Marvin Jakobs

Professor Tolic

SWEN 383

25 April 2023

Wellness Manager Design Document

Composite Pattern

This program incorporates the Composite design pattern in the Recipe class. The Recipe class implements the FoodComponent interface and has a Map of FoodComponent objects and their quantities. The addFood() method in the Recipe class checks to see if the parameter passed is an instance of the BasicFood class or the Recipe class. If it's an instance of the BasicFood class, it adds the food to the Map and if it's an instance of the Recipe class, it recursively calls the addFood() method on each of the ingredients in the Recipe object's Map. This allows the Recipe class to be treated as a composite object, containing other FoodComponent objects.

Model View Controller Pattern

This program also incorporates the Model View Controller (MVC) design pattern. The `LogForm` class is the view, the `LogController1` class is the controller, and the `BasicFood`, `DailyLog`, `FoodCollection`, and `Recipe` classes are the models which handle all of the calculations for foods, exercises, and daily logs. The controller communicates between these model classes and the LogForm view to update and pull data to be displayed in the AWT UI. The MVC design pattern helps separate potential problems and make the code more modular and maintainable, especially making it easy to build on top of Wellness Manager Version 1.

Memento Pattern

This program also incorporates the memento design pattern. The memento pattern lets an object capture its internal state and save it externally so it can be restored to that state later. In this program, the loading of CSV data in DailyLog, FoodCollection, and Exercises is an example of the memento pattern being implemented. Each of these classes has an internal state that is captured in a memento when loading data from a CSV file. The CSV reader updates their states based on the data in the file, and if necessary, can be restored to its previous state by retrieving the memento object. This separation of state from behavior allows flexibility and modularity in the code, making it easier to maintain.

Mediator Pattern

This program also uses the mediator design pattern. The LogController acts as a mediator between the LogForm UI view and the models FoodCollection, DailyLog, and Exercises. The LogForm UI view allows the user to input all kinds of data about foods, exercises and daily activities which is processed by the LogController. The LogController then updates the FoodCollection, Exercises, and DailyLog models with the relevant information. Acting as a mediator between the view and the models, LogController makes sure that the view and the models remain decoupled and independent. This is a separation of concerns and makes it easier to update the application without other elements being changed. The LogController as a mediator also makes it easier to test and debug the program.

Singleton Pattern

This program also uses the Singleton pattern to ensure that there is only one instance of the DailyLog and FoodCollection classes throughout the program. This is done by making the

constructors of these classes private and making a public static method to return the single instance of the class. In the LogModel class, the loadLogData method calls the loadLog method of the DailyLog class, which is a Singleton. Similarly, the loadFoodData and saveFoodData methods call the loadFoods and saveFoods methods of the FoodCollection class, which is also a Singleton. Using the Singleton pattern, the program can make sure that there is only one instance of these classes, which can be accessed by all other parts of the application. This helps to maintain consistency and avoid conflicts that may come up from having multiple instances of the same class.

Conclusion

In conclusion, the Wellness Manager program is a well-designed application that uses several design patterns to increase modularity, maintainability, and flexibility. The Composite pattern is used in the Recipe class to allow it to contain FoodComponent objects like basic foods. The Model View Controller pattern is used to separate potential concerns and make the code more modular and maintainable. The Memento pattern is used to capture and restore the internal state of objects from csv files, making it easy to maintain and update the application. The Mediator pattern decouples the view and the models, making it easier to test and debug the program. Finally, the Singleton pattern is used to make sure that there is only one instance of the DailyLog and FoodCollection classes throughout the program, maintaining consistency and avoiding conflicts. Overall, the use of these design patterns makes the Wellness Manager program an efficient application that can be easily maintained and updated in the future.

UML of Program Structure: