Diet Manager, Version 2.0

Project Design Document

SWEN - 383-01, Los Niños Sanos

Tim Endersby <tje7782@rit.edu>
Anna Jacobsen <aj4693@rit.edu>
John Hill <jxh6494@rit.edu>
Abbey Sands <als6301@rit.edu>
Xuting Zhang <xxz9708@rit.edu>

Project Summary

The goal of this project is to create a program capable of assisting users with maintaining their diets and tracking their exercise regimens. The program will contain a collection of basic foods and recipes. (Recipes being comprised of basic foods elements and other recipes). Users will be able to add new basic foods and recipes to the program, and the application will store them. Likewise, users will have access to a collection of exercises which may be selected from or expanded.

The daily log feature will provide users with the ability to track the various food items they consume throughout the day. The program will generate textual totals for the nutritional information (i.e. calories, grams of fat, carbs, and protein) and a graphical representation of this data upon request from the user.

Users will also be able to track their exercises by adding them to the log along with the calories burned during the activity. The program shall be able to calculate and display the information regarding the caloric goal, total daily caloric intake, total caloric expenditure, the difference between the caloric goal and caloric intake, and the net caloric intake.

In addition to the food and exercise tracking aspect of the program, individuals will be able to save personal data such as their weight. The program will facilitate the setting of personal health goals, such as caloric intake limits, and inform users of whether or not they are successfully meeting these benchmarks.

Design Overview

The second iterative design for Diet Manager was created over multiple meetings using a whiteboard and Lucidchart. Because the first phase design was already in place, we thought that the design for the second phase would be less time consuming. This was not so, as we made a fair amount of adjustments to the original design. The team worked together to draw out the UML on a whiteboard in a private team meeting room. An attempt was made to discuss each class and its purpose, and this led to a couple of classes being dropped, such as SubRecipe. Once again, an attempt was made to best separate concerns, but the team was a bit confused with the view and controller interaction.

It was thought that we would like to implement the Command pattern into our project, but ran out of time to try and do so before the due date of the first draft of the second phase design document. We may attempt to do this before the second draft, as we believe it would be beneficial for our program's extensibility.

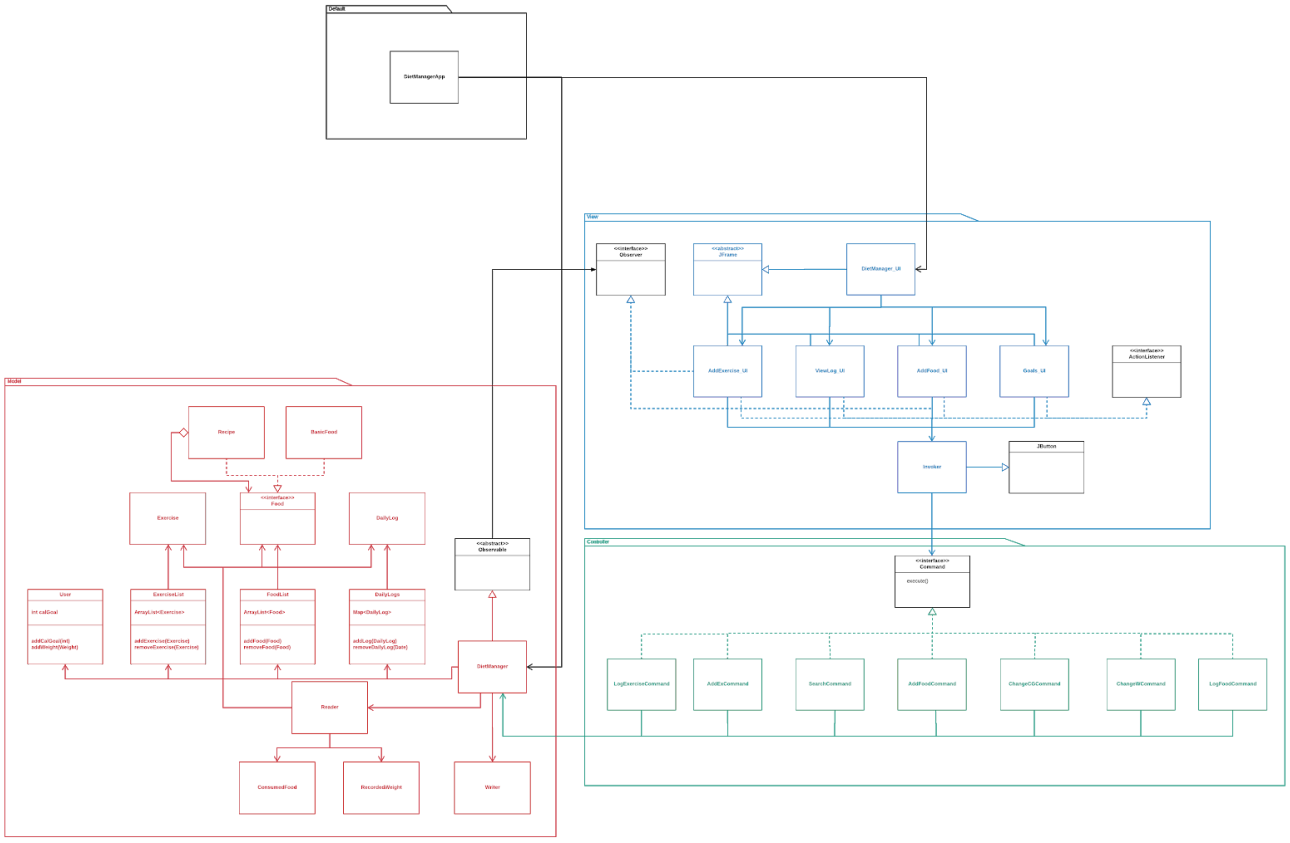
One large change was executed on the view package. Instead of just one UI class, our program now contains five. This allows for different screens to be focused on one particular task, granting high cohesion from a user experience standpoint. We also have started to map out how the wiring between the Controller and View classes will work, though, as aforementioned, we believe this may be improved upon. Additionally, we added the Exercise and ExerciseList classes in order to help us achieve the new goals delineated in the phase two requirements.

