Diet Manager - Version 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 as well as calories expended through exercise and monitor diet progress. Users have the ability to log foods and recipes they have consumed and log exercises completed. 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.

These logs 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 foods, recipes, and exercises 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 shown through a bar graph. The exercises will also contain information pertaining to how many calories are burned for the particular exercise in one hour. 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 the user to add/delete an exercise to/from that list for their current date's diet log.

Design Overview

From our original 10 classes, we have expanded the program across double that. While most of those classes are different pieces of the user interface, we've used this expansion to solidify our implementation of the MVC pattern. The Model currently consists of BasicFood, Food, Recipe, FoodList, Exercise, ExerciseList, User, and DietLog. Within this model, BasicFood, Recipe, and Food use the composite pattern to maximize code efficiency. The DietController was originally the only component of the controller, but we 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 Model so that it will have the ability to alter the states of the other classes. Our View consists of several SwingUI classes in order to separate an easily modify each individual part of the interface. Other than having the View associated with and initialize the controller, it is also associated with the Abstract Food class, BasicFood, DietLog, and ExerciseList classes. These were added so that the view has the ability to make use of the functions needed from the models in order to pull accurate information needed for displaying in the view. The default package only contains the DietManager class which initializes the view.

We added in the class FoodList, which is responsible for holding the global list of all foods and recipes for every user. This list is populated every time the program is started after reading from a CSV file.

We determined that the user should not be able to delete a food from the global food list, as it would cause unwanted issues with certain users deleting foods that other users may want to, or already do use. This would make the program more difficult to use as it would require them to add in a food that was previously added, but another user chose to delete.

In addition to a FoodList class, we also implemented a new class called ExerciseList. Similar to FoodList, ExerciseList is responsible for getting the list of exercises the users have entered and can log. This list contains information as to the name of exercises, as well as the number of calories burned per hour for a 100 pound person. To populate the list, all the user has to do is load the program, and the list is automatically grabbed and parsed from a CSV file.

The CSVParser class morphed into a controller instead of just being a strict utility class for DietController. We decided that it was more logical 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 as well as incorporating exercise into the calculations about the 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, based on their weight. The DietLog and ExerciseList will have a loose aggregation with Exercise. The DietLog will make whatever changes needed to the users caloric information based off of the exercises it may or may not receive.

For the ExerciseList, we have decided to treat it in the same fashion as we are the FoodList, as in we are not allowing the user to be able to delete an exercise from the global list. We talked to the client

about this change and Dr. E. Atwell gave us the go ahead to not allow users to change the global lists as it would conflict with other user's that are also using the program.

We're going to set up the GUI and the BarGraph 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.

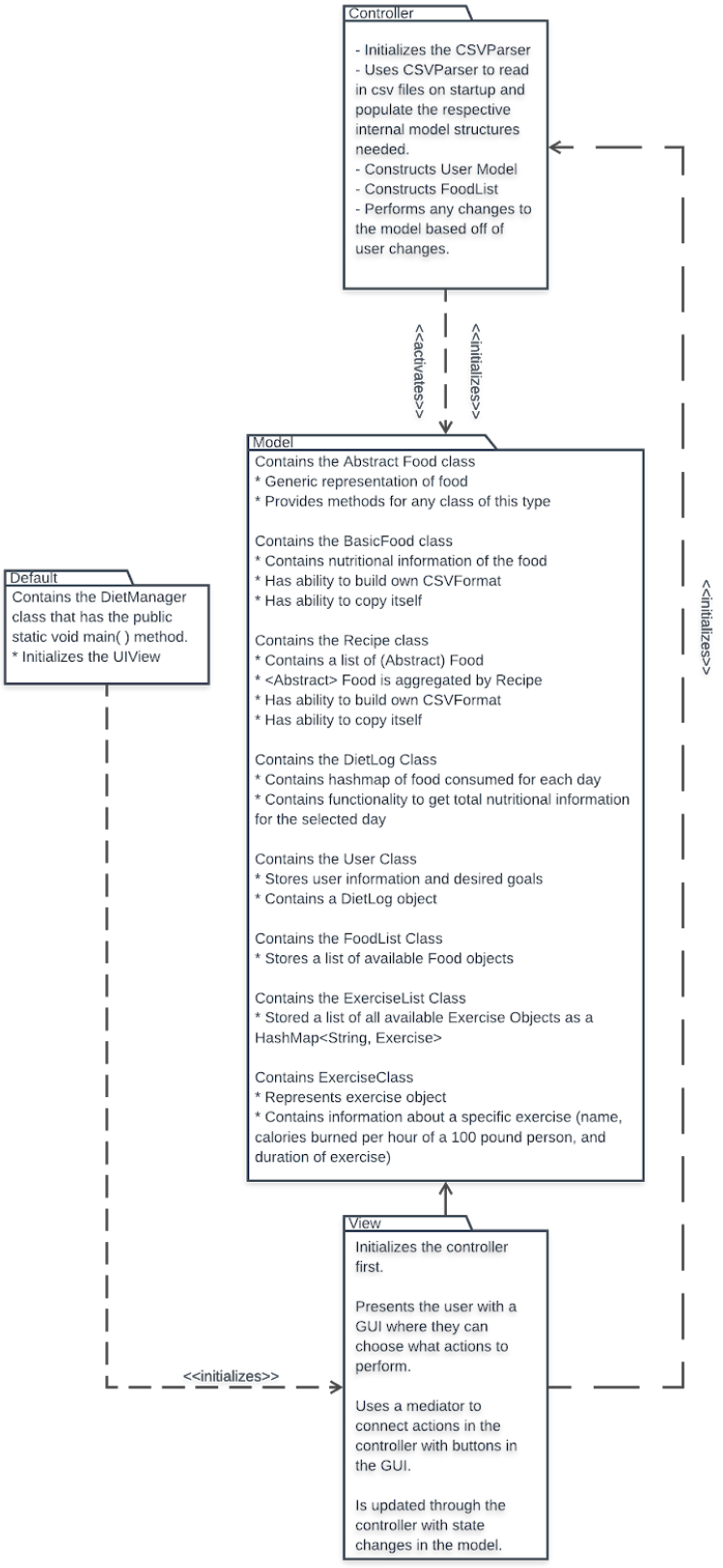
As a design decision, to avoid the DietController class from becoming a static class, we chose to pass the DietController object to each subview.

