Diet Manager 2.0

Project Design Document

Used Napkin

Josh Schrader <jas6531@rit.edu>
Neil Hiranandani <nsh2579@rit.edu>
Patrick Lennon <pnl1108@rit.edu>
Sean Decker <sxd7342@rit.edu>
Kevin Gleason <kg8128@rit.edu>

Project Summary

This system is a diet manager that allows users to track their food consumption and monitor diet progress. Users have the ability to log foods and recipes they have consumed. This information will be used to track progress towards various goals. These goals may be set by the user and can include: desired weight, daily caloric intake, weight loss/gain over time, etc.

This log will be maintained on a daily basis and will be stored in an external file. If the user decides to leave the program at any time, it will automatically load any previously saved information when they return. A history of these logs will be maintained so the user can access past information based on a selected date.

The user shall have the ability to add/delete foods and recipes in the collection, which will be maintained in external files. These foods will contain various nutritional information such as calories, carbohydrates, fats, and proteins and the user will be able to select a date to view this nutritional information along with a breakdown of nutrition in terms of percentage for that day. If the user does not log their weight or calorie limit for a a day, the program will automatically grab the last most recent date information that was logged and use that information to log to the weight and calorie limit for the day. This program will also maintain an exercise list and enable user to add an exercise from that list to their current date's diet log. The user will also be able to create their own exercise and add it to the exercise list so other users can add that exercise as well.

Design Overview

From our original noun and verb separations, we've condensed their responsibilities to 10 classes. With the 10 we've implemented the MVC pattern. The Model is consisted of BasicFood, Food, Recipe, User, and DietLog. Within this model, BasicFood, Recipe, and Food make up the composite pattern. The DietController was originally the only component of the controller, but ultimately decided to add a CSV parser to increase cohesion and handle all reading and writing to csv files. DietController is only aware of the User class so it will have access to alter the states of the other classes. With the View, other than having it associated with and initialize the controller, View is also associated with the Abstract Food class. This was added so that the view has the ability to make use of the food data from the User's food. The default package only contains the DietManager class which initializes the view.

We decided to add in the class called FoodList which is responsible for holding the global list of all food for every user. This list in then populated from a specified csv file on startup. We determined the user will not be able to delete a food from the global food list because it would cause unwanted issues with user's deleting foods that other users may want to use or have already used. This would then be punishing to users using the application as it would require them to add in food again that another user chose to delete.

The CSVParser class morphed into a controller instead of just being a strict util class for DietController. We decided that it was easier for the CSVParser to have an association with the model and be able to control the specifics of how each item would be saved to the respective csv files.

We determined that having the User contain a history of DietLog objects via a hashmap, instead of having only one DietLog object that represented the day. This enabled us to have an easier reference to each DietLog's information and the ability to correlate the information we retrieved from the Food List. This also enabled us to quickly search if we had a DietLog in existence for a specified date.

From an oversight in the initial design of the system, we needed to add a copy function to the Abstract Food class and implement in both Food and Recipe classes in order to prevent overwriting the serving size of each similar food in the log. We are thinking about removing serving size of Food altogether and making it a separate class down the line.

From the initial skeleton of the DietLog class, there has been the addition of many functions that perform calculations in order to return nutritional information about that specific instance of the object.

For the view we agreed that we are using the terminal for the View in the MVC architecture. The view as of right now, uses Scanner to handle all user navigation through the system. This is going to be overhauled to a GUI at a later date.

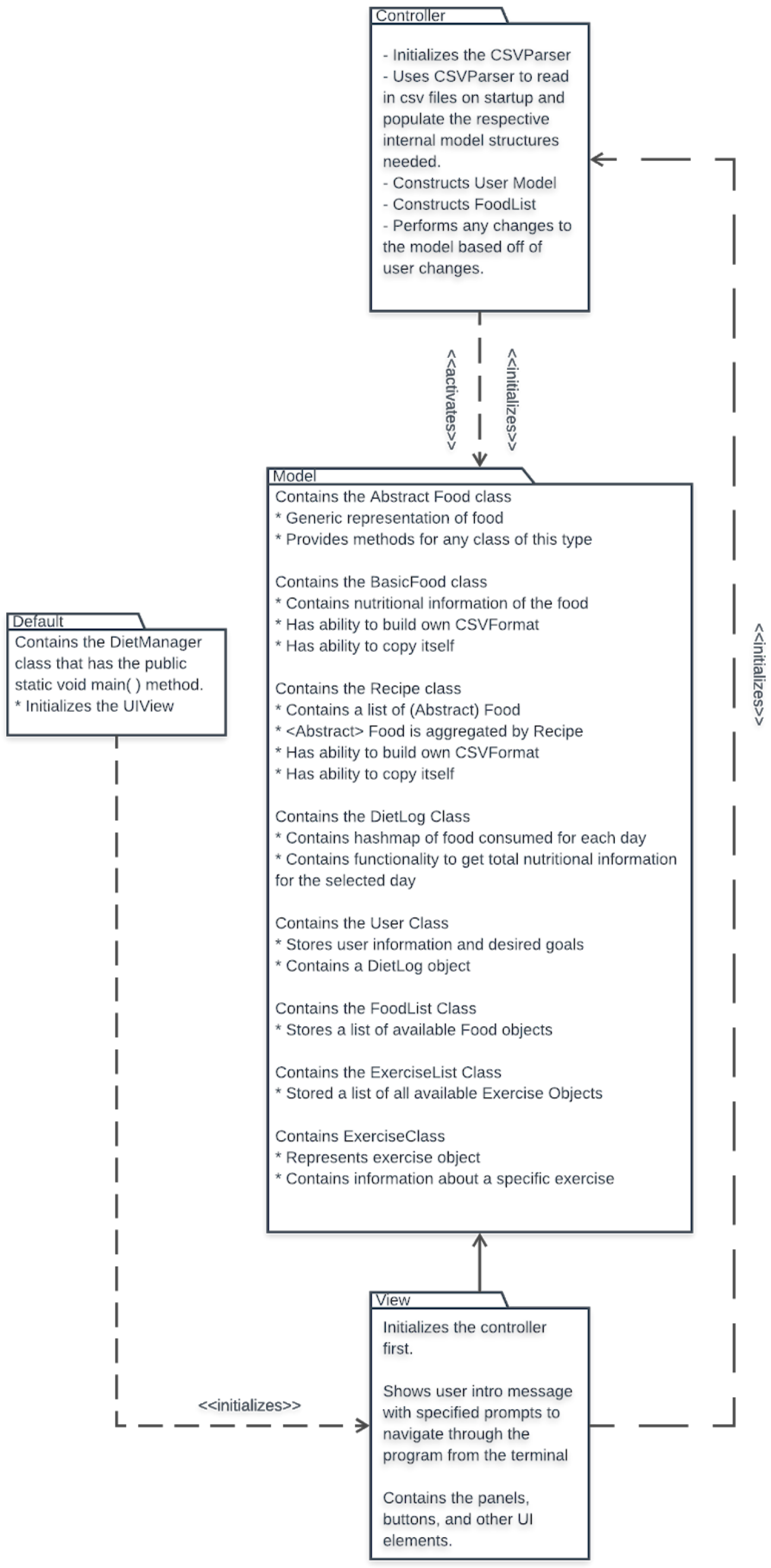
For the additional exercise functionality, We're going to make 2 classes: Exercise and ExerciseList. We're treating these two classes as FoodList and BasicFood respectively. Instead of making a composite pattern for different types of exercise, the user will just enter their time in terms of minutes and it'll be logged in term of calories burn per hour. The DietLog will have a loose aggregation with Exercise, and the ExerciseList will have a solid aggregation with Exercise. DietLog will make whatever changes needed to the users caloric information based off of the exercises it may or may not receive.

We're going to set up the GUI and the Barometer later after we fully incorporated the Exercise functionality and alter the functions for returning nutritional information.

We elected to use multiple views to avoid one large class that handled all of the logic of swapping view components in and out of the frame. Something of that nature can end up being very complex and hard to maintain. As a solution, we've elected to create a single class to handle the frame itself and the swapping of subviews in and out of the frame. The subviews themselves, all inherit from an abstract View class so they share similar functionality and can be treated the same by the DietFrame class. This allows DietFrame to simply call a common function of View to prepare and add the needed components to the frame.

As a design decision, to avoid the DietController class from becoming a static class, we chose to pass the DietController object to each subview.