As the UML was only finished slightly before the deadline, dynamic modeling was attempted but not finished.

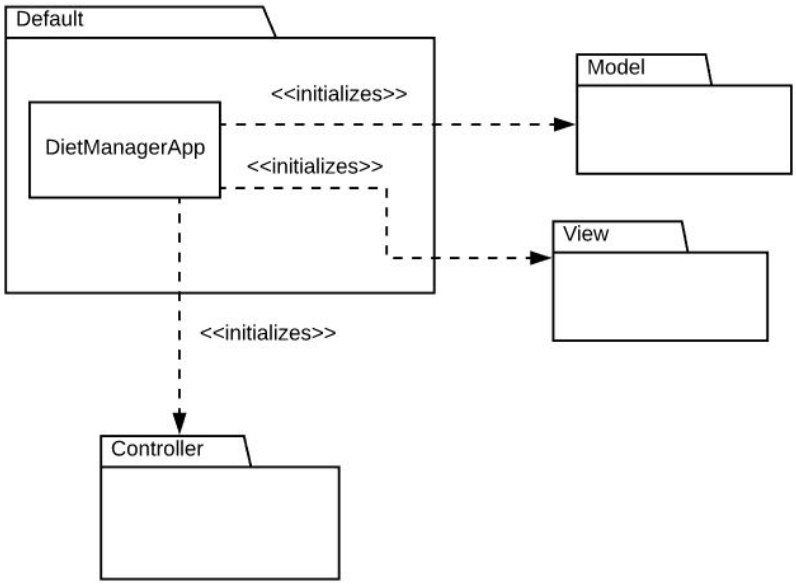
Some ideas that were deemed unfit for inclusion in the design were the following:

- SubRecipe. SubRecipe was removed when the team determined that it did not serve any special purpose that Recipe did not already fill. Rather, we decided to look more into redesigning the Recipe class to be more flexible.
- Collections classes were added to the model, such as the DailyLogs. DailyLogs contains a HashMap of DailyLog objects and methods for retrieving information from this data structure and removing it. We decided to use a HashMap because HashMaps allow for empty values, which means if a user doesn't make an entry for a particular day, that is OK. Also, key and value pairs will make the search function easier to execute.
- We realized that in the MVC pattern the manipulation of data should only occur in the controller. The model is simply for holding data and "modeling" objects.

