Recipe Run

Alexa Javellana

CMPT220

Recipe Run: An Appetizingly College Friendly Application

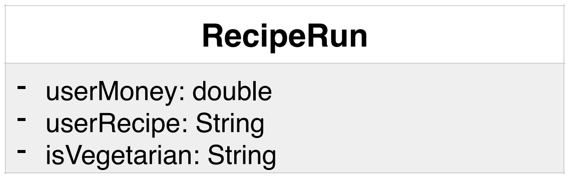
My project revolves around the classic problem most college students new to culinary innovation face: the aspect of creating new recipes. Low budget and virtually lost without the aid of others, like their parents, undergraduates out of the dorm lifestyle must face cooking for themselves for whole semesters, and more than often look towards the internet for help. My application, suitably called Recipe Run, aims to minimize the users panic regarding what to eat and readily presents curated recipes that require only a few steps towards creating a new dish. Basically a random content generator, the app once run asks for the user to input a numeric value in order to provide feedback, in this case being a recipe that would equate in value to what the user typed in. The only things required to run my application would be for the user to have a version of JVM installed on their computer, and an interest/need to new recipes. Technically, there are two main parts to this application's code that I will explain in this paper, being the RecipeRun class and the Recipe class, which contain all contents in order for it to run correctly.

In order to show how my application works and explain the code behind it's function, the normal flow of events must be described. Upon running the program, the application will present a sentence that asks the user to input a value to act as their budget. After accepting the number typed in, the program will then restate the budget inserted, and ask if the user would like vegetarian recipes to be shown (instead of the default non-vegetarian recipes). Dependent on whether or not they say yes or no, a vegetarian or non-vegetarian recipe will appear and the system will exit. The user may run the program again and get different recipes if they enter different budgets, and/or change whether or not they want vegetarian recipes.

The code regarding this flow of events exists in the RecipeRun java file, which uses methods, variables and other elements that are part of the Recipe class. The only way that this program can work is if a valid value for budget is put in by the user, which is restricted to a double range from 0 to 20.01 (which makes the user able to type any value from a penny to twenty dollars as a budget for their meal). If the value is over 20 dollars or below 0 (being negative), RecipeRun runs the if statement that rejects the value, prints “Invalid budget! Try again.” within the system and then exits the application. If the user uses a suitable number, this number becomes the variable userMoney, and is used to create a new Recipe object called userRecipe. This userMoney then becomes the budget variable in the Recipe class being run in RecipeRun, classified as mealBudget. This whole process is what allows the user to receive their randomly generated recipe by the end of the program.

But, as stated in the prior paragraph, before the recipe can appear in the program, the budget will be restated in a sentence by calling the userRecipe.mealBudget function, which will only work if the user put in a correct value. The sentence will continue and then ask whether or not the user would like a vegetarian or non-vegetarian recipe. A new string variable is created in the code then, called isVegetarian, whose contents depends on the next string input the user types in, which should either be yes or no in response to the vegetarian question. Next, an if - else if statement that judges whether or not the contents of isVegetarian equals to yes or no by using the line isVegetarian.equalsIgnoreCase(“yes”) and isVegetarian.equalsIgnoreCase(“no”). If the answer is yes, then the program will state that the user has chosen vegetarian options to presented and show an appropriate recipe. If the answer is no, then the program will do the opposite and simply state “Here is your budget-appropriate recipe”.

UML Diagram



- The user input budget
- The user's recipe from the budget
- Whether or not the user would like vegetarian recipes to be presented

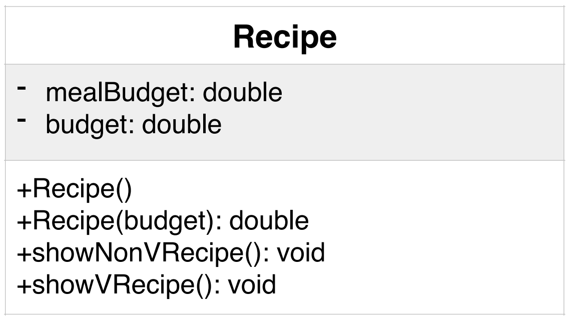
In order to understand the process that the program goes through to deduce which recipe is shown to the user within RecipeRun, the contents of the Recipe class must be explained. As stated before, there exists a variable within the Recipe class called mealBudget. In the code, mealBudget's value is defaulted at 1, aka 1 dollar. Additionally another constructor in the Recipe class creates a new double called budget (by Recipe(double budget)) that mealBudget becomes. This is so that in any type of program, any kind of double variable defined in a java program can act as budget in order to generate a recipe according to the variables quantity.

