Diet Manager Version 2.0

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

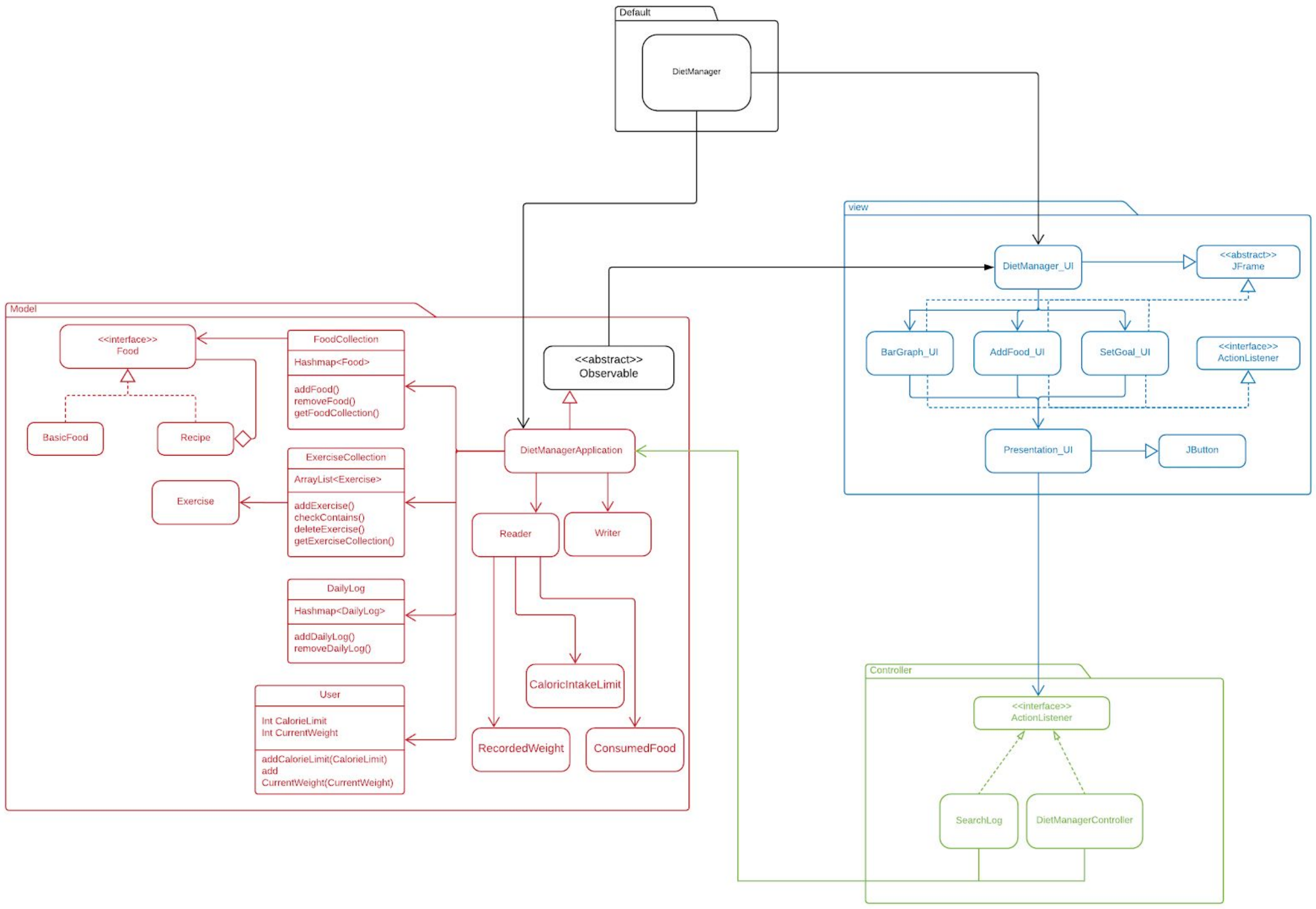
Diet Manager is a program that helps users monitor their diet. Each time a user eats, they can enter in a food item or multiple food items that make up a recipe. For each food item they enter, they must provide the name and nutritional information (i.e. calories, fat, etc.). For each recipe the user 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. Additionally, users are able to enter in exercises to the exercise collection and log specific exercises each day.

