PROJECT / RELEASE

Project Design Document

TWNN

Lowell Pence <lxp3901@rit.edu>
Jack Old <jxo4940@rit.edu>
Fred Amartey <fda9891@rit.edu>
Fengyi Chen <fxc1494@rit.edu>

Project Summary

This project aims to make the process of logging food consumption and weight changes streamlined. Diet Manager is a software designed to help users meet their diet goals by keeping them up-to-date on exactly they are consuming every day. Users will have features like weight and calorie consumption tracking, adding foods and recipes, and displaying a summary of the nutrition data logged with graphs. This will make diet managing and progression tracking simple for the user.

A food library will allow the user to save commonly consumed foods or complex recipes so that they can easily log them in the future. All user entered data and progress tracking is automatically saved to the user's computer when the program closes, and can be manually saved at any time as well.

Design Overview

To make this streamlined as possible, we decided to implement the model-view-controller design pattern to achieve separation of concerns within our application. This also aims to lower coupling and increase cohesion as well.

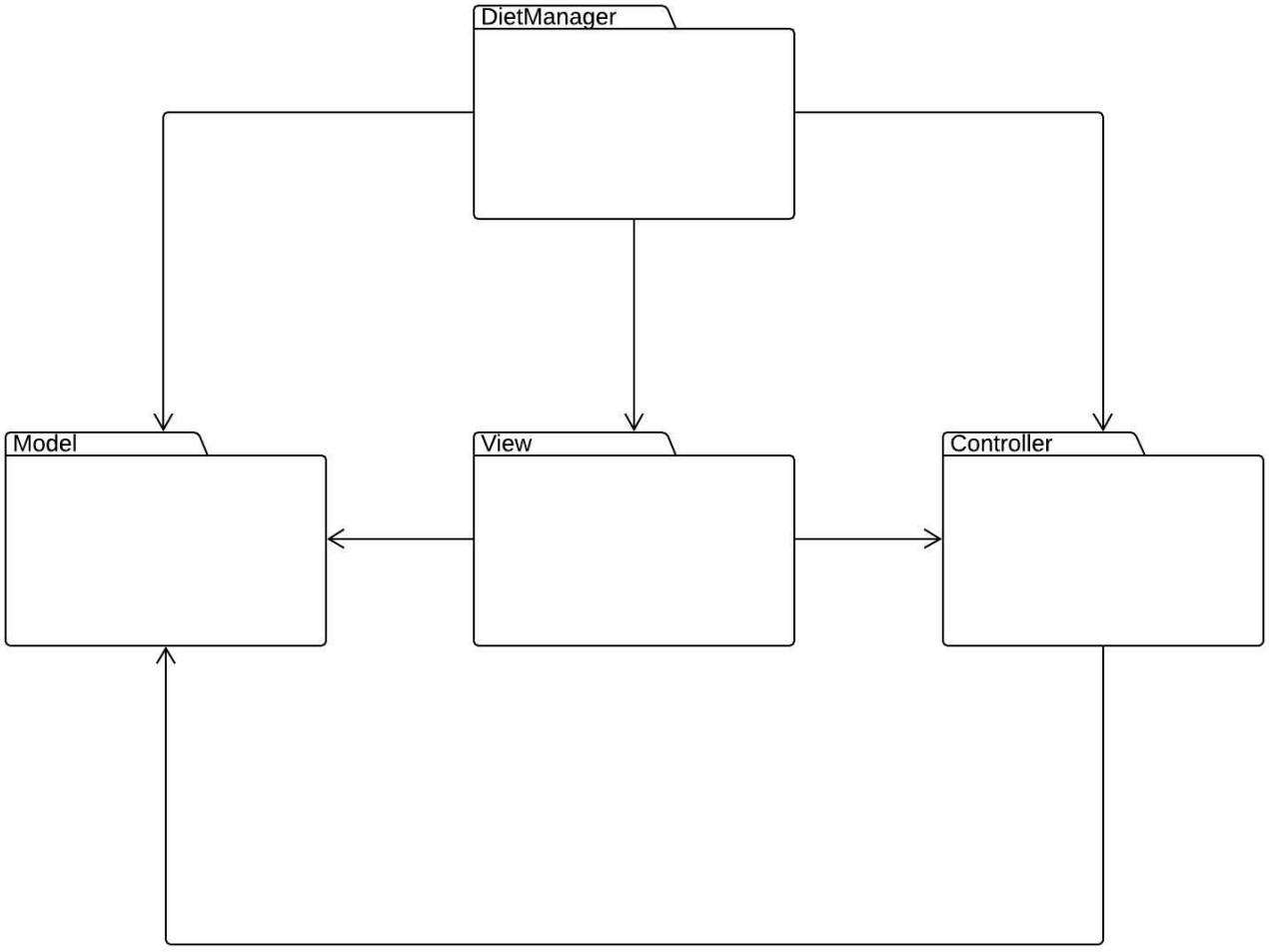
The model will store the system state when the program is running. This means the data from the user will be held and manipulated in the model.