For the initial implementation of the view, we elected to try and use IntelliJ's built in UI builder as a way to create our GUI. We unfortunately discovered that using IntelliJ's UI builder produced code that was too difficult to work with and implement further functionality with our program. We ended up throwing out what the builder had produced and proceeded to make a GUI from scratch which proved to be a much simpler and more malleable approach.

Upon implementation of the View, we made the decision to use the mediator pattern and introduce the ViewMediator class and pass it in to many of the subviews in order to reduce the coupling that would otherwise be overwhelming on every component.

For designing how the GUI looked, we settled on creating three different main panels. These panels are the DailyValuesPanel, ChartsPanel, and the DietManagerTabbedPane each handling their own subviews. The ChartsPanel holds the BarGraph that is rendered whenever the user has a food that is logged into the system. The DailyValuesPanel shows the users name, weight and nutritional information as well as having two JScrollPanes to show the foods and exercises they have logged for that current date. It also gives the user the ability to delete food or exercises from their logs. The user can also change the date of the current log they are viewing to a different date and it will render the logs for that date in the view. The DietManagerTabbedPane consists of four tabs, User, Food, Exercise, and Log. In the user tab, the user can set their name and weight as well as save. In the food tab, the user has the ability to add a BasicFood or Recipe to the global lists. In the exercise tab, the user has the ability to add an exercise to the global exercise list. In the log tab, the user has the ability to add either a BasicFood or a Recipe as well as an Exercise to the user's daily log.

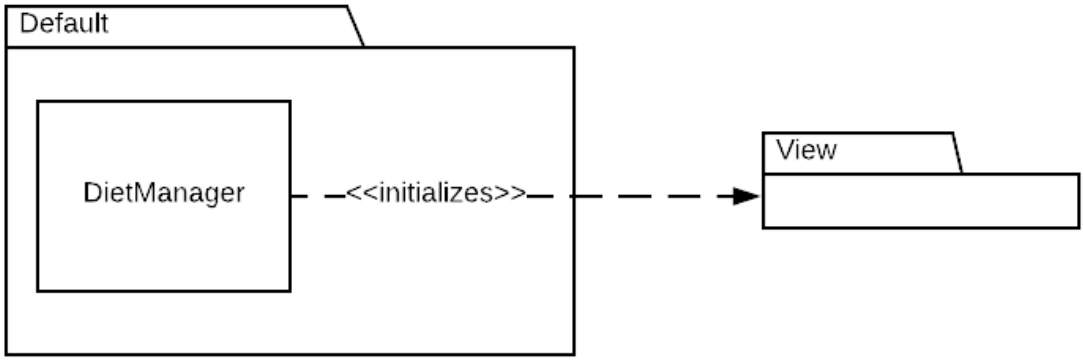
Subsystem Structure



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 - ability to return total nutritional info such as calories, carbs, proteins, and fats
Collaborators (inheritance)	**Model.Food** - contains basic nutritional information and basic functions that the Recipe class needs to have.