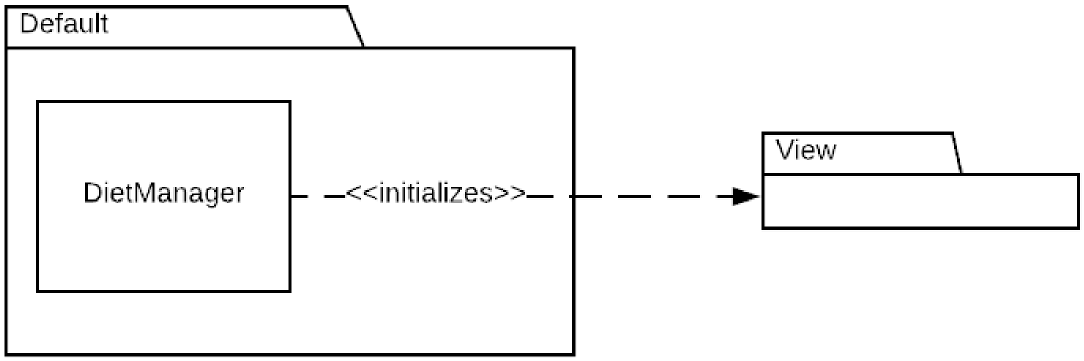
Subsystem Structure


Controller
- Initializes the CSVParser
- Uses CSVParser to read in csv files on startup and populate the respective internal model structures needed.
- Constructs User Model
- Constructs FoodList
- Performs any changes to the model based off of user changes.
<<activates>>
<<initializes>>
Model
Contains the Abstract Food class
* Generic representation of food
* Provides methods for any class of this type
Contains the BasicFood class
* Contains nutritional information of the food
* Has ability to build own CSVFormat
* Has ability to copy itself
Contains the Recipe class
* Contains a list of (Abstract) Food
* <Abstract> Food is aggregated by Recipe
* Has ability to build own CSVFormat
* Has ability to copy itself
Contains the DietLog Class
* Contains hashmap of food consumed for each day
* Contains functionality to get total nutritional information for the selected day
Contains the User Class
* Stores user information and desired goals
* Contains a DietLog object
Contains the FoodList Class
* Stores a list of available Food objects
Contains the ExerciseList Class
* Stored a list of all available Exercise Objects
Contains ExerciseClass
* Represents exercise object
* Contains information about a specific exercise
Default
Contains the DietManager class that has the public static void main( ) method.
* Initializes the UIView
<<initializes>>
<<initializes>>
View
Initializes the controller first.
Shows user intro message with specified prompts to navigate through the program from the terminal
Contains the panels, buttons, and other UI elements.

Subsystems

Default Subsystem

Class DietManager	
Responsibilities	- Initializes the View - Display the GUI so that the application can be used
Collaborators (uses)	**DietView** - the GUI for the application



Model Subsystem

Class Food (Abstract)	
Responsibilities	- Provides a generic structure for both the Basic Food class and Recipe class - Contains two ArrayLists of strings for recipe to use. One for the children and the other for the recipe servings of each of those children. - Provides those classes with basic functionality that both should be able to accomplish as subclasses.
Collaborators (uses)	(Doesn't need to collaborate with anything)

Class Basic Food	
Responsibilities	- Represents a basic food in the collection of foods. - Provides nutritional information about the food itself such as calories, fat, protein, and carbs - Can be used in recipes - Has the ability to copy itself - Ability to build its own csv format and returns it
Collaborators (inheritance)	**Model.Food** - contains basic nutritional information and basic functions that the BasicFood class needs to have.

Class Recipe	
Responsibilities	- Represents a recipe - Maintains a list of Food objects that the recipe is composed of - Has the ability to copy itself - Builds own csv format and returns it - Ability to remove a child of itself - Ability add a child - Needs to be able to return it's children
Collaborators (inheritance)	**Model.Food** - contains basic nutritional information and basic functions that the Recipe class needs to have.