Subsystems

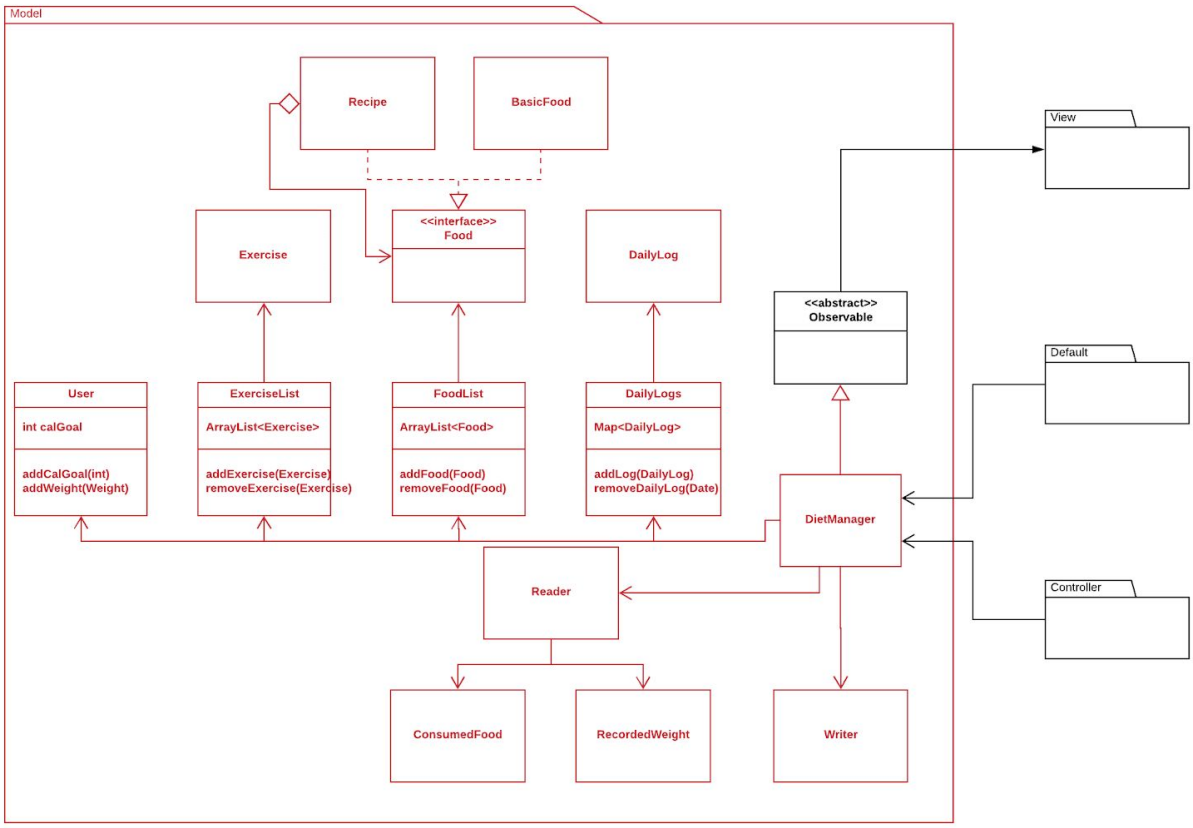

Default
DietManagerApp
View
<<interface>>
Observer
<<abstract>>
JFrame
DietManager_UI
AddExercise_UI
ViewLog_UI
AddFood_UI
Goals_UI
<<interface>>
ActionListener
Invoker
JButton
Model
Recipe
BasicFood
Exercise
<<interface>>
Food
DailyLog
<<abstract>>
Observable
User
int calGoal
ExercisesList
ArrayList<Exercise>
FoodList
ArrayList<Food>
DailyLogs
Map<Date|Log>
DietManager
Reader
ConsumedFood
RecordedWeight
Writer
Controller
<<interface>>
Command
execute()
LogExerciseCommand
AddExCommand
SearchCommand
AddFoodCommand
ChangeGGCommand
ChangeWCommand
LogFoodCommand

Subsystem: Default



Class DietProgramApp	
Responsibilities	Holds the main method. Constructs the main UI, DietManager_UI Constructs all the sub UI's (AddFood_UI, AddExercise_UI, ViewLog_UI) Constructs the DietManager, which is the main Model "Meat and Potatoes" class Constructs all the Commands (AddFood, AddExercise, Search) and injects the DietManager object into all of them
Collab. (uses)	**Model.DietManager** - the primary Model class **View.DietManager_UI -** the primary View class **View.AddFood_UI -** the sub-UI for adding a Food **View.AddExercise_UI -** the sub-UI for adding an Exercise **View.ViewLog_UI -** the sub-UI for viewing all the DailyLog information **Controller.Command** - the command interface **Controller.AddFoodCommand** - the command class that allows a user to add a Food **Controller.AddExerciseCommand** - the command class that allows a user to add an Exercise **Controller.SearchCommand** - the command class that allows the user to search through the DailyLog by Date

Subsystem: Model



Class DietManager	
Responsibilities	Notify observers (UI) of changes to the food collection Main “meat and potatoes” class, it is the driver of the whole program. The internal data structures reside in this class A Reader object is initialized and the corresponding methods are called to read in the three provided csv files (exercise.csv, log.csv and recipefoods.csv)
Collaborators (extends)	**java.util.Observable -** to allow this class to be observed by others
Collaborators (uses)	**java.util** - so that this class can use the ArrayList and HashMap data structures and also about the Date class **Model.DailyLogs** - allows a DailyLogs object to be stored in this class after reading **Model.FoodList** - allows a FoodList to be stored in this class after reading **Model.ExerciseList** - allows an ExerciseList to be stored in this class after reading **Model.Food** - allows the storage of an ArrayList of Foods

	Model.Exercise - allows the storage of an ArrayList of Exercises **Model.DailyLog** - allows the storage of a HashMap of Date and DailyLog objects **Model.Reader** - instantiates a Reader object to read in the initial data from the CSV files

Class DailyLogs	
Responsibilities	Holds a HashMap of Date and DailyLog objects Allows for additions and subtractions to the HashMap Also has an accessor for the HashMap
Collaborators (uses)	**Model.DailyLog -** so that it knows what a DailyLog object is and can store a DailyLog object as part of its HashMap **java.util.HashMap** - to use the HashMap data structure