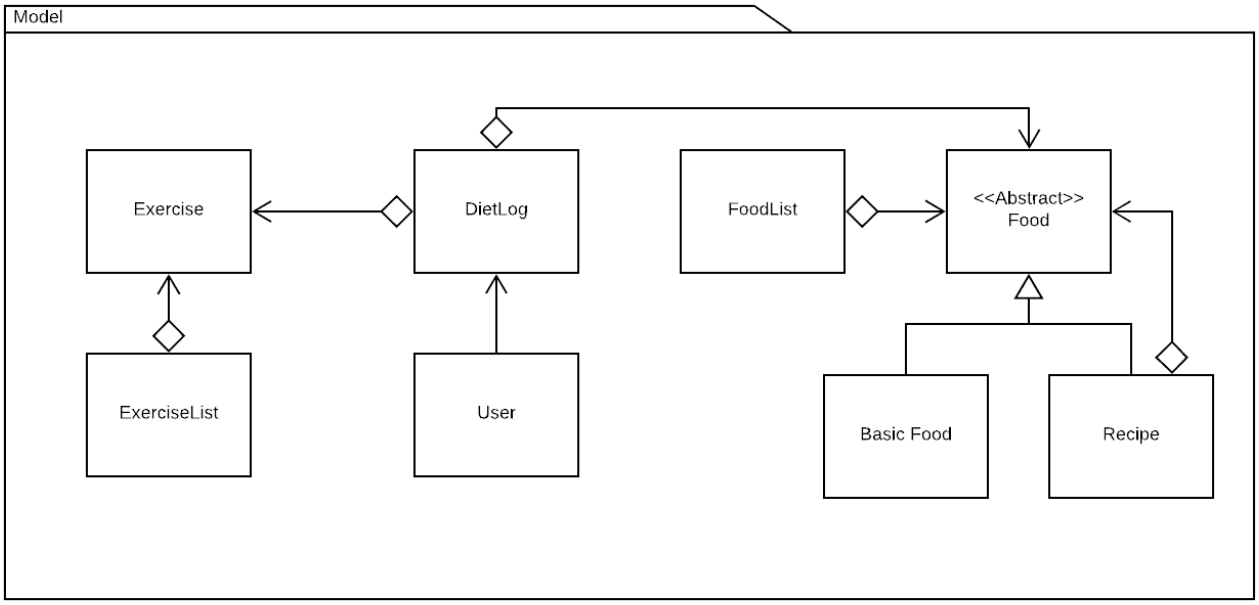
Class DietLog	
Responsibilities	- Store a daily log of food consumed in ArrayList - Store a daily log of exercises performed in ArrayList - Contains calculation functions to be able to provide the user nutritional information from the log of that day. Uses calories consumed and calories burned to get the total calories left for the day and the net calories. - Ability to log a food item that the user has specified. - Ability to delete a food item that the user has specified. - Ability to log an exercise item that the user has specified. - Ability to delete an exercise 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. **Model.Exercise** - Contains an arraylist of Exercise 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 - ability to update a specific DietLog's weight and caloric limit
Collaborators (uses)	**Model.DietLog -** contains the daily log history of food consumed by the user, and the daily log history of exercises performed by the user.

Class ExerciseList	
Responsibilities	- stores a HashMap of Exercise objects as available exercises - ability to return a HashMap 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 HashMap in ExerciseList will be populated with Exercise objects

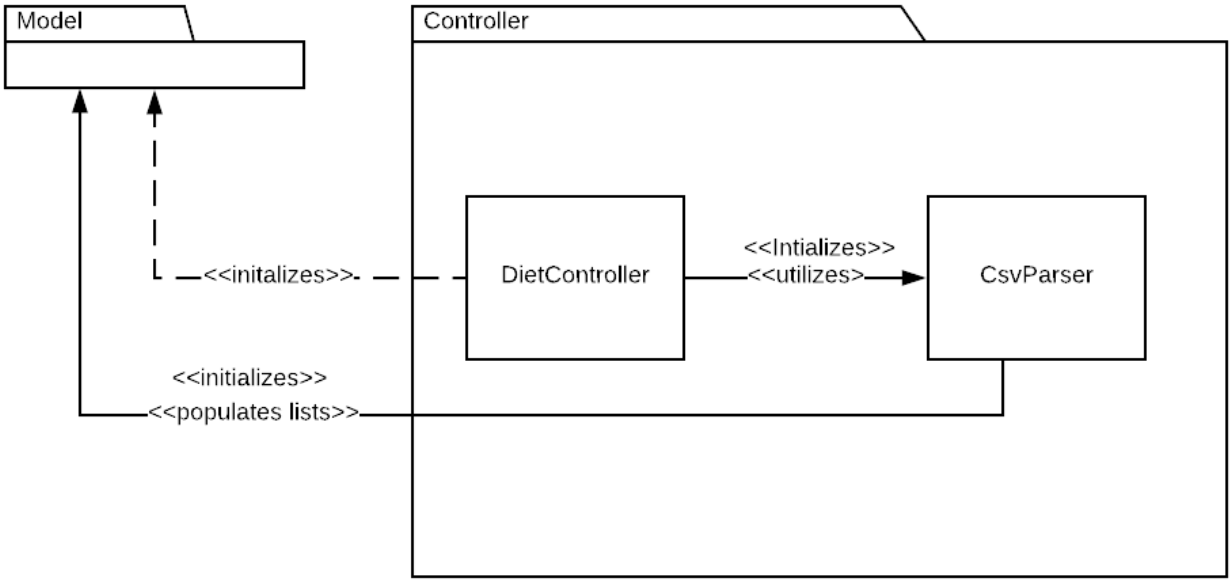
Class Exercise	
Responsibilities	- contains name, the amount of calories burned per hour, and the duration of the exercise - functions to get/set all attributes - Ability to build its own csv format and returns it - Has the ability to copy itself
Collaborators (uses)	(Doesn't need to collaborate with 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.ExerciseList** - Used to access and modify available exercises **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 - Saves FoodList, DietLog, and ExerciseList on exiting the program
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.BasicFood** - needs to create basic food objects to populate **Model.Recipe -** needs to creare recipe food objects to populate **Model.DietLog** - gets populated based on csv data **Model.ExerciseList** - gets populated based on csv data **Model.Exercise** - needs to create basic exercise objects to populate **Model.User** - gets user's log history to populate