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. Users can set a daily calorie limit and Diet Manager will inform the user if they are under or over their daily limit. Additionally, Diet Manager will record the exercises and the calories expended on exercise in order to compute net calories.

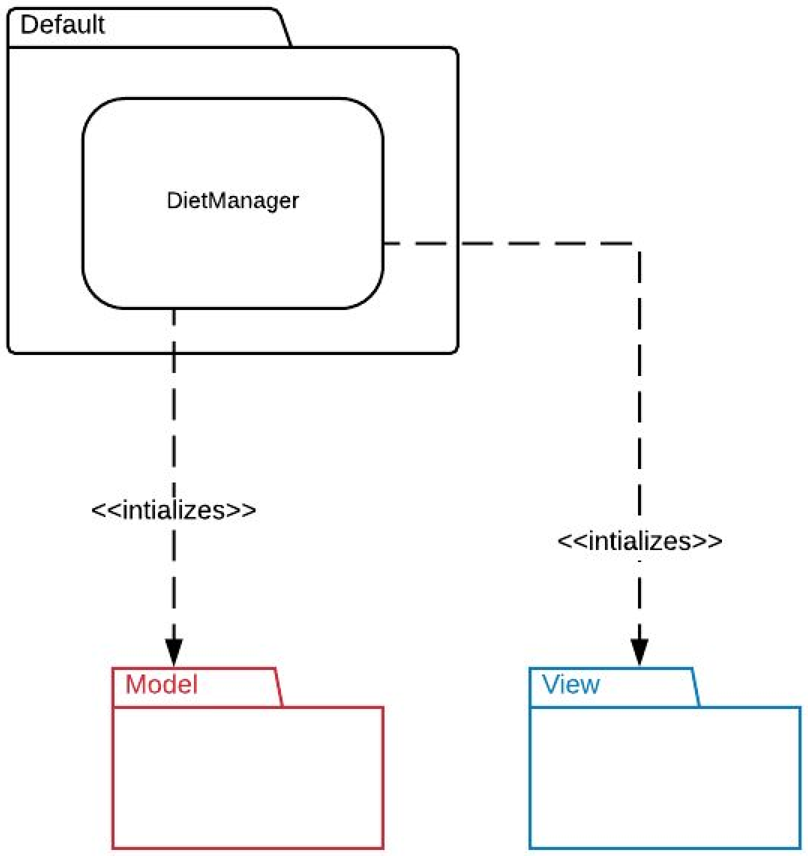
Design Overview

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

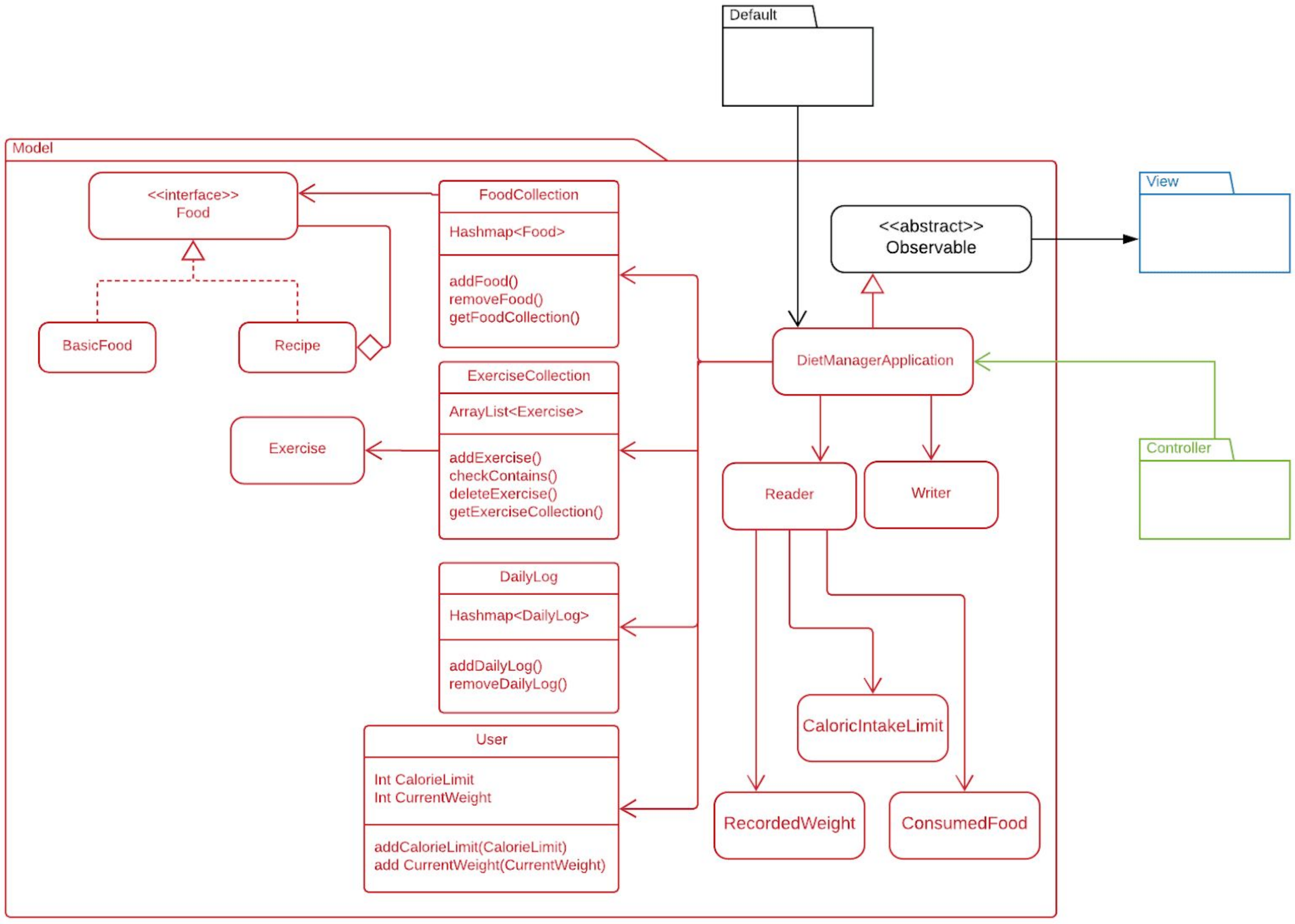


Default Subsystem



Class DietManager	
Responsibilities	Create the model object(s). Create the graphic user interface. Display the GUI for user to used.
Collaborators (uses)	**model.DietManagerApplication -** the primary model class **javax.swing.JFrame** - the outermost window for the application. **java.awt.BorderLayout** - layout manager for the main window **view.BarGraph_UI** - canvas showing the nutrition information **view.Presentation_UI -** displays foodcollection, dailylog, excercise log. **view.SetGoal_UI -** setting goal for weight. **view.AddFood_UI -** adding food, recipe to foodcollection, csv, and dailylog **view.DietManager_UI -** primary view class

Model Subsystem



Class DietManagerApplication	
Responsibilities	Notify observers (UI) of any changes
Collaborators (extends)	**java.util.Observable** - so nutrition information changes can be observed by others
Collaborators (uses)	**java.util.Observer** - for notifications of nutrition information changes to views **Model.DailyLog** **Model.FoodCollection** **Model.ExerciseCollection**

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 **Model.FoodCollection** - Creates an object of Food collection **Model.ConsumedFood -** Creates an object of Consumed Food **Model.DailyLog** **Model.ExerciseCollection**

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

Class User	
Responsibilities	Stores the user’s current weight and desired caloric intake. Update the user’s current weight and desired caloric intake.
Collaborators	**java.io** - to use the PrintWriter and File classes

(uses)	**java.util** - to use Calendar class

Class DailyLog	
Responsibilities	A Hashmap of the object and the date. Takes in consumed food. Takes in values and stores user's recorded weight. Takes in values and stores user's desired caloric limit. Add and Remove from the Hashmap
Collaborators (uses)	**Model.Weight** **Model.ConsumedFood** **Model.CarloricIntakeLimit** **Model.Excercise**