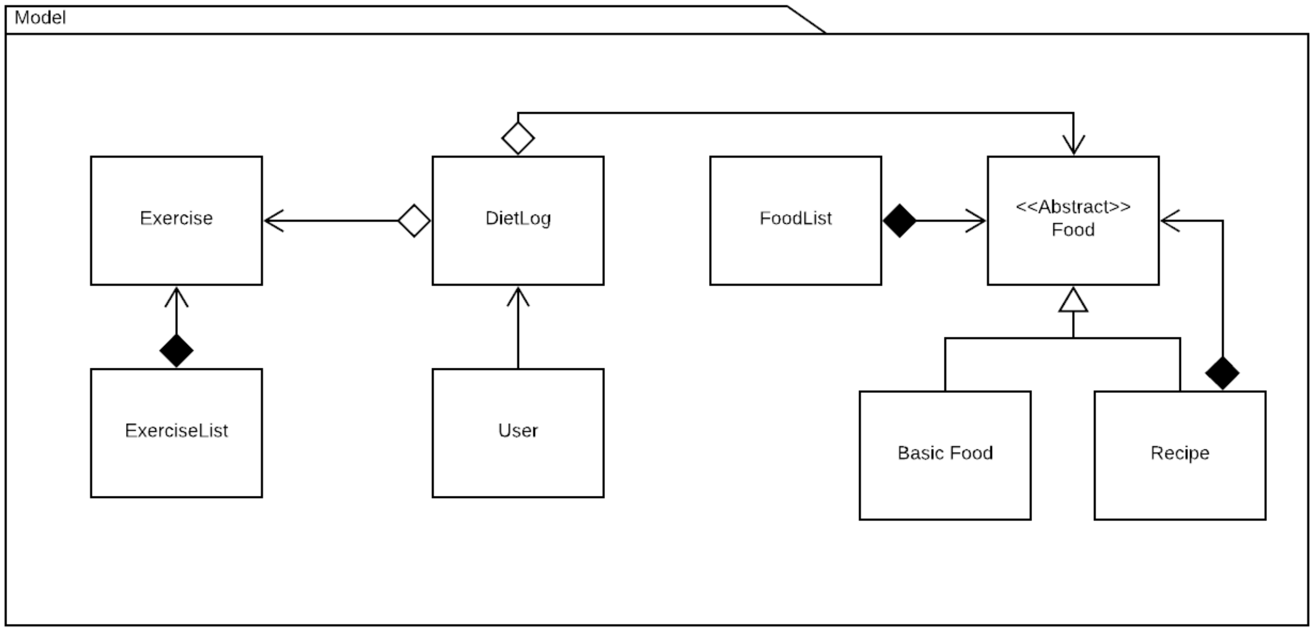
Class DietLog	
Responsibilities	- Store a daily log of food consumed in ArrayList - Contains calculation functions to be able to provide the user nutritional information from the log of that day. - Ability to log a food item that the user has specified. - Ability to delete a food item that the user has specified. - Ability to set daily caloric intake limit - Ability to set weight for the specified date - Functionality to add up total nutritional information and create a breakdown of information in percentages and return that info.
Collaborators (uses)	**Model.Food** - contains an arraylist of Food objects that keep reference to what has been logged. **Java.Util.Date -** The current date is set to a formatted Date for the daily log when it's initialized

Class FoodList	
Responsibilities	- Store the list of available foods - Have the ability to add a food to the global food list but the user cannot delete a food from the global food list. - Has the ability to go in the map of foods and search for a String key that is passed in. - Ability to return the food map.
Collaborators (uses)	**Model.Food** - contained in the list of available foods

Class User	
Responsibilities	- Stores user goals - Set and stores any other user information (name, weight, etc.) - Contains and maintains a history of DietLogs via a hashmap - Search through the history of DietLogs and return a DietLog object that is found in the list. - Return the current date's log
Collaborators (uses)	**Model.DietLog -** contains the daily log history of food consumed by the user

Class ExerciseList	
Responsibilities	- stores an ArrayList of Exercise objects as available exercises - ability to return an ArrayList of Exercise objects - ability to search the list of available exercises for exercise desired - ability to add an exercise to the exercise list
Collaborators (uses)	**Model.Exercise** - the Arraylist in ExerciseList will be populated with Exercise objects

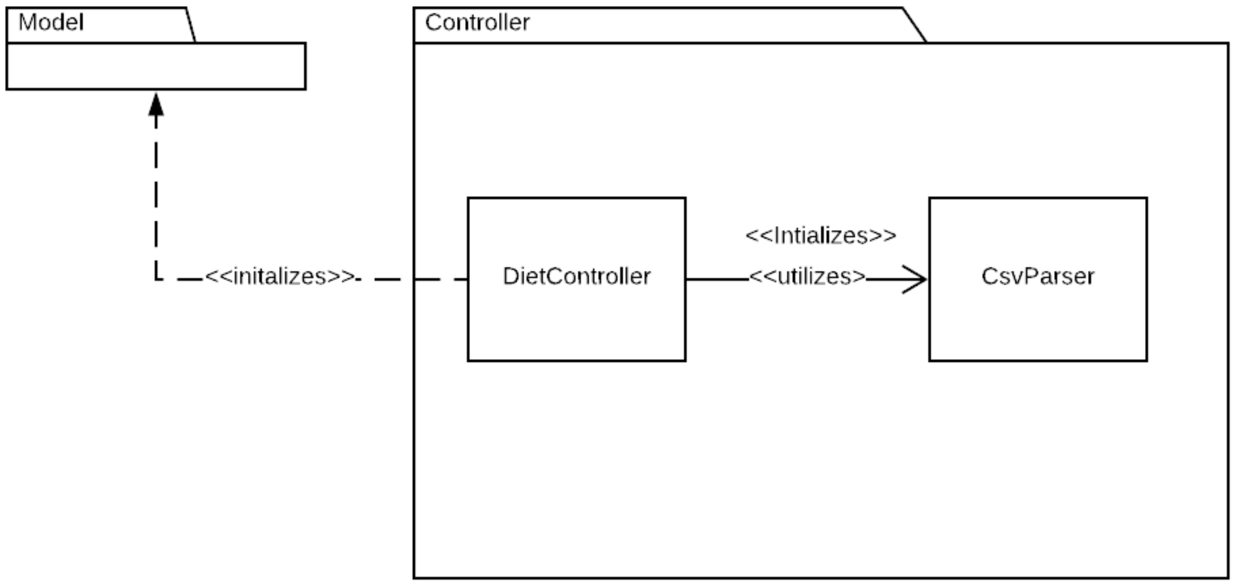
Class Exercise	
Responsibilities	- contains name and the amount of calories burned per hour - functions to get/set both attributes
Collaborators (uses)	(Doesn't need to collaborate with anything)