View Subsystem

Class DietView	
Responsibilities	- displays the data to the user in the console - initializes DietController - initializes ViewMediator - initializes DietManagerFrame
Collaborators (uses)	**Controller.DietController** - used to connect user interaction to application logic **Model.Food** - contains the food data that is needed for the view to display the proper information **Model.Exercise** - contains the exercise data that is needed for the view to display the proper information **Model.DietLog** - needs a reference to DietLog in order to get the specified DietLog object that the user requested **java.util.Scanner** - this is being used only for the CLI implementation for testing logic

Class ViewMediator	
Responsibilities	- handles interaction between different UI elements
Collaborators (uses)	Handles UI updating and other interaction the following classes: **View.ChartsPanel** **View.DailyValuesPanel** **View.DietManagerFrame** **View.DietManagerTabbedPane** **View.ExercisePanel** **View.FoodPanel** **View.LogPanel** **View.UserPanel** **View.BarGraph**

Class DietManagerFrame	
Responsibilities	- creates actual window of the program - responsible for initializing all panels in the GUI and adding them to the main frame - has ability to repaint itself - ability to save program data and close the program when window is closed
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object **View.ViewMediator** - passed in to DietManagerFrame so all components can communicate without having a direct relation with each to reduce coupling of the system **View.DietManagerTabbedPane -** initializes and adds this pane to the frame **View.DailyValuesPanel -** initializes and adds this pane to the frame **View.ChartsPanel -** initializes and adds this pane to the frame **javax.swing.JPanel** - used to create JPanels needed for the frame **java.awt.event.WindowAdapter** - used to trigger an event when window is being closed **java.awt.event.WindowEvent** - used as a parameter in the window closing event

Class UserPanel	
Responsibilities	- initializes and builds nameEntryPanel panel - initializes and builds weightEntryPanel panel - initializes and builds dailyGoals panel - ability to set the user's name upon clicking the submit button next to the user's name input field - ability to set the user's weight upon clicking the submit button next to the user's weight input field - ability to set the user's goals with JButton called goalsButton - ability to save the program data when the save button is clicked
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object **View.ViewMediator** - passed in so all components can communicate without having a direct relation with each to reduce coupling of the system **javax.swing.JPanel -** UserPanel inherits from JPanel **javax.awt.event.ActionEvent** - used as the parameter to pass when overriding the actionPerformed function **javax.awt.event.ActionListener** - used for the ability to attach action listeners onto their respectful buttons

Class ChartsPanel	
Responsibilities	- responsible for rendering the panel that will hold the BarGraph chart - responsible for initializing a new BarGraph object with the correct corresponding information and adding that BarGraph object to the panel - ability to update the panel with the correct information being displayed in the BarGraph object.
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object. **View.ViewMediator** - passed in so all components can communicate without having a direct relation with each to reduce coupling of the system. **View.BarGraph** - used for initialization of BarGraph object **javax.swing.JPanel** - ChartsPanel inherits from JPanel **javax.swing.TitledBorder** - used to set the title on the border

Class DailyValuesPanel	
	- responsible for rendering the panel that holds the user information for the current date - ability to change the current date that the user is viewing and will update the information based on that date - shows calories consumed, calories expended, and net calories as well as user's name, and weight. - displays the exercises and foods that have been logged for a specific date - uses Mediator to update the ChartPanel and this panel's information - ability to create a JScrollPane for both logged foods and logged exercises
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object. **View.ViewMediator** - passed in so all components can communicate without having a direct relation with each to reduce coupling of the system. **javax.swing.JPanel** - DailyValuesPanel inherits from JPanel **javax.swing.border.TitledBorder** - used to set the title on the border **java.awt.event.ActionEvent** - used as the parameter to pass when overriding the actionPerformed function **java.awt.event.ActionListener** - used for the ability to attach action listeners onto their respectful buttons **javax.swing.JButton** - used to create buttons for the GUI and add them to their respectful components **javax.swing.JTextField** - used to create text fields for the GUI and add them to their respectful components

