass and volume and that's why I ended up to this conclusion. All other classes, which use methods for reading or writing the file, are using simple loop 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.

Schedule:

I try to make progress in two week cycles.

22.2 - 7.2 First I will create a base for the program. All classes and "TODOs" for methods. Then I will start to model GUI and maybe start creating it too.

8.3 - 21.3 Basic GUI example ready. Then *Ingredient, IngredientStorage, Recipe* and *RecipeStorage* methods. These classes are maybe most important and how to read and write text files and use them.

22.3 - 4.4 Next two weeks I will use to create *SearchBar, Units,* and *UnitConverter*. Also the main class/object *SmartCookBook*. There I will have a pretty good picture, how things are working and maybe I can test almost all methods which are using other classes.

5.4 - 18.4 Maybe the hardest phase (for me) in my project is graphical user interface. I'm not sure how long it will take to get all methods and especially my ideas to interact with the UI. If everything is going well, I succeed in creating a well-working UI and I can start testing all kinds of bugs and errors. My testing plan is to use mostly the UI for that.

19.4 - 28.4 Rest of time I will use testing or fixing the program. Also if graphical look doesn't please me, I will maybe modify that too. I hope I will have time with the full working program.

Testing plan:

I will mostly use the GUI for testing. Testing function by function is probably the most efficient way to test my program. I can test my methods for reading and writing correct text-files easily by doing broken and unbroken test files. I will handle errors with specific error messages that are shown in the GUI. For *UnitConverter* I thought to use some kind of test-class to check all possibilities for correct conversions. If all goes like planned, I can test all my methods through the GUI. Also I can give my program's beta version to my friends for testing. If they find bugs or errors I can fix them after they have been found.

For example when adding or deleting ingredients from *IngredientStorage* the UI should show the right amount of ingredients. I can do a double check and check the text-file too to be sure. *SearchBar* can be tested by writing criteria on the GUI and checking if the *RecipeStorage* is correctly filtered. I think all the methods can be tested like that.

References and links:

Aalto A+ https://plus.cs.aalto.fi/

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

I haven't used any other material for my initial plans but I am sure I will use more references deeper into the process.