Skylee Blaine

What does the program do?:

This program is a ‘recipe finder/management app’ which will help users discover recipes given its ingredients. The user will input a list of ingredients they currently have, and the app will recommend recipes that can be made using those specific ingredients.

Algorithms:

1. findRecipesByIngredients method
    a. Input HashSet of user ingredients
    b. Create an empty list of matchingRecipes
    c. Loop through all recipes in the recipeManger
    d. Create a HasSet to find matching ingredients between user and file
        i. If this is not empty, there is a matching recipe
    e. Add to matchingRecipes list and return
2. findRecipesByCategory method
    a. Input category (string)
    b. Create an empty list of matchingRecipes
    c. Loop through all recipes in the recipeManger
        i. Use .equalsIgnoreCase to make it case-insensitive
    d. Add to matchingRecipes list and return
3. Save and load from a file
    a. Saving:
        i. Input recipe.txt and the recipes stored in the recipeManager
        ii. Format recipes using: name|category|ingredient1,ingredient2…

iii. Return as a string as a new line in the txt file

b. Loading:

i. Parse the lines/split the line into name, category and ingredients

ii. Handle invalid lines, will skip the line completely

These algorithms are essential to the core functionality of the program: conducts ingredient and category based searches based on user input. Categorizing this makes the app more organized and efficient in finding matching recipes for the user. File loading and saving is also very important in keeping the data stored, while also allowing the user to store more recipes later on (I haven't quite yet implemented this entirely into my program). HashMaps and Hash Sets allow for quick access to the recipes by name, then by ingredients so comparison can happen more quickly. I used Replits AI function to learn how to set up the HashMaps and HashSets for this specific program, while also allowing them to read the txt files correctly (parsing).

Big O of the algorithms:

I think the Big O for all of these algorithms are O(n) because they are pretty linear. For example, the loop required for searching is only as big as the amount of recipes given (n), so time is proportional to the number of recipes being looped through.

Data Structures:

As I mentioned before, I used HashMaps and HashSets because they can be a fast and efficient way to look through the recipes. The HashMaps would store each recipe by name, and the

corresponding HashSet would make it easier to compare each ingredient in the file to the user inputted ingredient.

Opportunities:

While I didn't originally plan on adding a 'save to file' option, while learning how to read from a file in java, I figured it would be best to implement it anyways. Now for future updates of my app, I can add a method/class specifically allowing the users to add their own recipes, rather than hardcoding them into the txt file like I did for this version of the project.

Errors:

Besides my many errors with setting up the HashMaps and HashSets, I did struggle with the recipe.txt file. I originally had the file set up with something like this:

—

recipe1

Category

Ingredient1, ingredient2, ingredient3…

—

recipe2

Category

Ingredient1, ingredient2, ingredient3…

—-

While I liked how it looked, the format made it more difficult to parse and create recipe objects for the HashSets. So, i decided to simply the format by putting everything onto one line instead: Name|category|ingredient1,ingredient2…

Future updates:

1. User interface: Because the program works only on the terminal, it doesn't feel like a proper app. I'd like to create a user interface to make the experience more personal and exciting for the user.
2. Database: If a user is able to save their own recipes, I think using a database rather than a file would be a better option for handling more recipes. It could also make searching for matching recipes a lot faster and more efficient.