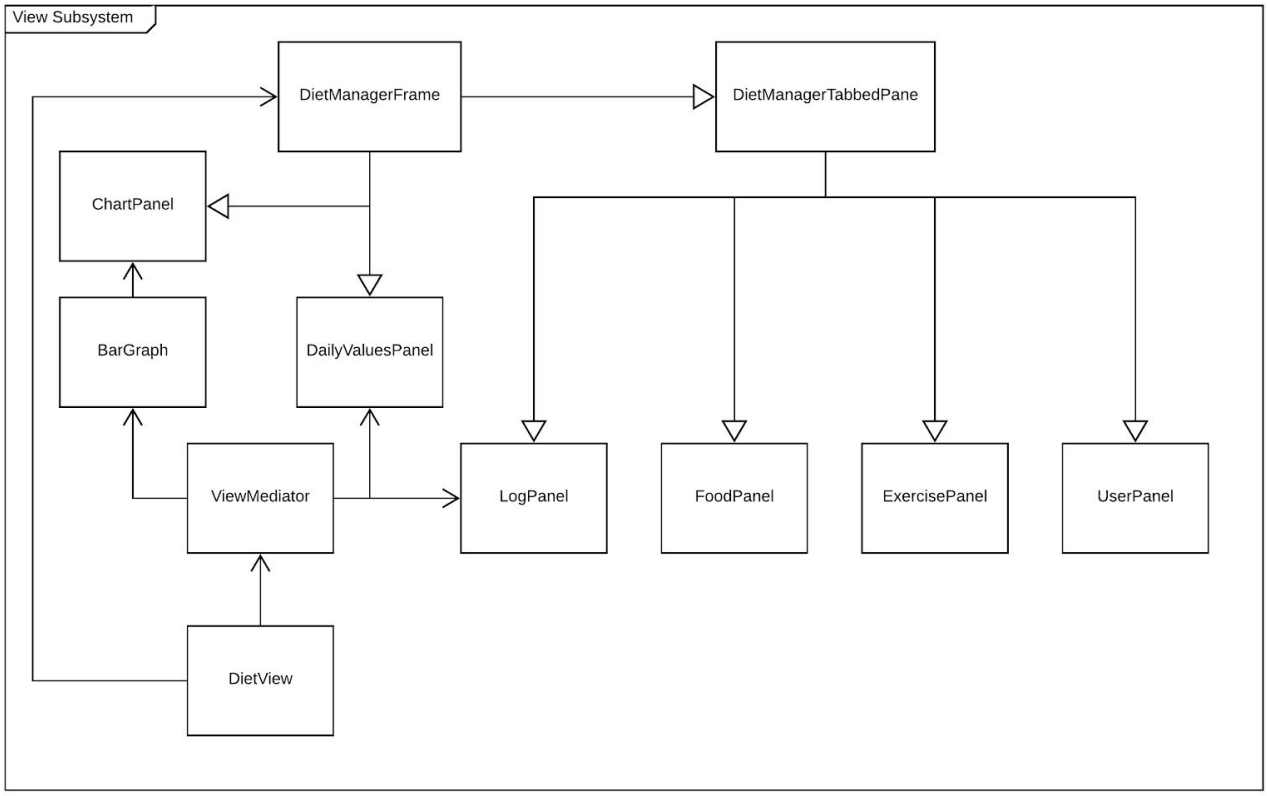
Class DietManagerTabbedPane	
Responsibilities	- responsible for initializing the UserPanel, ExercisePanel, FoodPanel, and LogPanel - holds the user, food, exercise, and log panels
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object. **View.ViewMediator** - passed in so all components can communicate without having a direct relation with each to reduce coupling of the system. **javax.swing.JTabbedPane** - this class inherits from JTabbedPane

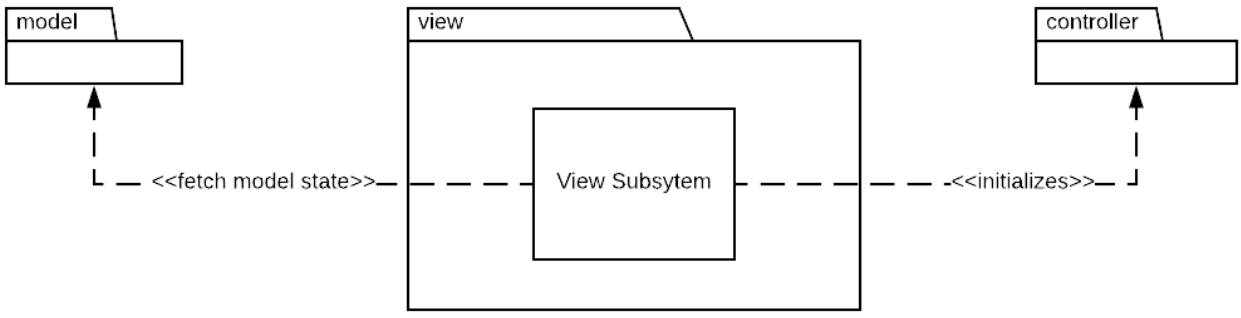
Class ExercisePanel	
Responsibilities	- responsible for getting user input to add an exercise to the list of exercises - responsible for getting the name of the exercise to be added - responsible for getting the calories burned from an exercise - uses the ViewMediator to update the information in the LogPanel - ability to reset both text input fields once the button to add an exercise has been clicked
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object. **View.ViewMediator** - passed in so all components can communicate without having a direct relation with each to reduce coupling of the system. **javax.swing.JPanel** - ExercisePanel inherits from JPanel **javax.swing.JTextField** - used to create textfields for the GUI and add them to their respectful components **javax.swing.JLabel** - used to create labels for the GUI and add them to their respectful components **javax.swing.JButton** - used to create buttons for the GUI and add them to their respectful components **java.awt.event.ActionEvent** - used as the parameter to pass when overriding the actionPerformed function **java.awt.event.ActionListener** - used for the ability to attach action listeners onto their respectful buttons

Class FoodPanel	
Responsibilities	- gets the user input to add a new basic food - gets the user input to add a new recipe - displays a list of foods that can be added to a recipe - ability to change the number of servings of a basic food to a recipe - ability to add more than one basic food to a recipe
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object. **View.ViewMediator** - passed in so all components can communicate without having a direct relation with each to reduce coupling of the system. **javax.swing.JPanel** - FoodPanel inherits from JPanel **javax.swing.JTextField** - used to create text fields for the GUI and add them to their respectful components **java.util.Set** - used to keep reference to a list of food information that is needed to populate the combo box of available foods **java.awt.event.ActionEvent** - used as the parameter to pass when overriding the actionPerformed function **java.awt.event.ActionListener** - used for the ability to attach action listeners onto their respectful buttons