Controller Subsystem

Class DietController	
Responsibilities	- Uses the CSVParser to read in data and using it to construct model objects - Makes changes to the model based on user events - Uses the CSVParser to save changes to csv on application exit - Provides functions to update the User's goals and Diet Logs
Collaborators (uses)	**Controller.CSVParser -** used to perform csv reading and writing **Model.FoodList -** used to access and modify available foods **Model.User** - used to update the User's log history and goals

Class CSVParser	
Responsibilities	- Responsible for reading and writing directly to csv files - Populate FoodList, DietLog, and ExerciseList on start of program after reading csv files
Collaborators (uses)	**java.io.FileWriter -** used to write to csv **java.io.FileReader -** used to read from csv **java.io.BufferedReader** - used in conjunction with FileReader **Model.FoodList -** gets populated based on csv data **Model.DietLog** - gets populated based on csv data **Model.ExerciseList** - gets populated based on csv data



View Subsystem

Class DietView (TERMINAL)	
Responsibilities	- displays the data to the user - constructs the controller - going to be opened up in terminal as of right now
Collaborators (uses)	**Controller.DietController** - used to connect user interaction to application logic **Model.Food** - contains the data that is displayed by the view

Class DietFrame (GUI)	
Responsibilities	- displays the window to the user - constructs the controller - manages all sub views
Collaborators (uses)	**Controller.DietController** - used to connect user interaction to application logic **View.View** - used to maintain the current view

Class View	
Responsibilities	- represents an abstract view - all sub views extend this class - provides functions to build different views
Collaborators (uses)	- no collaborators

Class LayoutUtil	
Responsibilities	- provides utility functions for building UI layouts
Collaborators (uses)	**View.View** - subviews will make use of this class when building layouts

Class DietLogView	
Responsibilities	- responsible for displaying current date's DietLog - allows user to view DietLog from previous dates - allows user to navigate to the LogFoodView - displays nutritional information along with exercise for current DietLog being shown
Collaborators (uses)	**Controller.DietController** - used to obtain information to display in the view **View.DietFrame** - shows other subviews as directed by this view **Model.DietLog** - using this to display information about the DietLog in the view **View.LogFoodView** - can navigate to this view in order to log a food

Class FoodListView	
Responsibilities	- responsible for displaying all foods in the FoodList - allows user to navigate to the AddFoodView
Collaborators (uses)	**Controller.DietController** - used to obtain information to display in the view **View.DietFrame** - shows other subviews as directed by this view **Model.FoodList** - uses this class to display information about the FoodList in the view **View.AddFoodView** - can navigate to this view in order to add a food