Class ExerciseCollection	
Responsibilities	An Arraylist contain the Exercise object. Following Attributes are the Name(*exercise name)*, Calories (*number of calories expended by the exercise in hour*) addExercice() - Method that add exercise to collection checkContains() - Check for information duplicate deleteExercice() - Method that remove exercise from collection getExerciseCollection() - Method that returns an array of the values of all the attributes listed above. Ands add them to a ExerciseCollection Array.
Collaborators (uses)	**java.util** - to use the Arraylist

Class Exercise	
Responsibilities	Define an Exercise object take in the exercise name and number of calories expended by the exercise in 1 hours for a 100 pound person. getName() - get exercise name getCalorie() - get calorie expended
Collaborators (uses)	**java.util** - to use the Arraylist

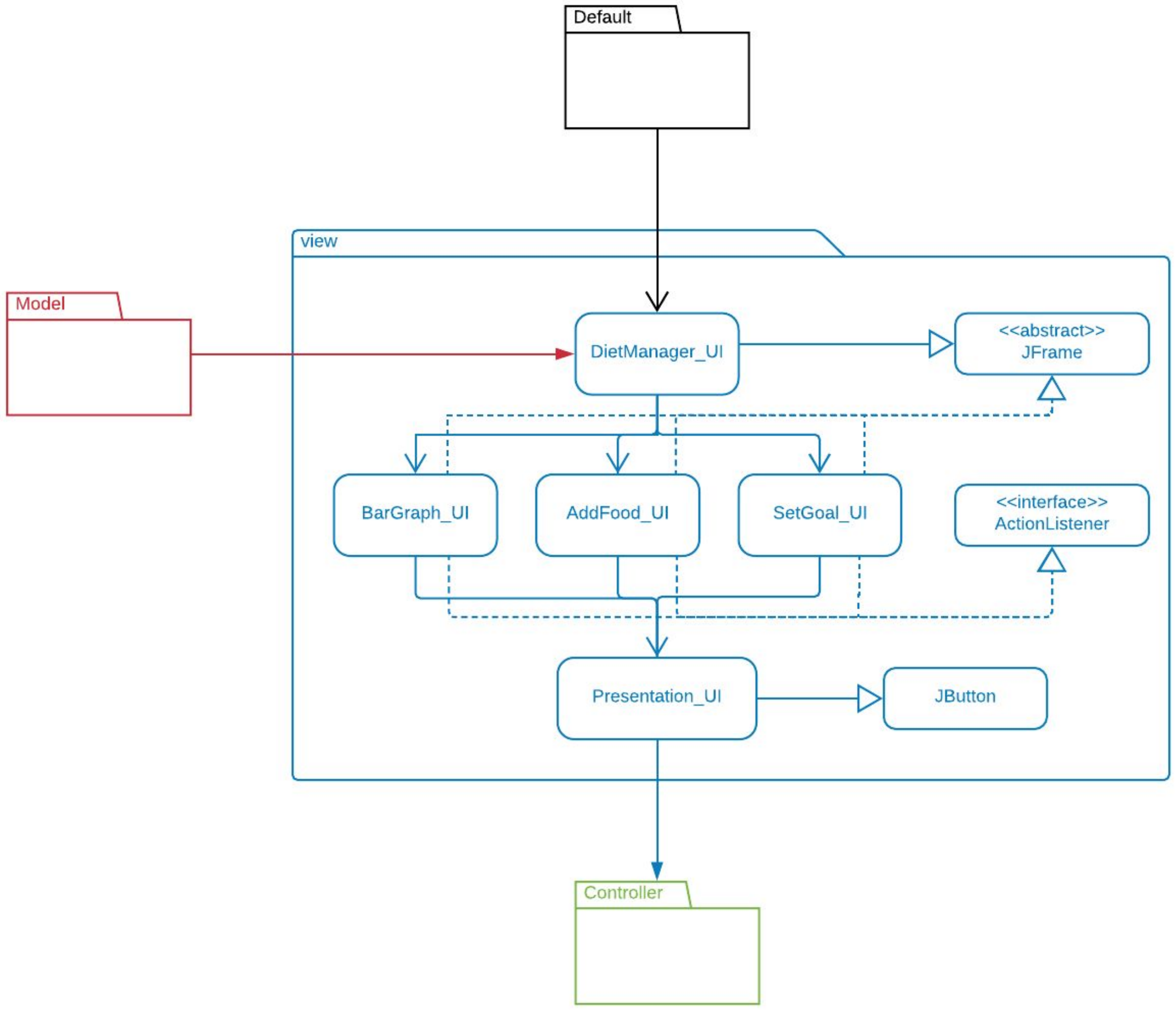
Class FoodCollection	
Responsibilities	An Hashmap that contains the Food object. addFood() - Method that add food to collection removeFood() - Method that remove food from collection getFoodCollectionList()
Collaborators (uses)	**java.util.Hashmap** **Model.Food** **Model.Recipe**

