Diet Manager Version 1.0

Project Design Document

Team Cake

Tenzin Dhondup <txd5857@rit.edu>
Chad Cummings <cbc6525@rit.edu>
Zhimin Lin <zxl1987@rit.edu>
Chandler Sofia Michel <csm4025@rit.edu>
Daniel Cox <drc8599@rit.edu>

Project Summary

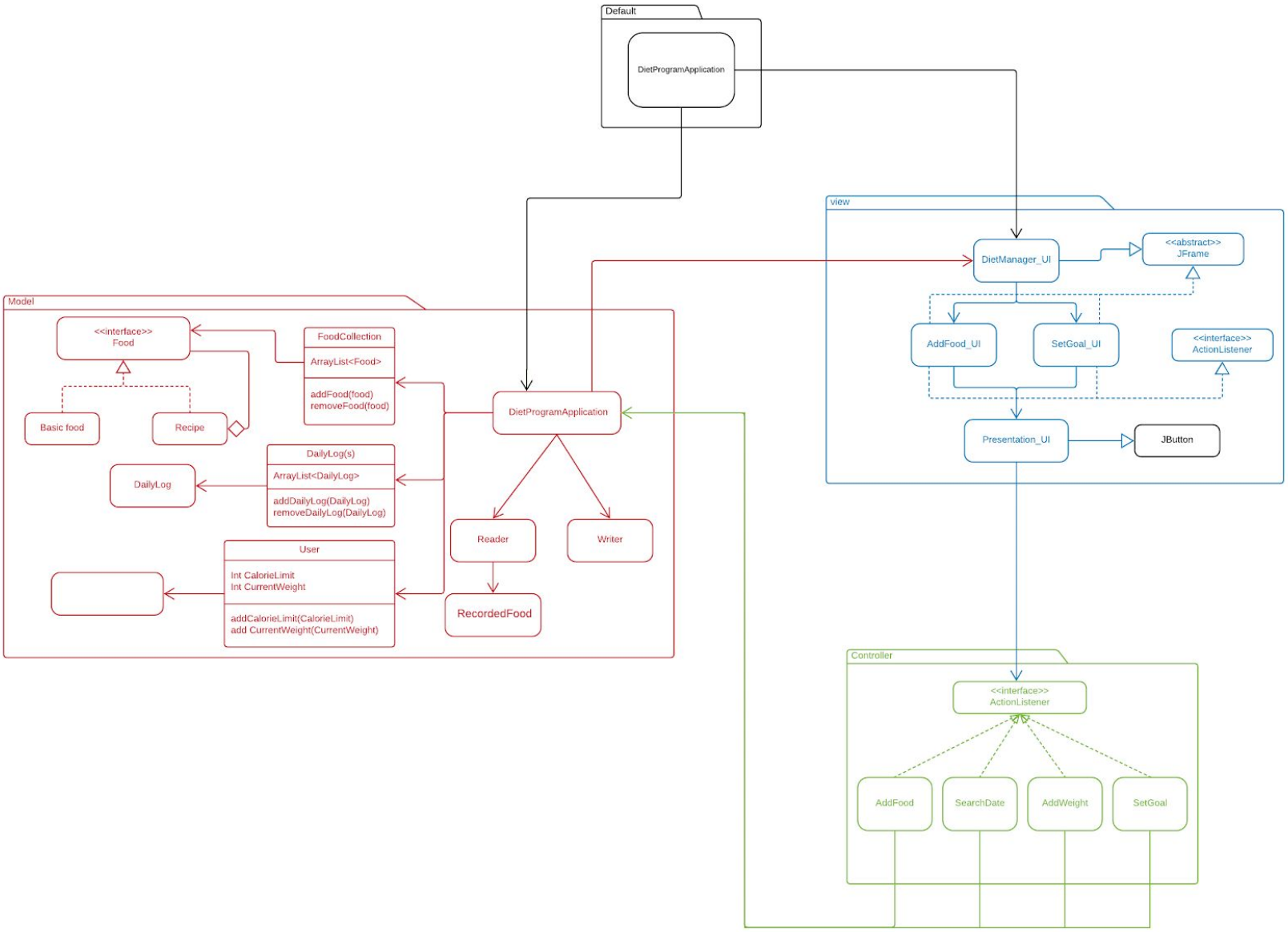
Diet Manager is a program that helps users monitor their diet. Each time a user eats, they can enter in a food item or multiple food items that make up a recipe. For each food item they enter, they must provide the name and nutritional information (i.e. calories, fat, etc.). For each recipe the user enters, they must provide the name and the basic food items that make up the recipe. After the user enters in a food item or recipe the first time, it is added to the food collection and they may select it at another time, without having to add it again.