Class DailyLog	
Responsibilities	Defines a DailyLog object
Collaborators (uses)	**java.util.ArrayList -** to use the ArrayList data structure **java.util.ConsumedFood** - to be able to hold an ArrayList of Consumed Food objects **java.util.RecordedWeight** - to be able to hold an ArrayList of RecordedWeight objects **java.util.RecordedExercise** - to be able to hold an ArrayList of RecordedExercise objects

Class FoodList	
Responsibilities	Holds an ArrayList of Food objects Can add and remove foods from the ArrayList Also has an accessor that returns the ArrayList of Food objects
Collaborators (uses)	**Model.Food -** so that it knows what a Food object is **java.util.ArrayList** - to use the ArrayList data structure

Interface Food	
Responsibilities	Food is an interface for all the food items (Recipe and Basic). It has a getTotal() method that returns the total calorie count for a food. It comprises the component of the composite pattern.
Collaborators (uses)	**None**

Class Basic	
Responsibilities	Basic is a class for basic foods. It has a Calorie, Fat, Carbs and Protein attributes. It has a getTotals() method that returns an array of totals for all the nutritional values discussed previously. Basic is the leaf in the composite pattern.
Collaborators (implements)	**Model.Food -** to use the Food interface as part of the Composite pattern

Class Recipe	
Responsibilities	Recipe is a collection of basic foods and sub recipes. Recipe has a serving size, and a getTotal() method which returns the total calories, carbs, fats, and protein, recursively calculated including all the basic foods and sub recipes. Recipe represents the composite in the composite pattern.
Collaborators (implements)	**Model.Food -** to use the Food interface as part of the Composite pattern

Class Exercise	
Responsibilities	Defines an Exercise Object, which consists of an Exercise name and the number of calories expended by the exercise in 1 hour for a 100 pound person.
Collaborators (uses)	**java.util.ArrayList** - to use the ArrayList data structure

Class ExerciseList	
Responsibilities	Holds an ArrayList of Exercise objects. Allows the addition and removal of Exercise objects from the ArrayList.
Collaborators (uses)	**Model.Exercise -** so that it knows what an Exercise object is **java.util.ArrayList** - to use the ArrayList data structure

Class User	
Responsibilities	Stores a user's weight and goals. Set/change the weight and goal.
Collaborators (uses)	None

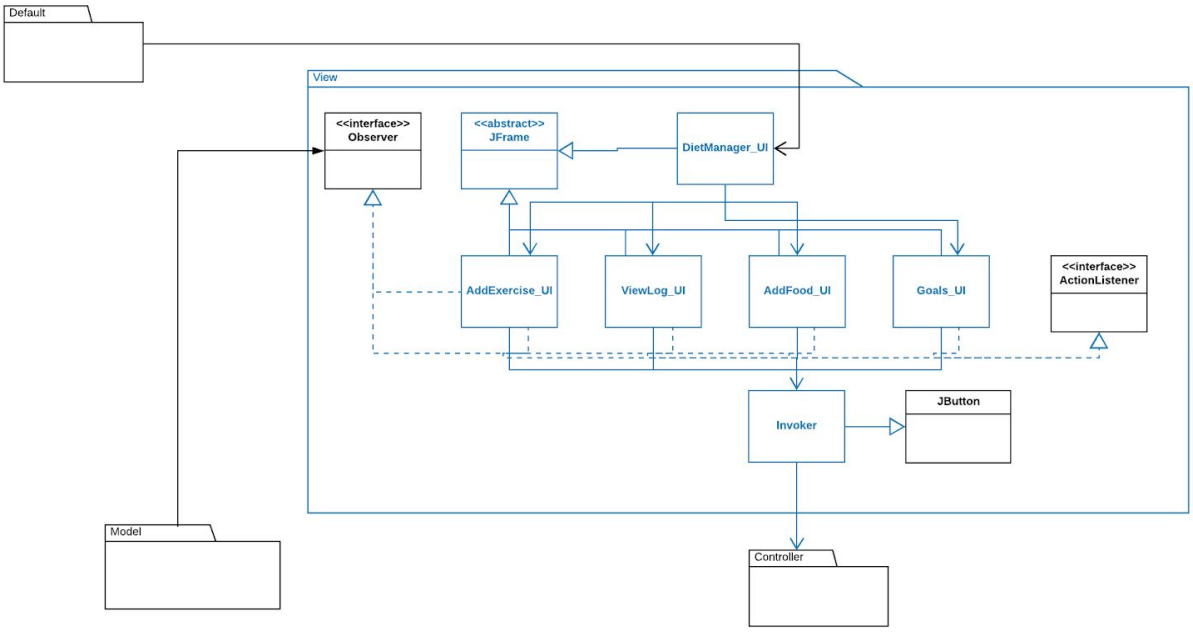
Class Writer	
Responsibilities	Provides the means to writing to all CSV files (basicfoods.csv, log.csv, exercise.csv)
Collab. (uses)	**java.io -** so that it can use the PrintWriter and File classes **java.net -** to use the URL class **java.util** - to use the Calendar class