Class LogPanel	
Responsibilities	- displays a list of foods for the user to log for their day - allows a user to log the number of servings of food they consumed - displays a list of exercises the user can log for the day - allows a user to log an exercise to their daily log
Collaborators (uses)	**Controller.DietController** - passed in an instance of DietController to make sure all components are using the same DietController object. **View.ViewMediator** - passed in so all components can communicate without having a direct relation with each to reduce coupling of the system. **javax.swing.JPanel** - LogPanel inherits from JPanel **javax.swing.JTextField** - used to create text fields for the GUI and add them to their respectful components **java.awt.event.ActionEvent** - used as the parameter to pass when overriding the actionPerformed function **java.awt.event.ActionListener** - used for the ability to attach action listeners onto their respectful buttons **java.util.Set** - used to keep reference to a list of food and exercise information that is needed to populate the food log panel and the exercise log panel

Class BarGraph	
Responsibilities	- displays a graph of fat, carbs, and proteins in percent format - updates graph as food items are added to the log
Collaborators (uses)	**Model.DietLog** - pulls percentages of fat, carbs, and protein from the current DietLog and updates the BarGraph **javax.swing.JPanel -** BarGraph inherits from JPanel

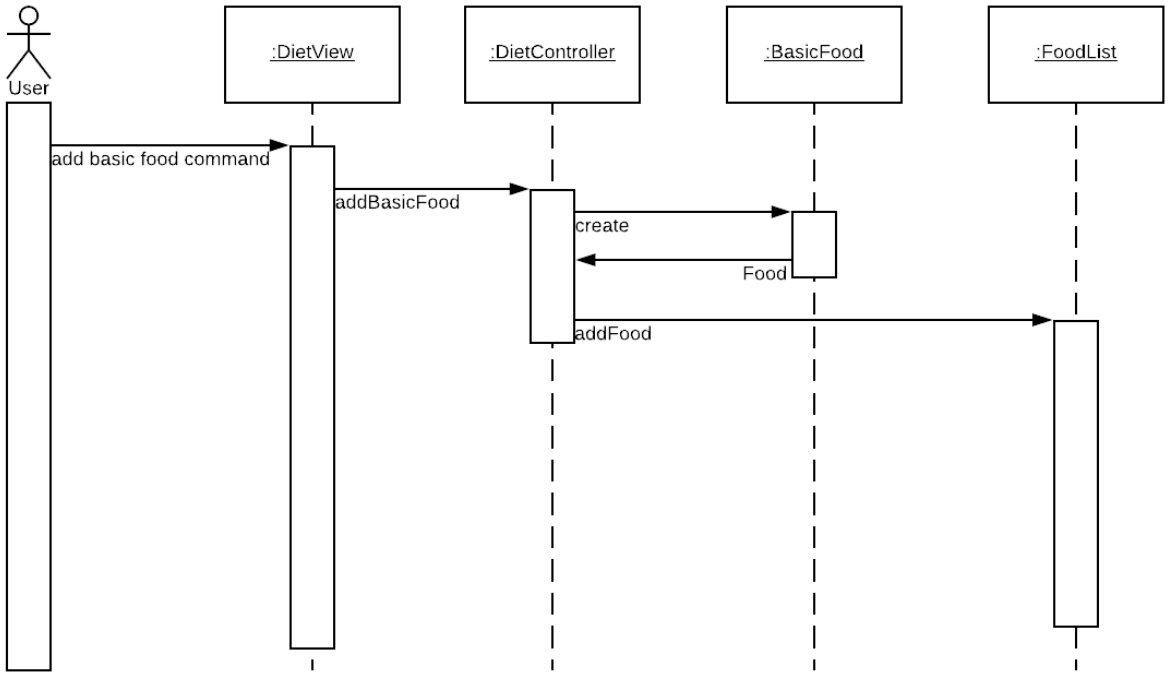




Sequence Diagrams

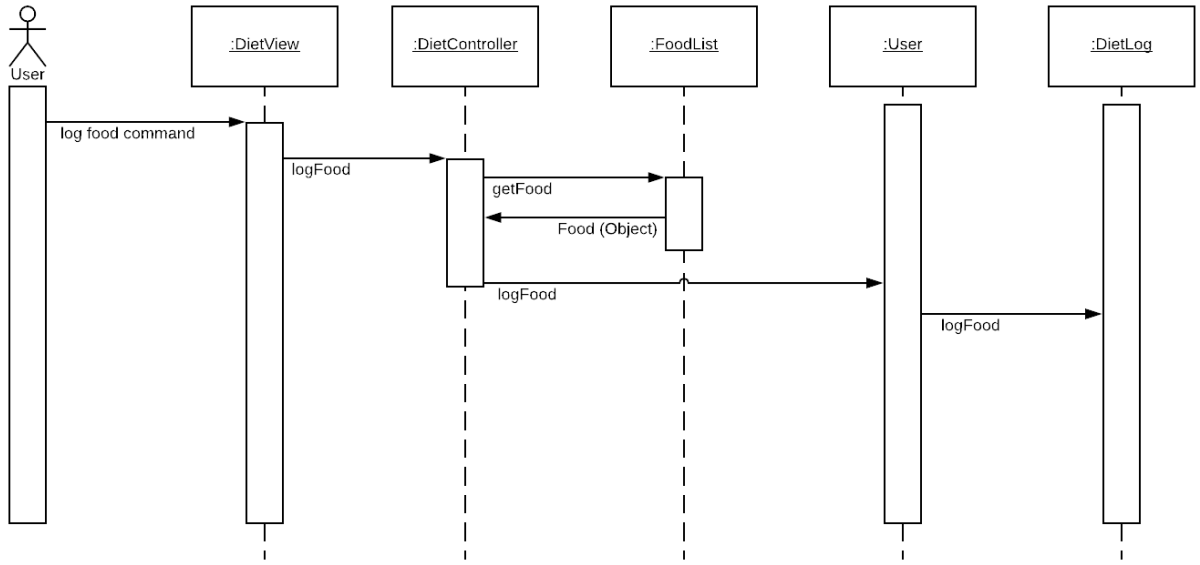
Sequence 1 - Add Food to Available Foods