Diet Manager shows users an overall look at the user's daily dietary information in terms of calories, fat, carbs, and protein. Diet Manager also tracks the user's weight by allowing the user to periodically input their weight. Additionally, users can set a daily calorie limit and Diet Manager will inform the user if they are under or over their daily limit.

Design Overview

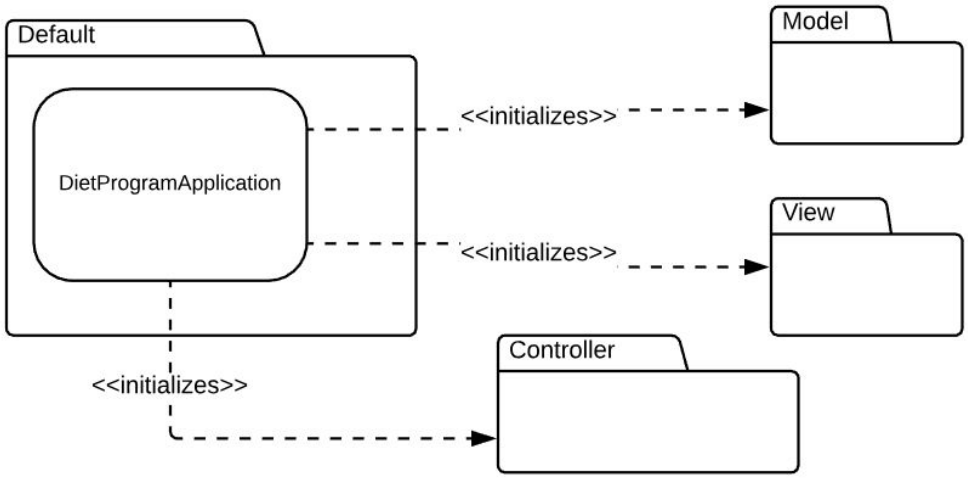
The Diet Manager design incorporates the Composite design pattern and the Model-View-Controller (MVC) architectural pattern. A preliminary design did not incorporate the MVC architectural pattern, but it was later added in order to produce a more extensible application. The Composite pattern is used in order to create a cohesive relationship between the Recipe and BasicFood objects. This section will be updated as we continue to move through the design process.

Subsystem Structure



Default Subsystem

Class DietManagerProgram	
Responsibilities	Create the model object(s). Create the graphic user interface. Display the GUI for user to used.
Collaborators (uses)	**model.DietProgramApplication -** the primary model class



Model Subsystem

Still needs to implement

Class DietProgramApplication	
Responsibilities	Notify observers (UI) of changes to the
Collaborators (uses)	

Class Writer	
Responsibilities	Allow csv entries to be written to all CSV files (foods.csv, log.csv)
Collaborators (uses)	**java.io** - to use the PrintWriter and File classes **java.util** - to use Calendar class

Class Reader	
Responsibilities	Allow csv entries to be read from all the CSV files (basicfood.csv, foods.csv, log.csv)
Collaborators (uses)	**java.io** - to use the PrintWriter and File classes **java.util** - to use Calendar class

Still needs to implement.