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 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 - java.awt.event.ActionListener**

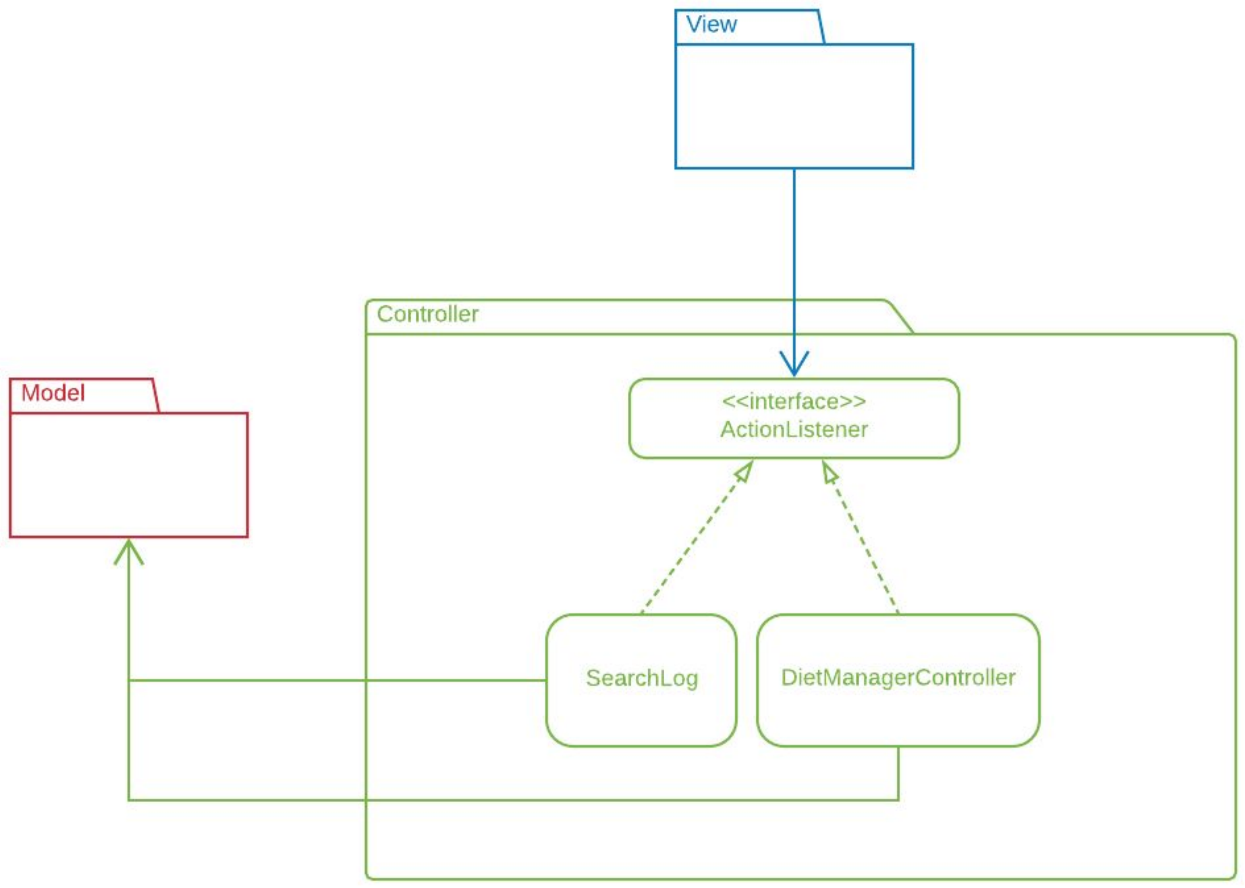
Class BarGraph_UI	
Responsibilities	Shows nutrition information for currently selected day. Respond (as Observer) to updates from DietManagerApplication to change the nutrition information
Collaborators (implements)	java.awt.Canvas - display the bar graph of nutritional information java.util.Observer - to observe DietManagerApplication
Collaborators (uses)	java.util.Observable - to register for change notifications java.awt.Color - control the color of the bar graph java.awt.Graphics - control draw the overall graph java.awt.Rectangle - scale the graph to the actual canvas size model.DietManagerApplication