Class Reader	
Responsibilities	Provides the means to reading in all CSV files (basicfoods.csv, log.csv, exercise.csv) The readFoods method creates Basic and Recipe objects and adds them to the FoodList The readLog method creates DailyLog objects for each individual date and adds them to the DailyLogs object The readExercise method creates Exercise objects and adds them to the ExerciseList
Collab. (uses)	**Model.ConsumedFood -** So that it can create an object of ConsumedFood **Model.RecordedWeight** - So that it can create an object of RecordedWeight **Model.RecordedExercise** - So that it can create an object of RecordedExercise **Model.FoodList** - So that it can store the Basic and Recipe objects in a FoodList object **Model.DailyLog** - So that it can create an instances of DailyLog **Model.DailyLogs** - So that it can store the DailyLog objects in a DailyLogs object **Model.Exercise** - So that it can create instances of Exercise **Model.ExerciseList** - So that it can store the Exercise objects in an ExerciseList object **java.util.Date** - to use the Data class structure **java.io** - to use the BufferedReader class

Class RecordedWeight	
Responsibilities	Defines the structure of a RecordedWeight object. A RecordedWeight object is made up of a Date and a Weight (as a double)
Collab. (uses)	**java.util.Date** - to use the Data class structure

Class ConsumedFood	
Responsibilities	Defines the structure of a ConsumedFood object. A ConsumedFood object is made up of a Date, the Food name, and the Servings consumed.
Collab. (uses)	**java.util.Date** - to use the Data class structure

Subsystem: View



Interface Observer	
Responsibilities	Provides a way for Diet_UI to watch DietManager When DietManagerchanges, Diet_UI will automatically update its graphics.
Collaborators	**View.DietManager_UI -** The class that actually acts as the observer.

Class DietManager_UI	
Responsibilities	Provide the user with an interface for interacting with program features, such as updating weight, adding food and adding exercises.
Collaborators (uses)	**javax.swing -** Used to create the visual elements, **Exercise_UI -** Can collaborate with the Exercise Frame, **ViewLog_UI -** Can collaborate with the Log Frame, **AddFood_UI -** Can collaborate with the Food Frame, **Goals_UI -** Can collaborate with the Goals Frame **BarGraph -** can collaborate with the graphic Frame **java.util.Observable -** So that it can observe changes from DietManager
Collaborators (implements)	**java.util.Observer -** Implements this class so that it an observe changes from the model **java.awt.event.ActionListener -** Implements the ActionListener interface so that it can capture the action performed

Class AddExercise_UI	
Responsibilities	Provides the user interface for adding a new exercise
Collaborators	**javax.swing -** So that the class can utilize visual elements, **Controller.AddExerciseCommand -** allows the user to adding an exercise to the csv file

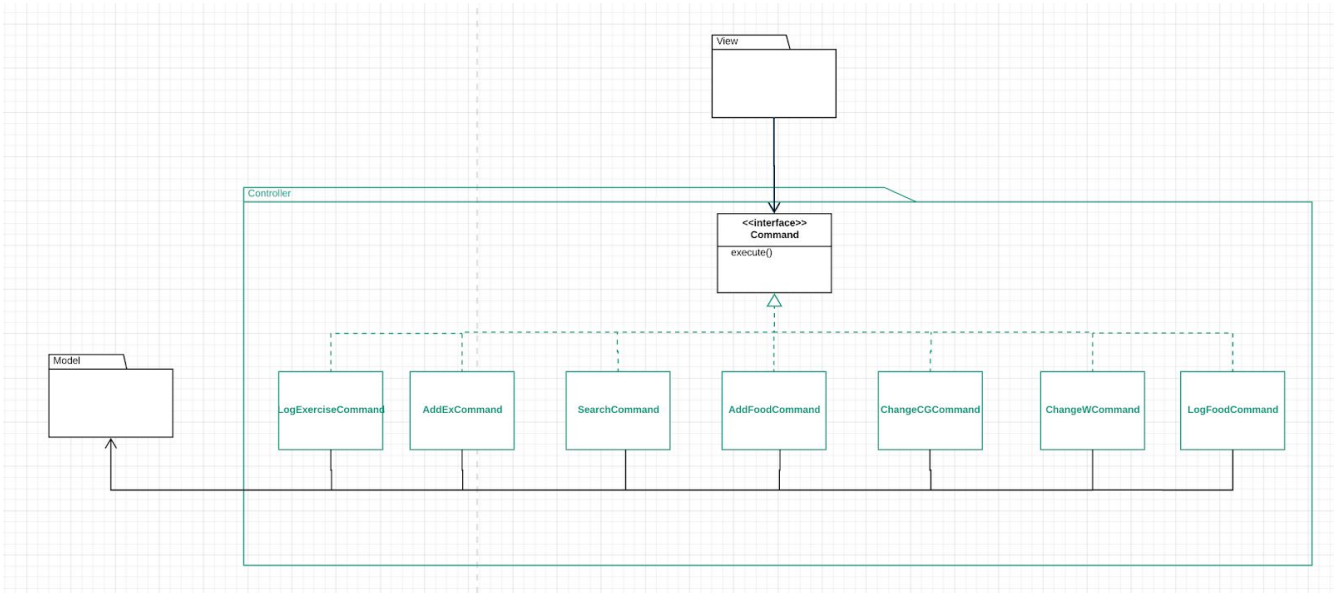
Class ViewLog_UI	
Responsibilities	Provides the interface for user to see their current log
Collaborators	**javax.swing** - So that the class can utilize visual elements, **Controller.SearchCommand -**allows a user to find a daily log for a specific date