Class User	
Responsibilities	Stores the user's current weight and desired caloric intake. Update the user's current weight and desired caloric intake.
Collaborators (uses)	**java.io** - to use the PrintWriter and File classes **java.util** - to use Calendar class

Class DailyLog	
Responsibilities	Creates a Daily Log object / entry for the Writer Class. Takes in values and stores user's recorded weight. Takes in values and stores user's desired caloric limit.
Collaborators (uses)	**Model.RecordWeight** **Model.RecordFood**

Class Weight	
Responsibilities	Takes in values and stores user's recorded weight and date.

	Stores it in an object, along with the selected date.
Collaborators (uses)	**java.util.Date**

Class ConsumedFood	
Responsibilities	Takes in values and stores user's "consumed" food and date. Stores it in an object, along with the selected date.
Collaborators (uses)	**java.util.Date**

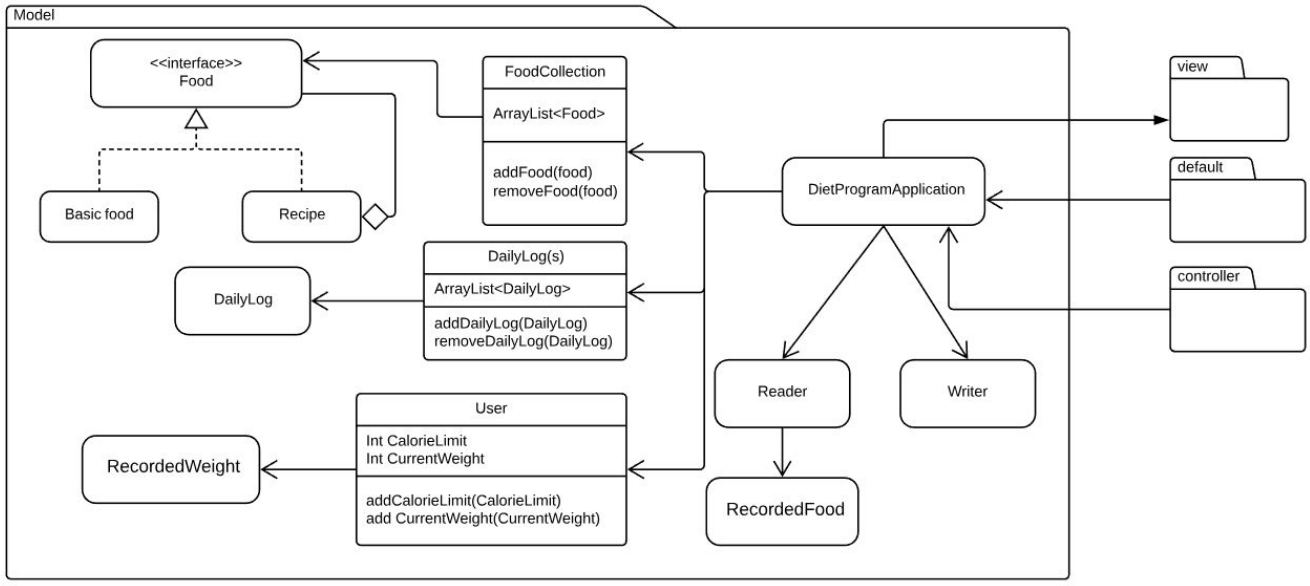
Class CaloricIntakeLimit	
Responsibilities	Takes in values and stores user's "caloricintake" and date. Stores it in an object, along with the selected date.
Collaborators (uses)	**java.util.Date**

Class Food (interface)	
Responsibilities	Provide a generic interface to all food items (Recipe and Basic Food). getNutrDiet() - Method that returns the nutritional information. getName() - Method that returns the name of the food or recipe *Composite Pattern*

Class BasicFood	
Responsibilities	Following Attributes are the Name, Calories (*number of calories in one serving of that food*), fat, carb, and protein (*number of grams in one serving of that food each*). getNutrDiet() - Method that returns an array of the values of all the attributes listed above. Ands add them to a NutrDiet Array. getName() - Method that returns the name of the food *Leaf in the Composite Pattern*
Collaborators (implements**)**	**Model.Food**

Class Recipe	
Responsibilities	A collection of basic food and sub-recipes. Following Attributes are the Name (*of the Recipe*), the name of either the Basic Food or Subrecipe(*that makes up Recipe*), number of servings (*of that Basic Food or Subrecipe that was listed*).