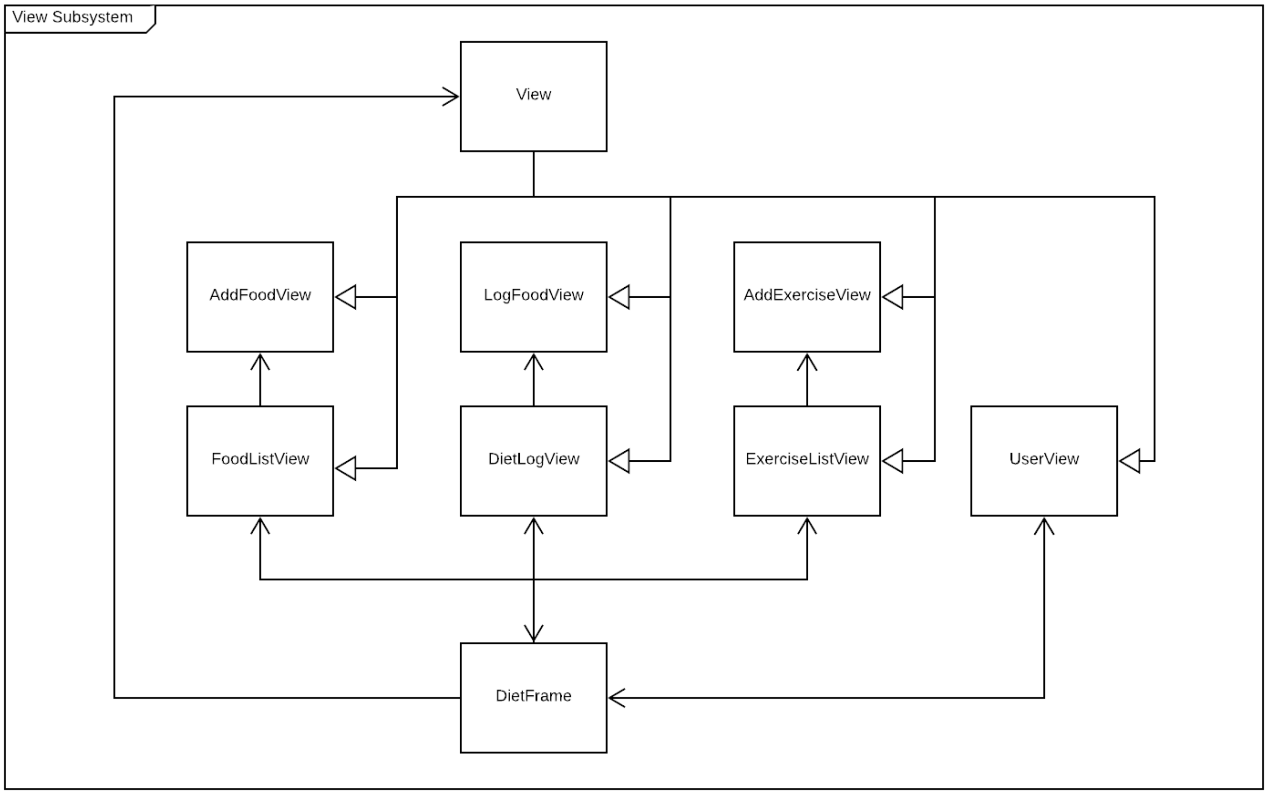
Class AddFoodView	
Responsibilities	- allows user to add a food to the FoodList
Collaborators (uses)	**Controller.DietController** - used to add the food to the FoodList **View.DietFrame** - shows other subviews as directed by this view

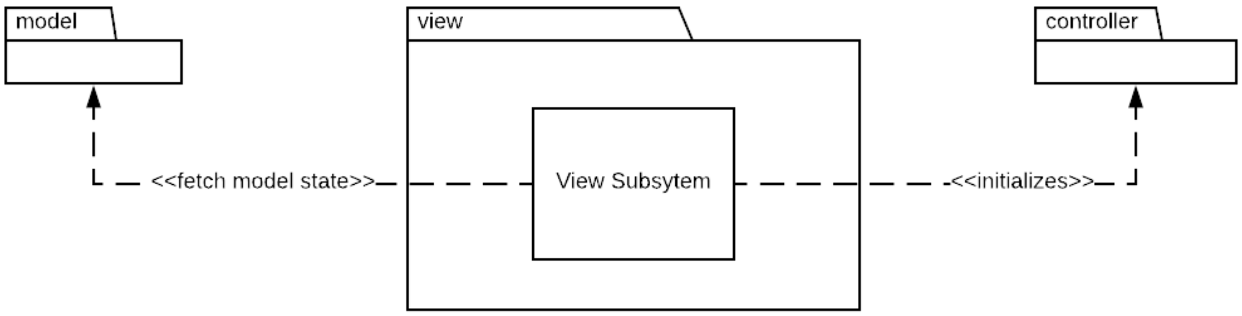
Class LogFoodView	
Responsibilities	- allows user to log a food to a DietLog
Collaborators (uses)	**Controller.DietController** - used to log a food into the DietLog **View.DietFrame** - shows other subviews as directed by this view

Class ExerciseListView	
Responsibilities	- responsible for displaying all exercises in the ExerciseList - allows user to navigate to the AddExerciseView
Collaborators (uses)	**Controller.DietController** - used to obtain information to display in the view **View.DietFrame** - shows other subviews as directed by this view

Class AddExerciseView	
Responsibilities	- allows user to add an exercise to the ExerciseList
Collaborators (uses)	**Controller.DietController** - used to add an exercise to the DietLog **View.DietFrame** - shows other subviews as directed by this view

Class UserView	
Responsibilities	- displays goals and nutritional information for the user
Collaborators (uses)	**Controller.DietController** - used to obtain information to display in the view **View.DietFrame** - shows other subviews as directed by this view