Class AddFood_UI	
Responsibilities	Contains all of necessary UI elements and functions for the screen related to adding a new food to the program or recording a food to the user's daily log.
Collaborators	**javax.swing -** So that the class can utilize visual elements, **Controller.AddFoodCommand -** The controller for adding food to the csv file

Interface Goals_UI	
Responsibilities	Provides an interface to display and edit calorie and weight goals
Collaborators	**javax.swing** - So that the class can utilize visual elements, **Controller.CalorieGoalCommand -** allows the user to edit their calorie goal

Interface BarGraph	
Responsibilities	Creates a graph to display nutritional information.
Collaborators	**java.awt.Canvas** - so that the class can receive inputs created by user **java.awt.Graphics** - so that the class can draw onto components that can be realized on screen. **java.util.Observable -** So that it can observe changes from DietManager j**ava.util.Observer -** Implements this class so that it an observe changes from the model

Subsystem: Controller



Interface: Command	
Responsibilities	Holds the execute method, which all commands overwrite in order to execute their own specific methods.

Class: Invoker	
Responsibilities	Creates an invoker to be used in triggering a command object.
Collaborators (Uses)	**Controller.Command -** Invoker takes in a Command object so that the correct Command can be called when the Invoker is triggered. **Java.util.Date -** Invoker uses Java.util.Date so that it can use the Date object, which is a parameter in the SearchCommand method setDate(). **Javax.swing.*** - Invoker extends JButton, therefore needing to use the Javax.Swing package. **Java.awt.Font, Color -** Invoker uses this library to style the JButton it creates. **Java.awt.event.* -** Invoker uses this library to trigger the commands when it is clicked.

Class: AddExerciseCommand	
Responsibilities	Creates a new Exercise object, alerting the model and view of this change to keep the log as updated as possible.

Collaborators (Implements)	**Command -** An interface that holds the execute method, which all commands override when they are invoked.
Collaborators (Uses)	**Model.Writer -** AddExerciseCommand uses a writer to write the newly created exercise to the daily log. **Model.DietManager -** AddExerciseCommand takes in a DietManager and uses its readExAndUpdate() method to read in the newly written exercise and display it. **Model.Exercise -** AddExerciseCommand creates a new exercise object to be written to the daily log.

Class: SearchCommand	
Responsibilities	Searches through the daily log for a specific date, then filters the daily log to only show the results for that date.
Collaborators (Implements)	**Command -** An interface that holds the execute method, which all commands override when they are invoked.
Collaborators (Uses)	**Model.DietManager -** SearchCommand takes in a DietManager object, and then calls its searchLogsAndUpdate() method to filter the daily logs by a specifically entered date. **Java.util.* -** Search Command uses Java.util.* so that it's setDate() method can take in a date object to later search for.

Class: AddFoodCommand	
Responsibilities	Creates a new BasicFood object that is then added to and displayed in the daily log.
Collaborators (Implements)	**Command -** An interface that holds the execute method, which all commands override when they are invoked.
Collaborators (Uses)	**Model.Basic -** AddFoodCommand creates a BasicFood object to be added to the daily log. **Model.Writer -** AddFoodCommand creates a Writer object and uses its writeFood() method to add the newly created food to the log csv file. **Model.DietManager -** AddFoodCommand takes in a DietManager object and uses its readFoodsAndUpdate() method to correctly update and display the newly created food in the daily log.

Class: CalorieGoalCommand

Responsibilities	A command to update the user's calorie goal for the day.
Collaborators (Implements)	**Command -** An interface that holds the execute method, which all commands override when they are invoked.
Collaborators (Uses)	**Model.DailyCalorieLimit -** CalorieGoalCommand creates a new DailyCalorieLimit object which becomes the users new calorie goal for the day

Class: AddWeightCommand	
Responsibilities	A command to update the user's weight for the day.
Collaborators (Implements)	**Command -** An interface that holds the execute method, which all commands override when they are invoked.
Collaborators (Uses)	**Model.RecordedWeight -** AddWeightCommand creates a new RecordedWeight object which becomes the users new recorded weight for the day.

