GROUP NAME \_\_\_\_\_\_\_\_\_\_Group 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Diet Manager V1 – Rubric

SWEN.383 SW Design Principles and Patterns

Final Java Solution for Diet Manager V1

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **Tasks** | **Points** | **Score** |
| ☐ 1 | Load a foods.csv file with only basic foods and an empty log.csv file. User can see an empty log for today with default calories and weight. User can view the basic foods loaded.  This feature is executed at the beginning of the program, as well as on change. It starts by calling model.getAllFoods() in the DietController class. That method is populating the collection of foods in the DietModel class by calling Food.loadFoods() method. loadFoods() method of the Food class is loading the foods.csv file and reading it line by line, and using the composite pattern to store both BasicFoods and Recipes into a single collection of foods – ArrayList<Food>. The loadFoods() method of the Food class returns an Arraylist of Foods to the DietModel class. To show loaded foods to the user, in the DietController if the ui is instance of HomeUI, it calls the setFoods method which injects an ArrayLIst<Food> as a parameter, and the second parameter is a ComboBox in which the foods will be shown. That method is then setting each food from the list into the ComboBox so that users can see and select that Food to be logged. Regarding the log, in the HomeUI we have an empty TextArea that will be populated when user selects a date and clicks to create a log. Also, the user can click a button Check your daily intake which will lead him to InfoUI where he can see all log history, sorted by date. Regarding weight, the user can click Check your weight button which will lead him to WeightUI where he can see his weight log history (and a message if no weight has been logged). | (60) |  |
| ☐ 2 | Level 1 PLUS Select a basic food(s) for the daily intake. The selection is stored in the database (log.csv) the log view is updated with the dietary information about the nutrients consumed.  This feature starts when the user chooses food from the comboBox and clicks the Log button. (The user can choose a date, but if no date is chosen it will automatically log it for today's date). When the log button is clicked it triggers a setOnAction in the DietController class (((HomeUI) ui).getLog().setOnAction(), which checks if the user chose a date or not. Then it calls the logSelectedFood method of the DietModel class sending the String of the food and the Date as parameters. That method creates a new Log instance and calls the logFood() method of the Log class. That method is responsible for writing that Log instance to the log.csv file using the toCsv() method for formatting. Log view is updated when the InfoUI is instantiated or when the user clicks to create a new daily log. | +10 (70) |  |
| ☐ 3 | Level 2 PLUS Add new basic food(s) to the food database. This implies the ability to then add such basic food(s) to today's log as in level 2  This feature starts when the user fills in information about that basic food (name, calories, proteins, carbs, fats) and clicks the Add Food button. When that button is clicked the setOnAction is triggered in the DietController class (((HomeUI) ui).getAddBasicFoodButton().setOnAction(e -> {) …). That setOnAction is calling addFood method in the DietModel clas, parsing a String as a parameter. That method is creating a new BasicFood instance and calls the addBasicFood methods of the BasicFood class. That method is responsible for writing that BasicFood object into the foods.csv file using the toCsv method for formatting. Both comboboxes are then updated to show that new food by calling the setFoods method for both comboBoxes in DietController in setOnAction method of Add Food button. | +5 (75) |  |
| ☐ 4 | Level 3 PLUS Loading and viewing a foods.csv file with recipes.  Because of the composite pattern, this feature works exactly like Level 1. The loadFoods method returns the collection of Foods, that includes both BasicFoods and Recipes. Recipes are also then shown in the ComboBoxes for the users to see and log them. | +5 (80) |  |
| ☐ 5 | Level 4 PLUS Select recipe(s) as well as basic food(s) for the daily intake. The selection is stored in the database (log.csv) and the log view is updated with the dietary information about the nutrients consumed.  This feature works exactly like the Level 2 feature. When choosing a Food to log, thanks to the composite patterns both recipes and basicFoods are shown the same. Because of that if the user chooses a basic food or recipe, the process of logging will be the same. It will trigger the setOnAction that will call the logSelectedFood, create a new Log instance, and write it into the log.csv file using the logFood method. | +5 (85) |  |
| ☐ 6 | Level 5 PLUS add new recipe(s) to the food database. This implies the ability to  then add such recipe(s) to today's log as in level 5.  This feature starts when the user fills in information about the recipe (name, what Foods it consists of, and amounts), and when the user clicks the Add Recipe button it triggers the setOnAction in DietController (((HomeUI) ui).getAddRecipeButton().setOnAction(e -> { … ). In that setOnAction the addRecipe method from the DietModel is called with String as a parameter. That method creates a new Recipe instance and calls the addRecipe method of the Recipe object, which is responsible for writing that Recipe object into the foods.csv file using the toCsv method for formatting. Both comboboxes are then updated to show that new food by calling the setFoods method for both comboBoxes in DietController in setOnAction method of the Add Recipe button. | +5 (90) |  |
| ☐ 7 | Level 6 PLUS the ability to read a non-empty log.csv file, to navigate to  different days in the log, and to select foods for the intake for the days other than today.  Users can choose a date and click the Create Daily Log button, which will show them the logs for that date and calculate the total calories logged that day. When the user click the Create Daily Log button, the setOnAction is triggered in the DietController class (((HomeUI) ui).getDailyButton().setOnAction(e -> { … ). That setOnAction first updates the log collection with the latest logs using the updateLog method from the DietModel class which calls the Log.getData() method that reads the log.csv file. Then it calls the addDailyLog method of the DietModel class, parsing the String of the date as a parameter, and that method is responsible for calling the dailyLog method of the Log class, also passing that String as a parameter. That dailyLog method gets the collection of all logs, iterates through it, and for every log that is logged for the date that the user chose, it stores it into a new collection called dailyLog. It also calculates the calories of all Logs for that date. Then to display that collection to the user, the addDailyLog method of the HomeUI is called in that setOnAction in DietController, passing the collection of Logs (generics used), textArea where that collection will be shown and totalCalories calculated. That method then iterates through that collection and displays all logs for that date, with the total calories at the end. For selecting foods for intake for the days other than today, a new Date picker is created next to the ComboBox of the foods to log, if the user chooses a date it will log the food for that date, and if not it will log it for today’s date. (Explained how it works in Level 2). | +5 (95) |  |
| ☐ 8 | Level 7 PLUS the ability to save the log and food database back to the log.csv and foods.csv files.  To save a new food (BasicFood or Recipe) into a foods.csv file is explained in Level 6. Saving new logs to log.csv file is explained in Level 5. | +5 (100) |  |
| SUBTOTAL: | | **100** |  |

To receive any credit for level N, the preceding levels must be sufficiently functional to test level N. In general, this means previous levels must work without failure when the user enters normal (non-error) data.

Remember, the solution should apply design patterns as required. The final grade will be adjusted based on how good you have applied the pattern. You may lose up to 25% of the final project grade in this regard.