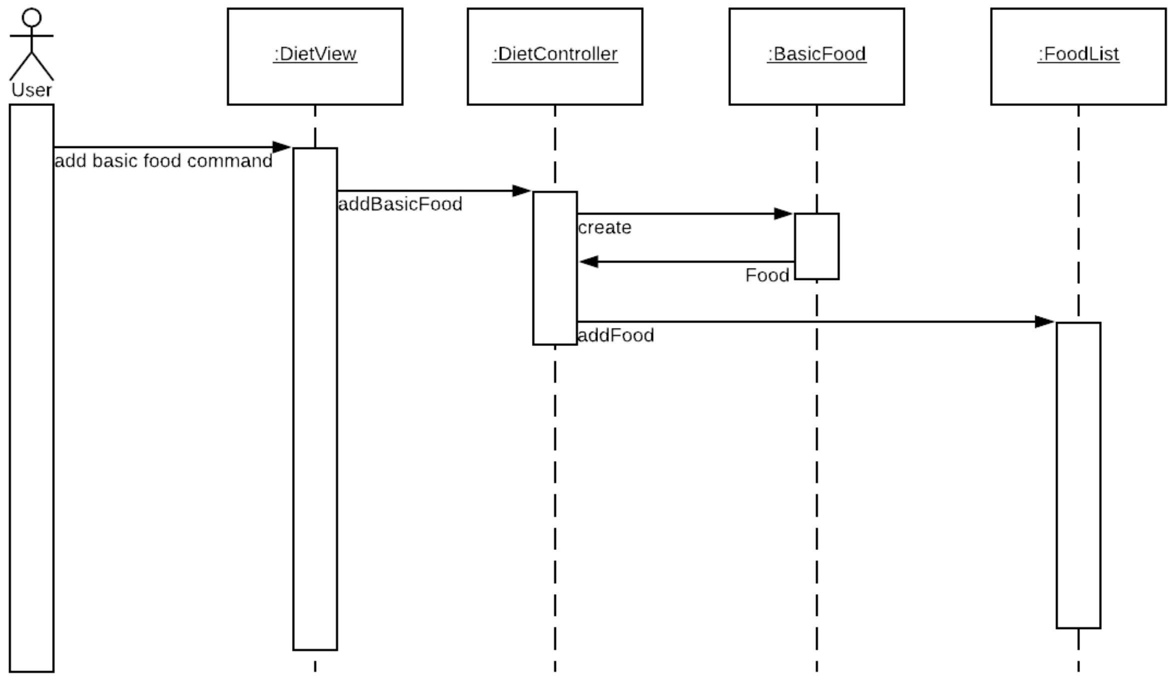

View Subsystem
View
AddFoodView
LogFoodView
AddExerciseView
FoodListView
DietLogView
ExerciseListView
UserView
DietFrame


model
view
controller
View Subsytem
<<fetch model state>>
<<initializes>>

Sequence Diagrams

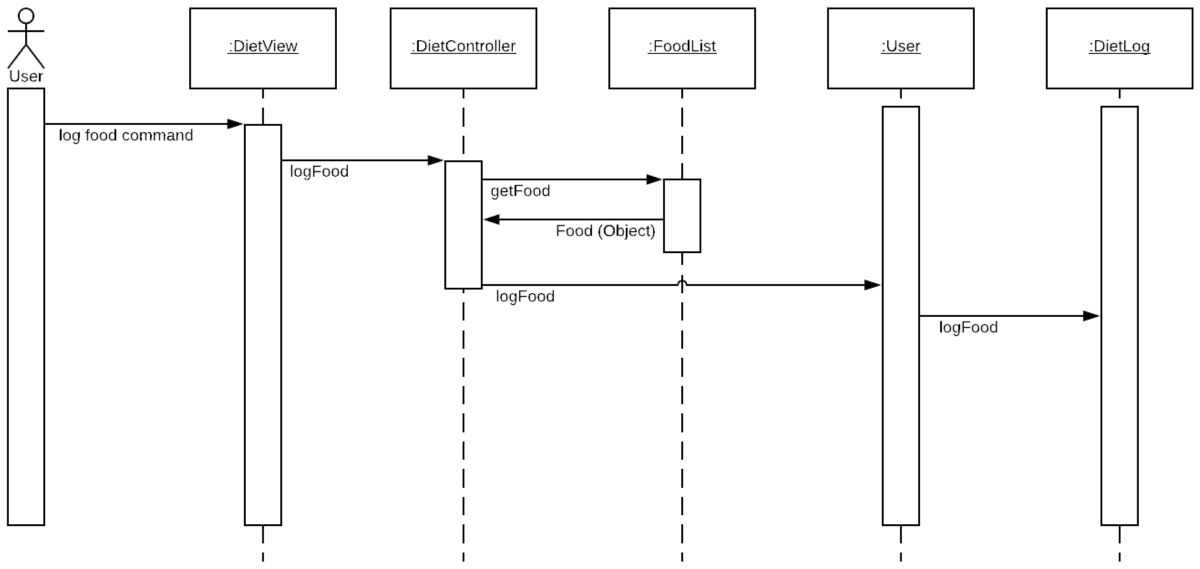
Sequence 1 - Add Food to Available Foods

The user wants to add a new food to the collection of available foods.