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. Prints out the daily log, food collections, using tabs.
Collaborators (uses)	

Controller Subsystem



Class DietManagerController	
Responsibilities	Controlling the application execution
Collaborators (uses)	**Model.***

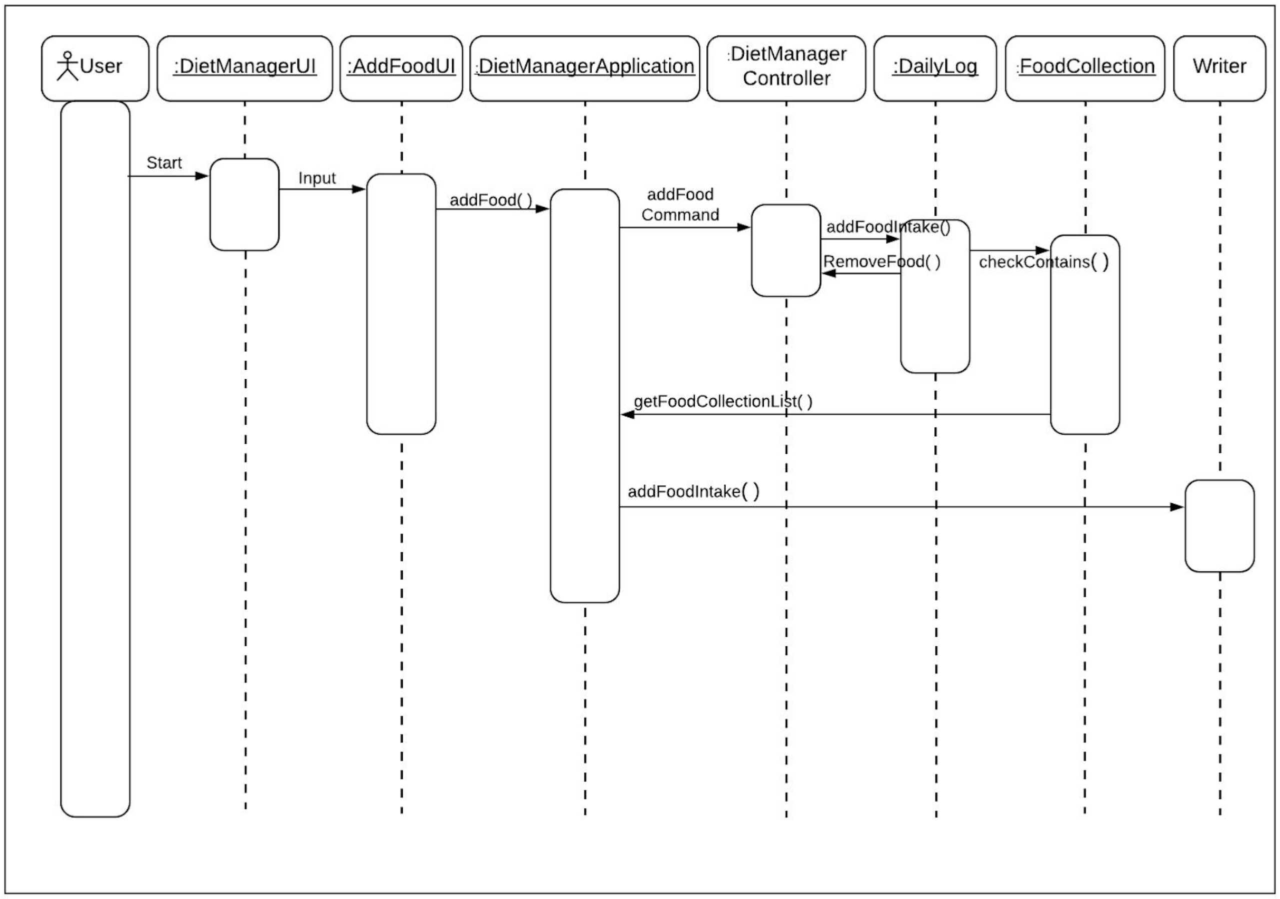
Class SearchLog	
Responsibilities	Searches through the Daily Log using the Date attribute, which filter the results correlated to that selected date
Collaborators (uses)	**Model.DietManagerApplication**