	In the Recipe composite, it can have multiple Basic Food/Subrecipe depending on the composition of the Recipe. getNutrDiet() - Method that returns an array of the values of all the attributes listed above. Ands add them to a NutrDiet Array. getName() - Method that returns the name of the recipe
Collaborators (implements**)**	**Model.Food**



View Subsystem

Class Current Command Line Interface.	
Responsibilities	Is the User Interface of the DietManager in terms of interacting with the features such as adding food, updating weight/desired caloric limit.
Collaborators	**Controller.**

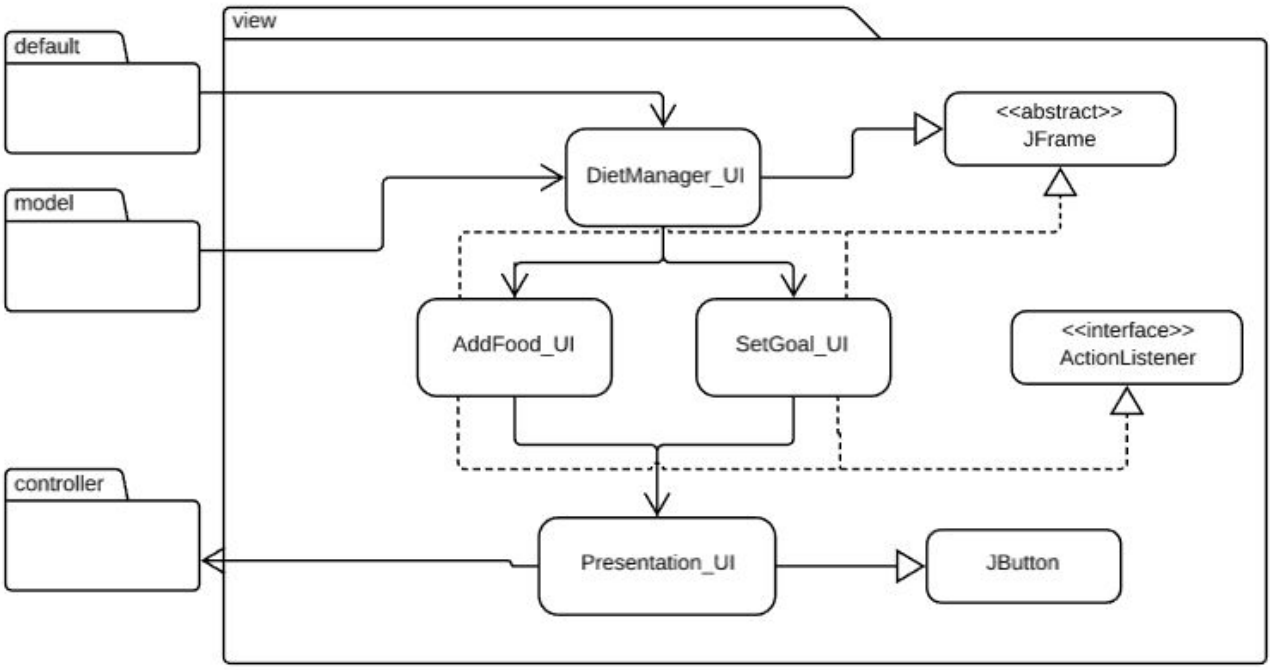
(uses)	

Class DietManager_UI	
Responsibilities	Is the User Interface of the DietManager in terms of interacting with the features such as adding food, updating weight/desired caloric limit.
Collaborators (uses)	**All the classes listed below.** **As well as all classes will use javax for the visual elements.** **implements - ava.awt.event.ActionListener**