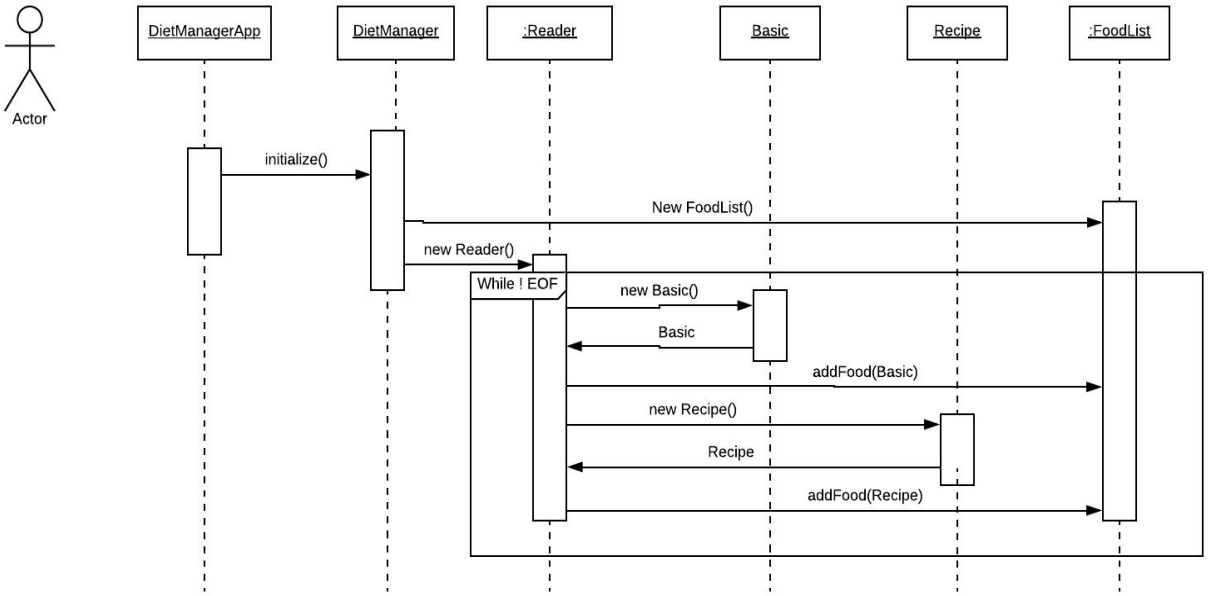
Class: LogFoodCommand	
Responsibilities	Saves a serving of a previous food to the log.
Collaborators (Implements)	**Command -** An interface that holds the execute method, which all commands override when they are invoked.
Collaborators (Uses)	**Model.DietManager -** LogFoodCommand takes in a DietManager object and uses its readFoodsAndUpdate() method to correctly update and display the newly created food in the daily log.

Class: LogExerciseCommand	
Responsibilities	Saves a record of a previous exercise to the log.
Collaborators (Implements)	**Command -** An interface that holds the execute method, which all commands override when they are invoked.
Collaborators (Uses)	**Model.DietManager -** LogExerciseCommand takes in a DietManager object and uses its readExAndUpdate() method to correctly update and display the newly created food in the daily log.