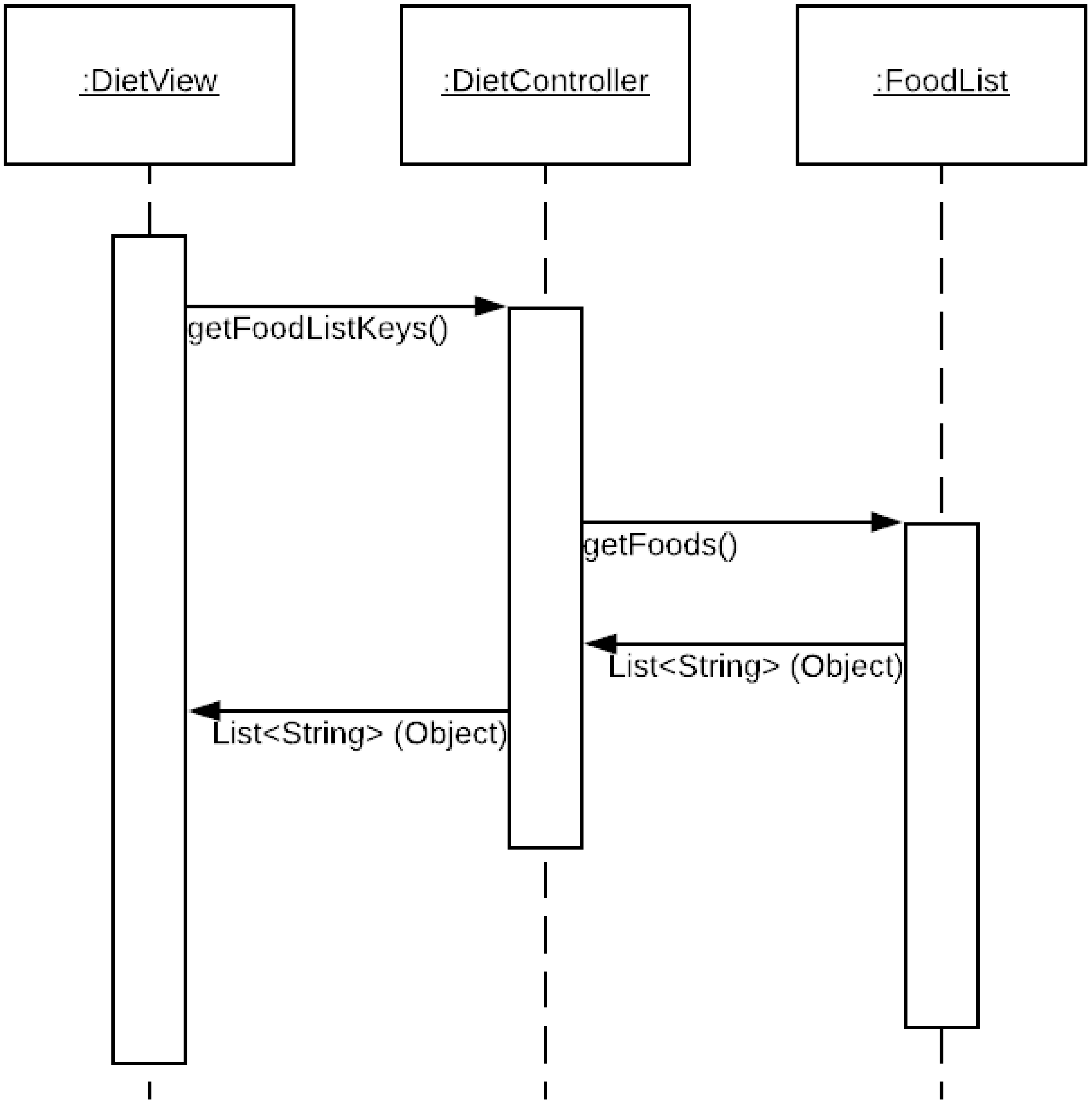
Sequence 2 - Log Food Consumed

The user want to log an existing food they have consumed into their daily diet log.



Sequence 3 - Display List of Available Foods

The view displays available foods on several circumstances.



Pattern Usage

Pattern #1 Composite Pattern

Composite Pattern	
Component	Food <Abstract>
Composite	Recipe
Leaf	BasicFood

Pattern #2 MVC Pattern

MVC Pattern	
Model	Food <Abstract> BasicFood Recipe FoodList User DietLog
Views	DietView
Controller	DietController CSVParser