Design Document
Diet Manager
SWEN.383
Prof. Domagoj Tolic

Orest Kisi, Orest Brukhal

Table of Contents

1.Introduction

The Food and Exercise Tracking Application is a desktop program designed to help users monitor their dietary intake, track exercise activities, and manage their weight goals. It provides features to log food consumption, record exercise sessions, set calorie targets, and visualize nutritional data. This design document outlines the architecture, components, and functionalities of the application.

2. Objectives

1. Food and Exercise Logging: Enable users to easily log their daily food intake and exercise activities, including details such as servings consumed, exercise duration, and intensity.
2. Database Management: Provide a database of basic foods, recipes, and exercises, allowing users to create custom meals and workouts tailored to their preferences and dietary requirements.
3. Customization and Flexibility: Allow users to customize exercise entries and calorie expenditure calculations based on factors such as weight, intensity, and duration.
4. User-Friendly Interface: Design an intuitive and visually appealing user interface that facilitates seamless navigation, data entry, and information retrieval.
5. Data Persistence and Integrity: Implement robust mechanisms for storing user data, ensuring data integrity, and enabling seamless synchronization across devices.

3. Architecture Overview

1. MVC (Model-View-Controller) Pattern:

Model: Classes such as LogManager, RecipeFinder, and RecipeManager encapsulate the application's business logic and data management functionality.

View: The SwingGUI class and its associated factory classes (ComponentFactory, TextFieldComponent, TextAreaComponent, etc.) create and manage the graphical user interface components.

Controller: Event listeners (ActionListener and ItemListener) in the SwingGUI class respond to user interactions and delegate control flow to methods in the model layer.

2. Composite Pattern:

Component (Interface/Abstract Class): This base class defines a common interface for all individual components and composites.

Leaf (Concrete Classes like ButtonComponent, LabelComponent, etc.): These classes represent the leaf nodes in the composition hierarchy, meaning they do not have any children. They

implement the operations defined in the Component class which typically involve rendering the GUI element and handling user interactions. Each component like ButtonComponent wraps a specific Swing component (e.g., JButton) and standardizes its initialization and behavior.

ComponentFactory (Creator): Central to the Composite Pattern in our implementation, ComponentFactory abstracts the instantiation details of both leaf and composite components. It provides a unified interface to create any GUI component required by the application, ensuring that all components are instantiated in a consistent manner. This factory handles the specifics of creating each component type, whether it's a simple button or a complex panel with nested components.

3. Factory Method Pattern:

ComponentFactory: Responsible for creating various Swing components (JLabel, JTextField, JButton, etc.) based on specific requirements, encapsulating the instantiation logic, and promoting loose coupling.

4. Observer Pattern:

The Swing framework itself uses the Observer pattern extensively. Components like buttons and combo boxes register event listeners (action listeners, item listeners, etc.), which are notified when events occur. In the provided code, the addActionListener and addItemListener methods register action and item listeners, respectively, to handle user interactions with GUI components.

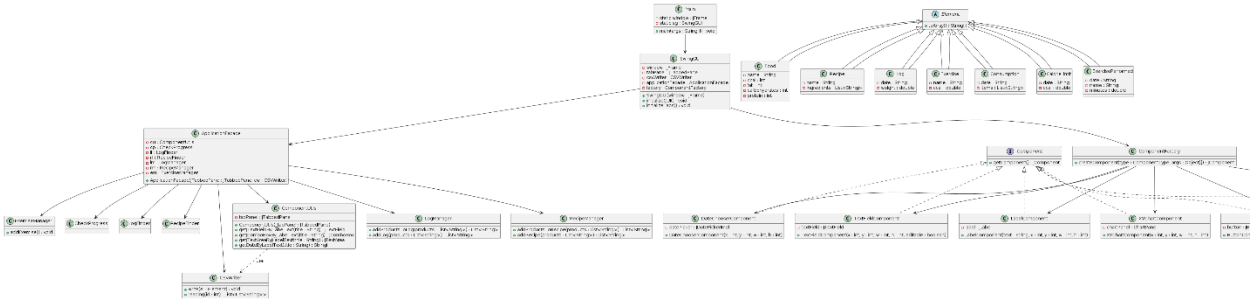
5. Singleton Pattern :

The CSVWriter class implements the Singleton pattern. This pattern ensures that only one instance of the CSVWriter class is created throughout the application's lifecycle. It achieves this by providing a static method, getInstance(), which returns the same instance of the CSVWriter class every time it's called. This pattern is beneficial when there's a need for a single, shared resource or utility class, such as a file writer in this case.

6.Façade Pattern:

This pattern is represented by ApplicationFacade class which abstracts the complexity of interactions with multiple manager classes (ExerciseManager, CheckProgress, LogFinder, RecipeManager). It acts as a centralized point for the GUI to interact with the application logic, conforming to the Facade Pattern.

4.Class Diagram



Main: The entry point of the application that instantiates SwingGUI. It is responsible for starting the GUI and orchestrating the initial application setup.