Sequence Diagrams

Sequence Description 1:

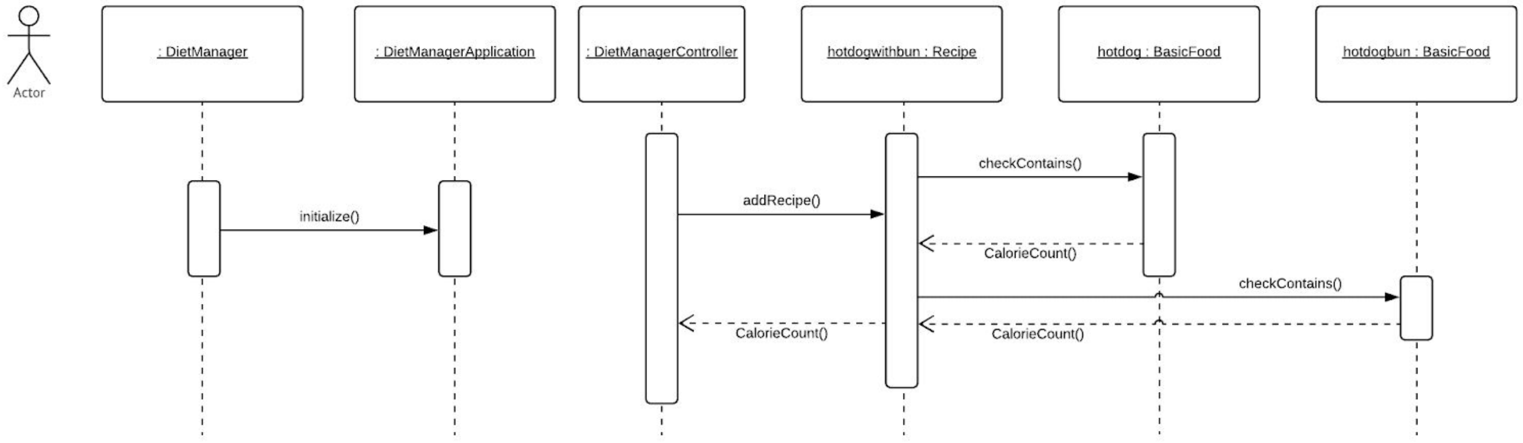
Adding a meal into the daily log

Sequence Diagram #1: Adding a meal into dialy log



Sequence Description 2:

Getting the total calories for the current date — with the recipe checking if it contains the basic foods (e.g., recipe of hotdog with bun).



Pattern Usage

Composite Pattern

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

Model-View-Controller Pattern

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_UI, SetGoal_UI, Presentation_UI, BarGraph_UI
Controllers	DietManagerController

Observer Pattern

Notifies automatically of any change.

Observer Pattern	
Observer	DietManager_UI Presentaion_UI BarGraph_UI
Observable	DietManagerApplication