The view is going to be responsible for displaying logged data, their respective graphs, saved foods/ recipes, and other data relevant to the user on a given date. It will have a well designed UI that allows a user to enter new foods, recipes and user information such as weight and caloric goals that will ultimately be handled by the model.

The controller is going to take in data from the view and use it to manipulate the data into our a format in which the model can understand. The controller will update the model when the view changes, and also update the view is certain instances. The view will be updated by the model as well. With each component orchestrating one functionality, it will foster high-cohesion and lead to classes only interacting with entities that are needed to function; which promotes low-coupling.

There is an external main class that will run and instantiate the subsystem pieces so that the program can run.

Subsystem Structure



The Model updates the view with relevant data as necessary and stores data

The View is constantly being updated by the Model as data changes

The Controller is constantly updating the model and sometimes updates the View when necessary

Subsystems

Subsystem - Model

Class BasicFood	
Responsibilities	Contain a food object that contains data about the food, including the name and nutritional values.
Collaborators (inheritance)	FoodInterface- SimpleFood implements the interface FoodInterface for operations and attributes regarding its name and nutritional data that is assigned to that object. LoggableItem - Implements interface LoggableItem to be able to submit

Class CalorieTarget	
Responsibilities	Generates the CalorieTarget object when the user wants to create a new calorie target.
Collaborators (inheritance)	Implements LoggableItem interface

Class DataFileManager	
Responsibilities	Reads and writes to a CSV file.
Collaborators (inheritance)	Uses the OpenCSV library to parse and write to a CSV file.

Class EntryCollection	
Responsibilities	Provide abstract functionality for subclasses to use such as using the DataFileManager and converting the collections to strings.
Collaborators (uses)	Extended by FoodCollection, ExerciseCollection, and Log

Class Exercise	
Responsibilities	Provides information about what an exercise object is made up of and stores the relevant values.
Collaborators (inheritance)	Implements LoggableItem

Class ExerciseCollection	
Responsibilities	Store exercise objects in an internal data structure and manage them with the use of the csv file through the parent EntryCollection class.
Collaborators (inheritance)	Extends EntryCollection, uses LogEntryFactory

Class FoodCollection	
Responsibilities	Stores FoodComponent objects in an internal data structure and uses its parent class, EntryCollection, to leverage a CSV file to store data when the program isn't running. Manages the collection of foods by offering different methods to create, read, update, and delete foods.
Collaborators (inheritance)	Extends EntryCollection, uses FoodFactory and FoodInterface objects.

Class FoodFactory	
Responsibilities	Creates the concrete food and recipe objects when needed by FoodCollection.
Collaborators (inheritance)	Is used by FoodCollection

Class FoodInterface	
Responsibilities	Provides the rules (methods) for any class that is to be considered a food.
Collaborators (inheritance)	Implements LoggableItem

Class Log	
Responsibilities	Stores loggable data in an internal data structure and provides functionality to create, read, update, and delete the data. Uses the parent EntryCollection class to leverage a csv file to store data while the program isn't running.
Collaborators (inheritance)	Extends EntryCollection, uses LogEntry, uses LoggableItem objects