SwingGUI: Represents the main application window, which contains a JTabbedPane for tabbed navigation. It integrates an ApplicationFacade to streamline operations between the GUI and the underlying business logic.

ApplicationFacade: This class abstracts the complexity of interactions with multiple manager classes (ExerciseManager, CheckProgress, LogFinder, RecipeManager). It acts as a centralized point for the GUI to interact with the application logic, conforming to the Facade Pattern.

ExerciseManager, CheckProgress, LogFinder, RecipeManager: These classes encapsulate specific business logic operations of the application, such as managing exercises, tracking progress, finding logs, and managing recipes, respectively.

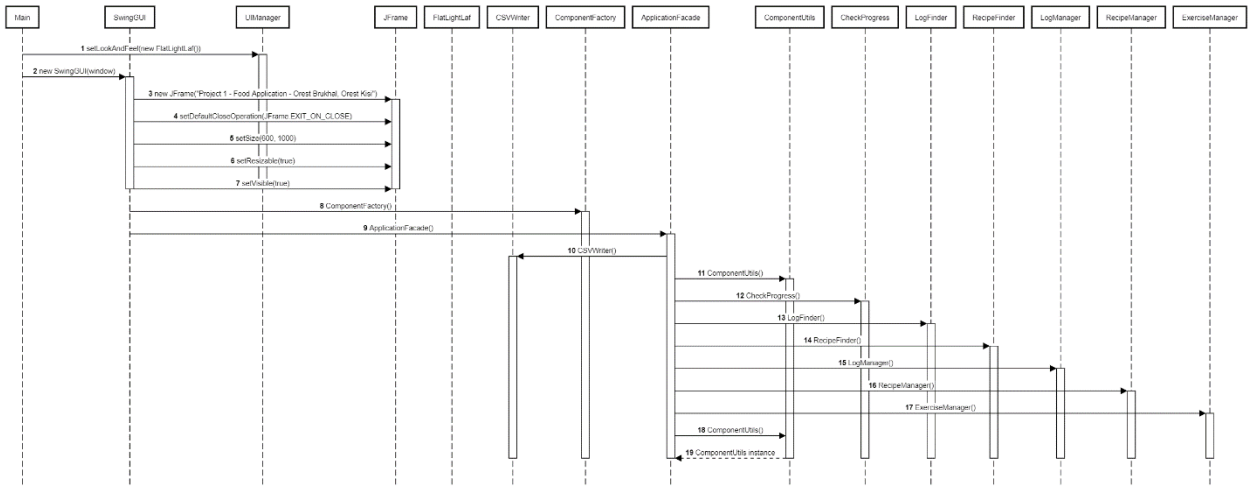
ComponentFactory: Utilizes the Factory Pattern to create GUI components. It has a method createComponent, which returns instances of Component. It is the crux for creating interface elements, suggesting a dynamic approach to UI construction.

Component: Is an interface representing the base for all UI components. Classes like LabelComponent, ButtonComponent, TextFieldComponent, and XYChartComponent implement this base, applying the Composite Pattern when these components are used into a larger structure in the SwingGUI and using the ComponentFactory to pass these components.

Data Classes: Food, Recipe, Log, Exercise, Consumption, ExercisePerformed represent domain objects that hold the data and are manipulated by the manager classes.

CSVWriter: A utility class that is used for exporting data to CSV format.

5.Sequence Diagram



Main initializes SwingGUI:

The Main class starts the GUI process by setting the look and feel to FlatLightLaf and creating a new instance of SwingGUI, passing window as an argument.

SwingGUI JFrame Operations:

SwingGUI then creates a new JFrame.

It sets the default close operation to EXIT_ON_CLOSE.

It sets the size of the frame to 600x1000 pixels.

It makes the frame resizable and finally visible, indicating the GUI is ready for the user to interact with.

Component Factory and Application Facade Initialization:

SwingGUI initializes ComponentFactory and ApplicationFacade which are responsible for creating GUI components and managing application logic, respectively.

CSVWriter Initialization:

ApplicationFacade initializes CSVWriter, which handles writing data to CSV files.

ComponentUtils and Manager Classes Initialization:

ApplicationFacade further initializes ComponentUtils, which could be a utility class for common component operations.

CheckProgress, LogFinder, RecipeFinder, LogManager, RecipeManager, and ExerciseManager are also initialized, indicating these classes are responsible for various aspects of the application logic related to their names.

Interaction with Manager Classes:

The sequence shows ComponentUtils being used before the initialization of various manager classes. ComponentUtils provides shared functionality needed by the managers.

Manager Class Interactions:

LogFinder, RecipeFinder, LogManager, RecipeManager, and ExerciseManager are used in sequence, creating a flow of operations that involve finding logs, looking up recipes, managing logs, managing recipes, and managing exercises. They reflect the business logic of the application.

Conclusion:

This design document is a comprehensive guide to this project's development and how each component of the program and each class work together to create a user-friendly program that anyone can use. The use of design patterns in this project was crucial to the success of the program and although quite tough to implement as newbies, we think we did an okay job. The program is easily updatable and modular for later additions. An interesting experience to say the least that we will definitely make use of it later in life.