Class AddFood_UI	
Responsibilities	Contains all the required UI components that makes up the screen of which users interact with to add food (basic food / recipe) to the food collection or adding the consumed food to the daily logs
Collaborators (uses)	**Controller.AddFood**

Class SetGoal_UI	
Responsibilities	Contains all the required UI components that makes up the screen for which the user can not only enter in their desired caloric count and their weight but the interface will also display the change of values over time.
Collaborators (uses)	**Controller. setGoal / .AddWeight**

Class Presentation_UI	
Responsibilities	Contains all the required UI components that makes up the screen that will display users dietary nutritional information.
Collaborators (uses)	



Controller Subsystem

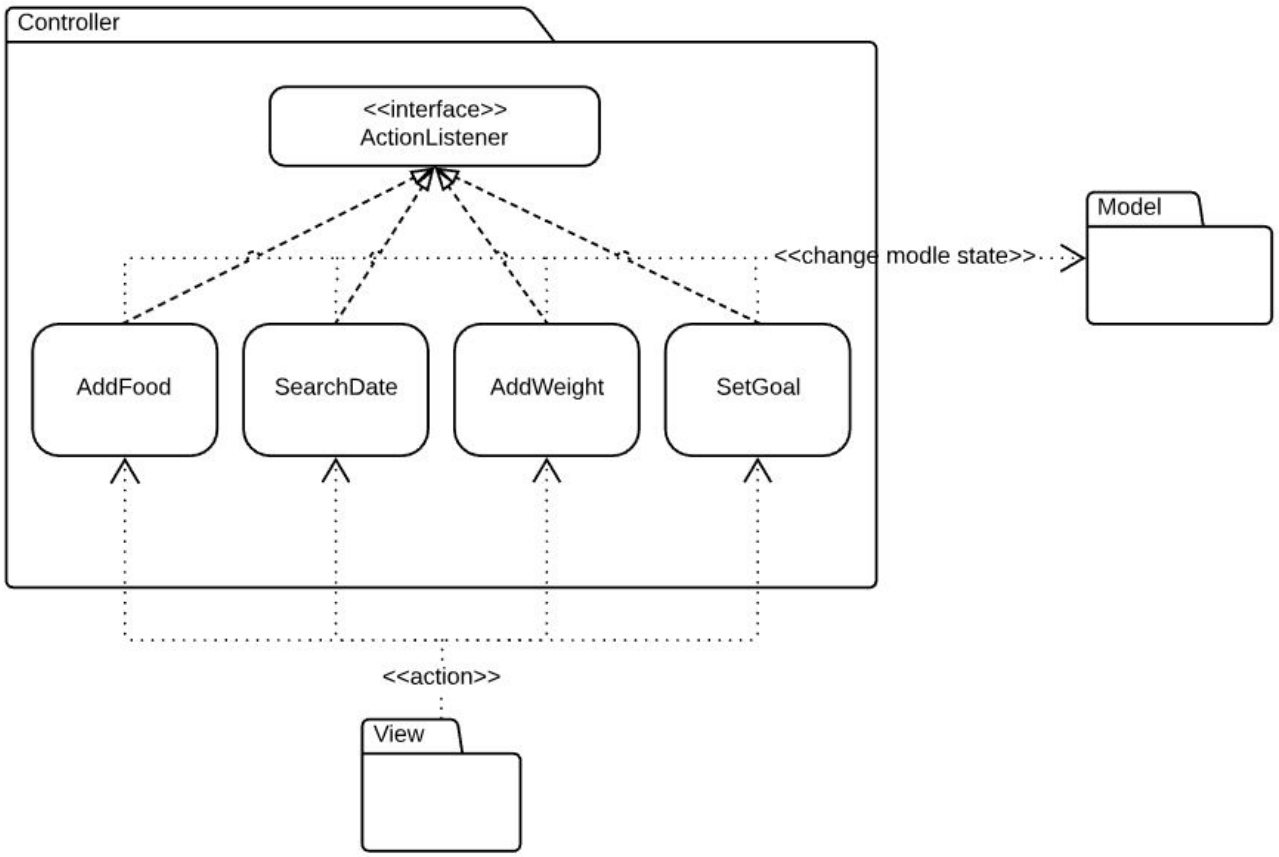
Class AddFood	
Responsibilities	Creates a Food object which will basically add the object to the Daily Log and Food Collection.
Collaborators (uses)	**Model.Food** **Model.Writer** **Model.DietManagerApp**

Still need to work on.