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



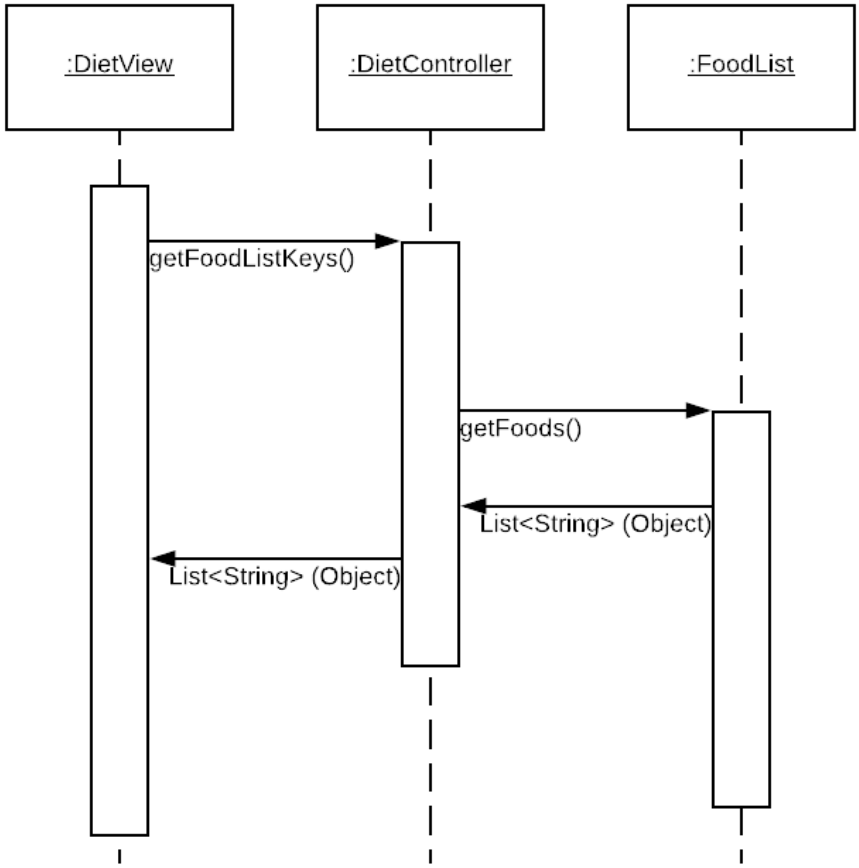
Sequence 2 - Log Food Consumed

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

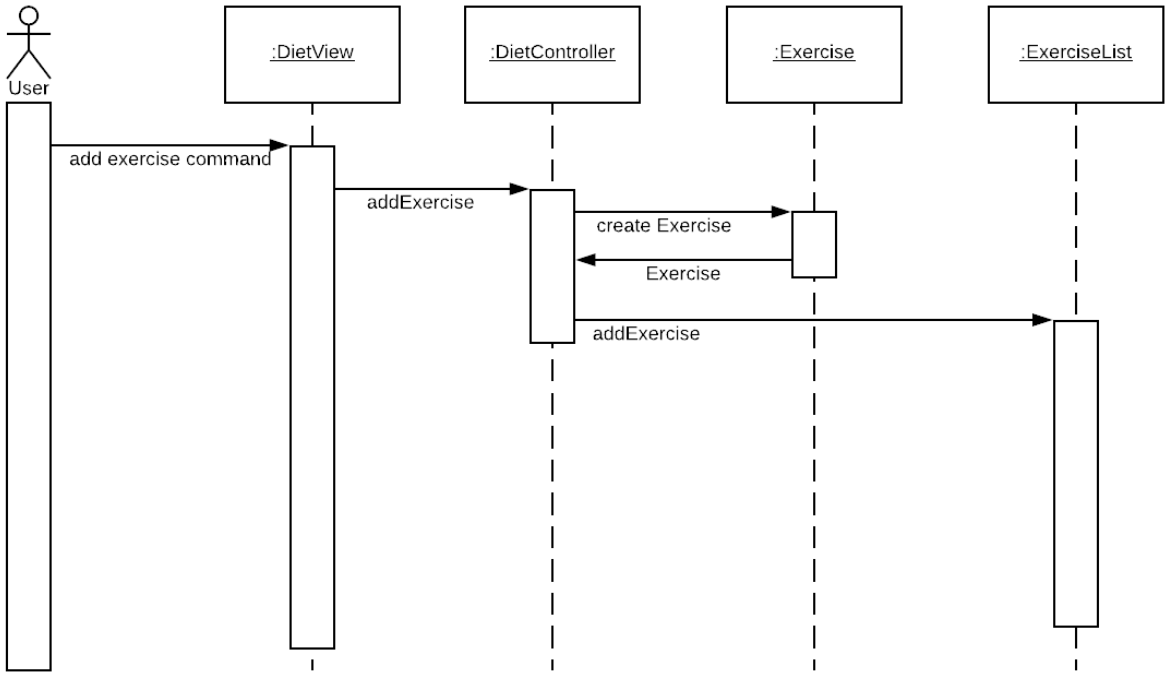


Sequence 3 - Display List of Available Foods

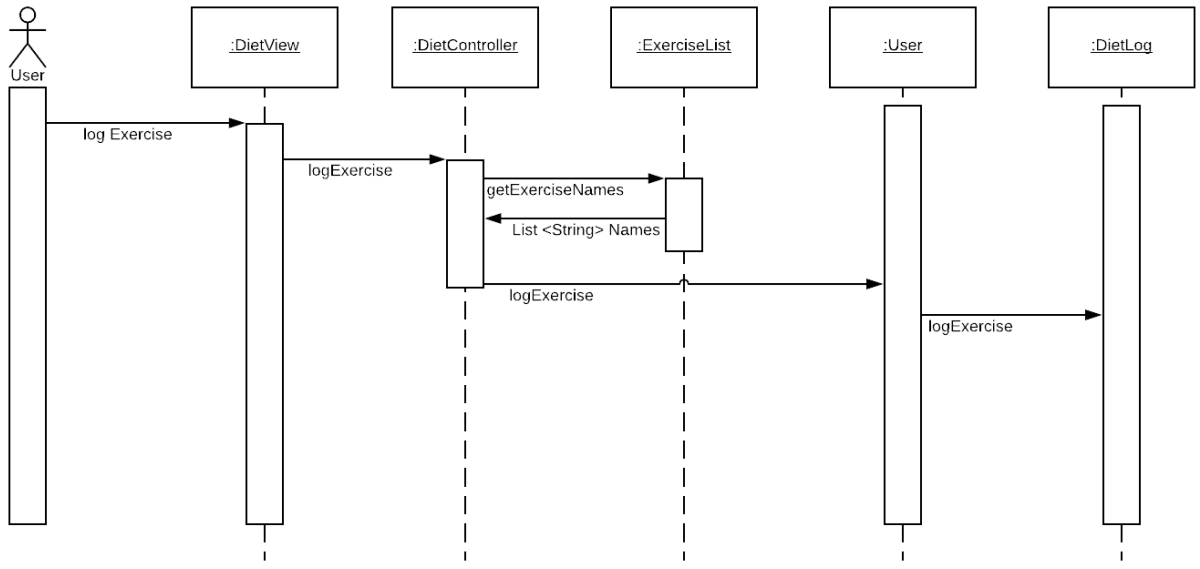
The view displays available foods on several circumstances.



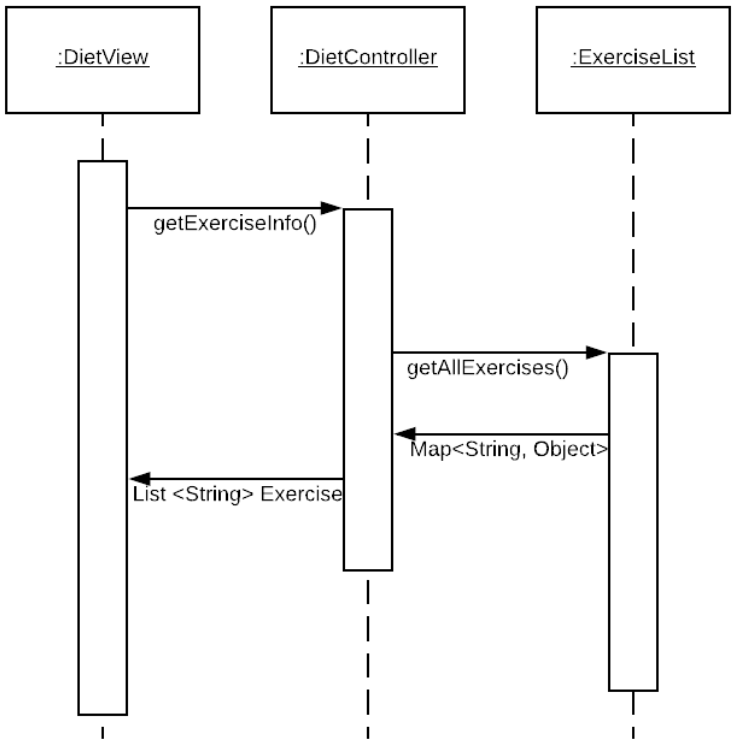
Sequence Diagram #4 - User Adds an Exercise



Sequence Diagram #5 - Log an Exercise



Sequence Diagram #6 - Show list of Exercises



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

Pattern #3 Mediator Pattern

Mediator Pattern	
Mediator	ViewMediator
Mediated Classes	ChartsPanel DailyValuesPanel DietManagerFrame DietManagerTabbedPane ExercisePanel FoodPanel LogPanel UserPanel