Sequence Diagrams

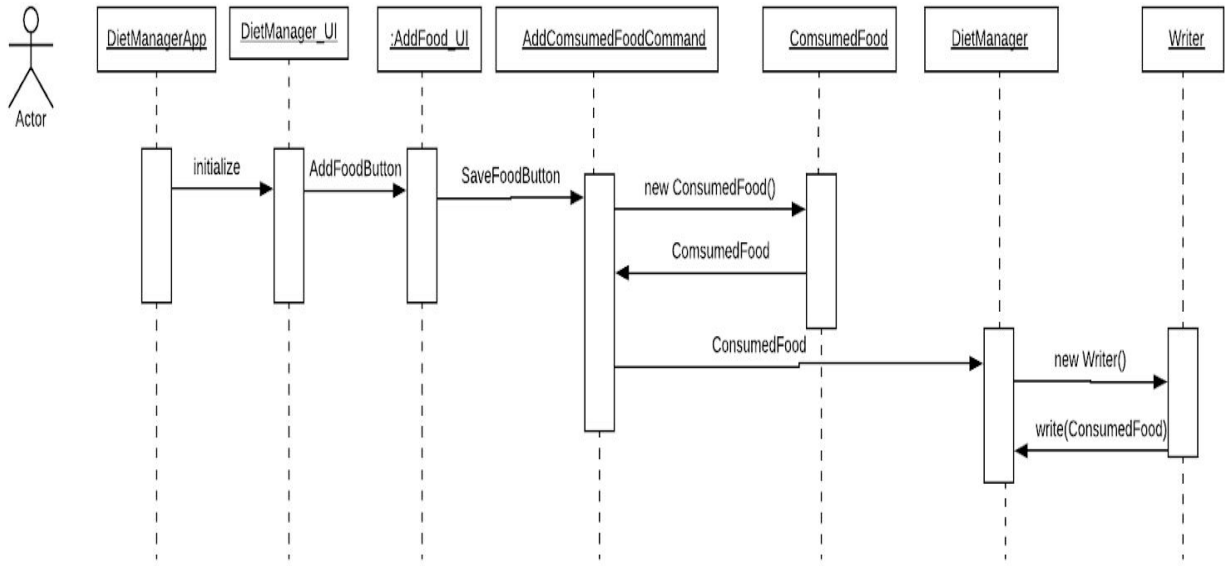
Sequence description 1

Read in a food database consisting of three basic foods, a recipe that contains two of the basic foods, and a recipe that contains the first recipe and the remaining food.



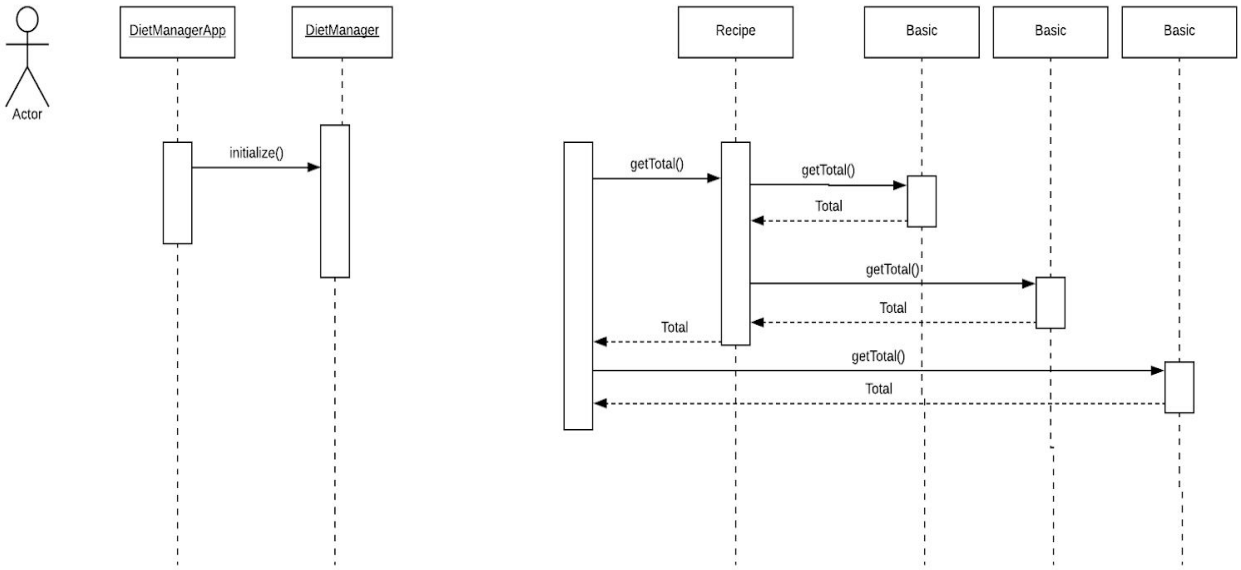
Sequence description 2

Add two servings of a food to the log entry for the current date.


Actor
DietManagerApp
DietManager_UI
:AddFood_UI
AddComsumedFoodCommand
ComsumedFood
DietManager
Writer
initialize
AddFoodButton
SaveFoodButton
new ConsumedFood()
ComsumedFood
ConsumedFood
new Writer()
write(ConsumedFood)

Sequence description 3

Compute the total number of calories for the current date, assuming the log consists of a basic food and a recipe consisting of two basic foods.



Pattern Usage

Observer Pattern

Observer Pattern	
Observer(s)	View.DietManager_UI
Observable(s)	Model.DietManager

Composite Pattern

Composite Pattern	
Composite	Model.Recipe
Component	Model.Food
Leaf	Model.BasicFood

MVC Pattern

MVC Pattern	
Model	DietManager, Writer, Reader, ConsumedFood, RecordedWeight, DailyLogs, DailyLog, FoodList, Food, Recipe, BasicFood, ExerciseList, Exercise, User, java.util.Observable
View	ActionListener, AddExListener, AddFoodListener, ChangeCGListener, ChangeWListener
Controller	java.util.Observable, javax.swing, DietManager_UI, Exercise_UI, ViewLog_UI, AddFood_UI, Goals_UI

Command Pattern

Command Pattern	
Invoker	Controller.Invoker
Command Interface	Controller.Command
Commands	LogExerciseCommand, AddExCommand,SearchCom

	mand, AddFoodCommand, ChangeCGCommand, ChangeWCommand, LogFoodCommand