Class SearchDate	
Responsibilities	Searches through the Daily Log using the Date attribute, which filter the results correlated to that selected date
Collaborators (uses)	**Model.DietManagerApp**

Class AddWeight	
Responsibilities	Updates the user's current Weight of that day.
Collaborators (uses)	**Model.DietManagerApp** **Model.User**

Class SetGoal	
Responsibilities	Updates the user's current Weight of that day. / Caloric Limit
Collaborators (uses)	**Model.DietManagerApp** **Model.User**

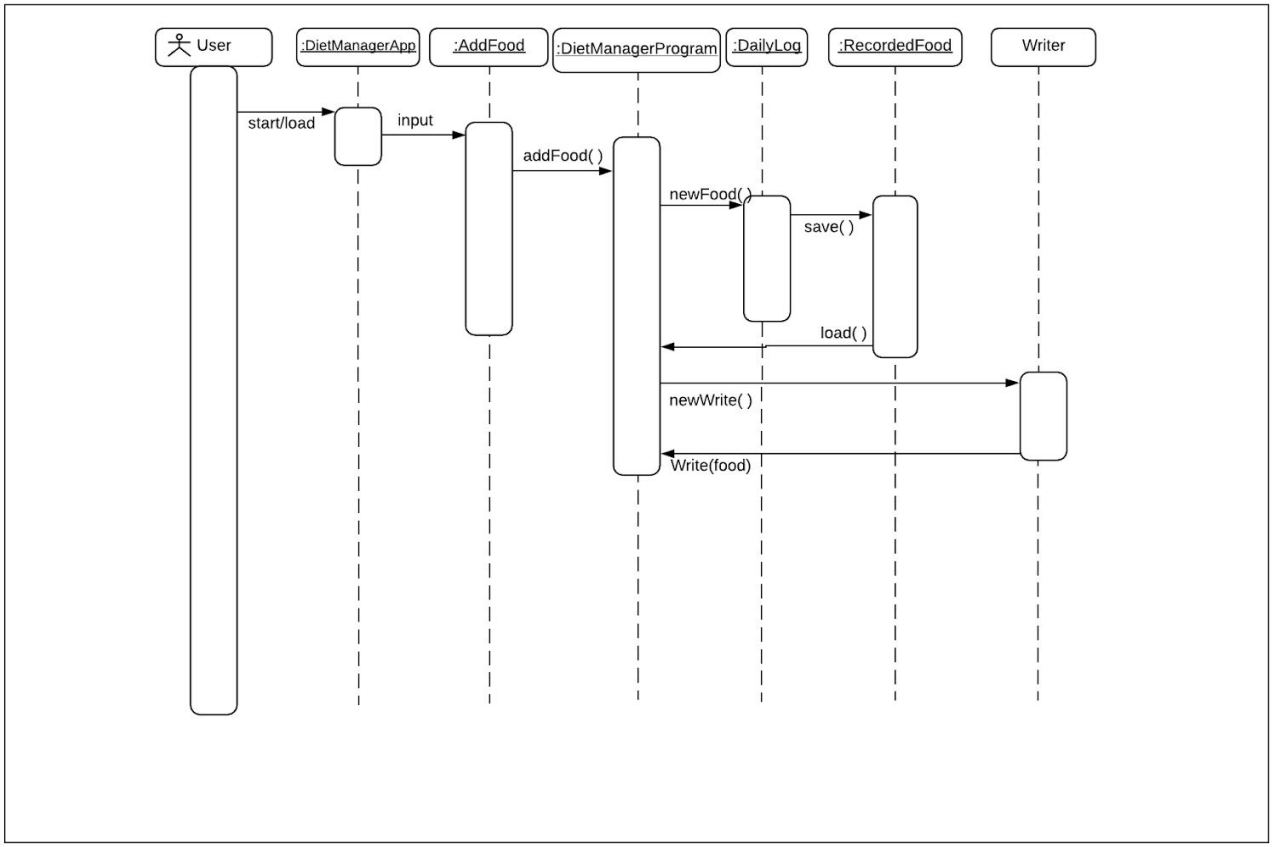


Sequence Diagrams

Sequence description 1:

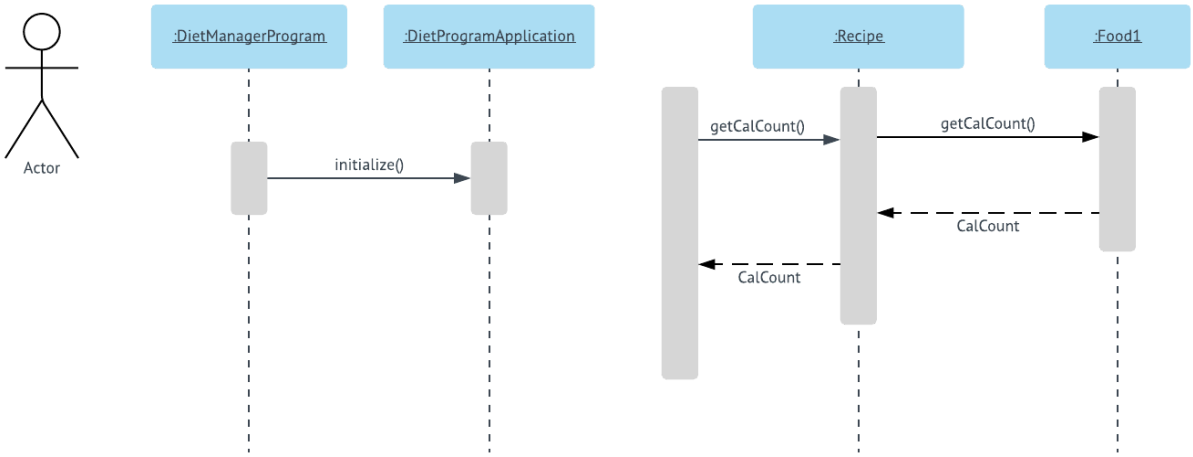
Adding a meal into the daily log

Sequence Diagram #1: Adding a meal into daily log



Sequence description 2:

Getting the total calories from the recipe and its basic foods.



Pattern Usage

Pattern #1 Composite

The Composite pattern is used in order to create a cohesive relationship between the Recipe and BasicFood objects.

Composite Pattern	
Composite(s)	Recipe
Leaf(s)	BasicFood
Component	Food

Pattern #2 MVC

Use the Model View Controller pattern to organize the application.
Example: User wants to add a new basic food.

View: 1. Received “add a new basic food” request from user.
2. Send a request to Controller about the user adding a new basic food.

Controller: 1. Received “user adding a new basic food” request from View
2. Send a request to Model about adding a new basic food for that user.

Model: 1. Received “add a new basic food for the user” from Controller.
2. Adding the new basic food for that user.
3. Send notice to Controller about “New food added”

Controller: 1.Received notice from Model, start collect new set of data.
2. Send notice to View about “new data collected”.

View: 1. Receive notice from Controller about “new data collected”.
2. Display the new data to the User.

MVC Pattern	
Model	DietProgramApplication, Writer, Reader, User, DailyLog, RecordedWeight, RecordedFood, BasicFood, Recipe
Views	DietManager_UI, AddFood, SetGoal, Presentation
Controllers	EventListener, SetGoal, AddFood, SearchDate, AddWeight