Now that the program has a budget to work with and the knowledge of if the user wants a regular vs vegetarian recipe, the Recipe class will evaluate both data fields to deliver a recipe. The two methods defined in the Recipe class are called showNonVRecipe, which runs if the else if statement is true where isVegetarian equals no, and showVRecipe, which runs if the if statement is true where isVegetarian equals yes. In both methods, there are if else if statements that mathematically consider the values of mealBudget separately, into ranges. Specifically, the statements evaluate if mealBudget's quantity falls between 1-5, 6-10, 11-15, or 16-20.

In the showVRecipe method, there are four parts to the if else if statement which evaluates mealBudget, which equates to a total of four recipes that exist within the showVRecipe method. If mealBudget's value inserted when running RecipeRun lies between 1-5, the recipe for "Avocado Toast" will be present. For mealBudget's between 6-10, the recipe for "Vegetable Stir

Fry” comes out, 11-15 “Sweet Potato Chickpea Buddha Bowls”, and finally for 16-20, the recipe for “Thai Red Vegetable Curry” shows on screen. The recipe name, ingredients (numbered and in quantitative measurements like tablespoons, cups, numbers, and descriptions like sliced, diced, pinches, etc.), and instructions will appear and the system will exit.

Likewise, the showNonVRecipe method runs in a similar fashion, but instead has two recipes for each range in mealBudget, equating to in total 8 recipes within it, meaning that there are 8 parts to it’s if else if statement(s). There is no specific reason to why it has more recipes than the vegetarian method, other than just to provide more variety to what types of recipes appear since most people using the application would most likely have no strict dietary restrictions. If the mealBudget falls within 1-5, and is a even value, then the recipe for “Mac N’ Cheese” will appear. If mealBudget falls within 1-5 but is an *odd* value, then the recipe for “Asian Style Fried Rice” will appear instead. This even and odd judgement occurs for the rest of the ranges under this method, too, in order to provide the content/variety as explained before. Similarly, the even recipe if mealBudget falls from 6-10, “Mixed Green and Chicken Salad” will appear. The odd recipe for the 6-10 range is “Spiced Peanut Noodles”. Continuing this trend, the even recipes for mealBudget ranges 11-15 and 16-20 are “Spiced Pork Belly Tacos” and “Mongolian Beef with Broccoli” respectively, and the odd recipes for those ranges are “Grilled Pineapple Teriyaki Burgers” and “Black Cherry Glazed Pork Tenderloin”.



Recipe

- mealBudget: double
- budget: double

+Recipe()
+Recipe(budget): double
+showNonVRecipe(): void
+showVRecipe(): void

UML Diagram

- The default budget of the meal
- The user’s input budget of the meal

- Recipe constructor
- Equates the new user input to the mealBudget
- Shows a non-vegetarian option out of two recipes in the program
- Shows the singular vegetarian option for the user input budget

In regards to errors and error prevention within the application, there is not much room for errors since the program itself is simple in nature. The only things that RecipeRun asks from the user for is for a value to act as the budget and a yes or no in correspondence to the presentation of vegetarian meals. Defined errors occur if, when putting in a budget, the value is larger than 20.01 or a negative number, and are dealt within the else if else statement that states it's invalidity and exits the system which exists within the RecipeRun program. Since the program is run through java, also, if any foreign characters (i.e. ^%&*$@) or strings are put into the field instead of a double value, the system stops running the program because of an InputMismatchException. When it comes to the isVegetarian string input, the code is set up to ignore any case differences, which means there would be no errors if the user puts in "yes", "yES", "YES", "yeS", "yEs", "YEs", "Yes", "no", "No", "nO" or "NO". But, since the program can only run if the input is a variation of yes or no, if anything else is inserted, such as a number, another word or character, the system won't declare an error or exception, but simply exit without showing a recipe.

Collectively, this project turned out to produce a viable application that provides a straightforward service to the consumer in question. Easy to maneuver, the user only needs to put in a minimal amount of effort in order to get what they desire. Additionally, the code is simple to read for any programmer, and recipes could be added and improved upon if wanted. RecipeRun approaches an otherwise complicated real-life situation in a casual way effectively and efficiently, without too much features that would make it overwhelming.