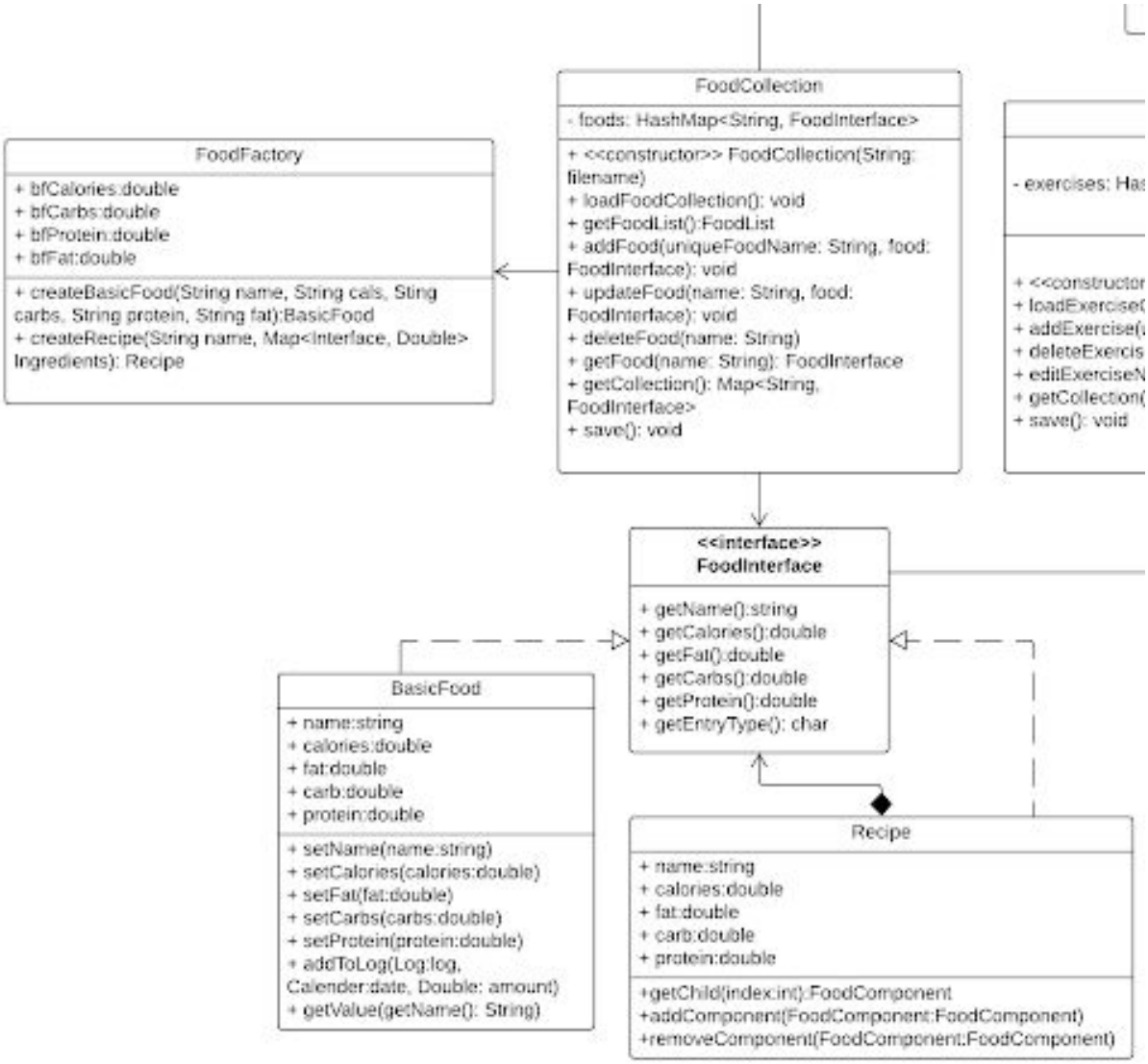
Class LogEntry	
Responsibilities	Stores the LoggableItem object with some other related data such as the amount, a unique id, and a date that represents where in the log the new entry will be.
Collaborators (inheritance)	Is used by Log.

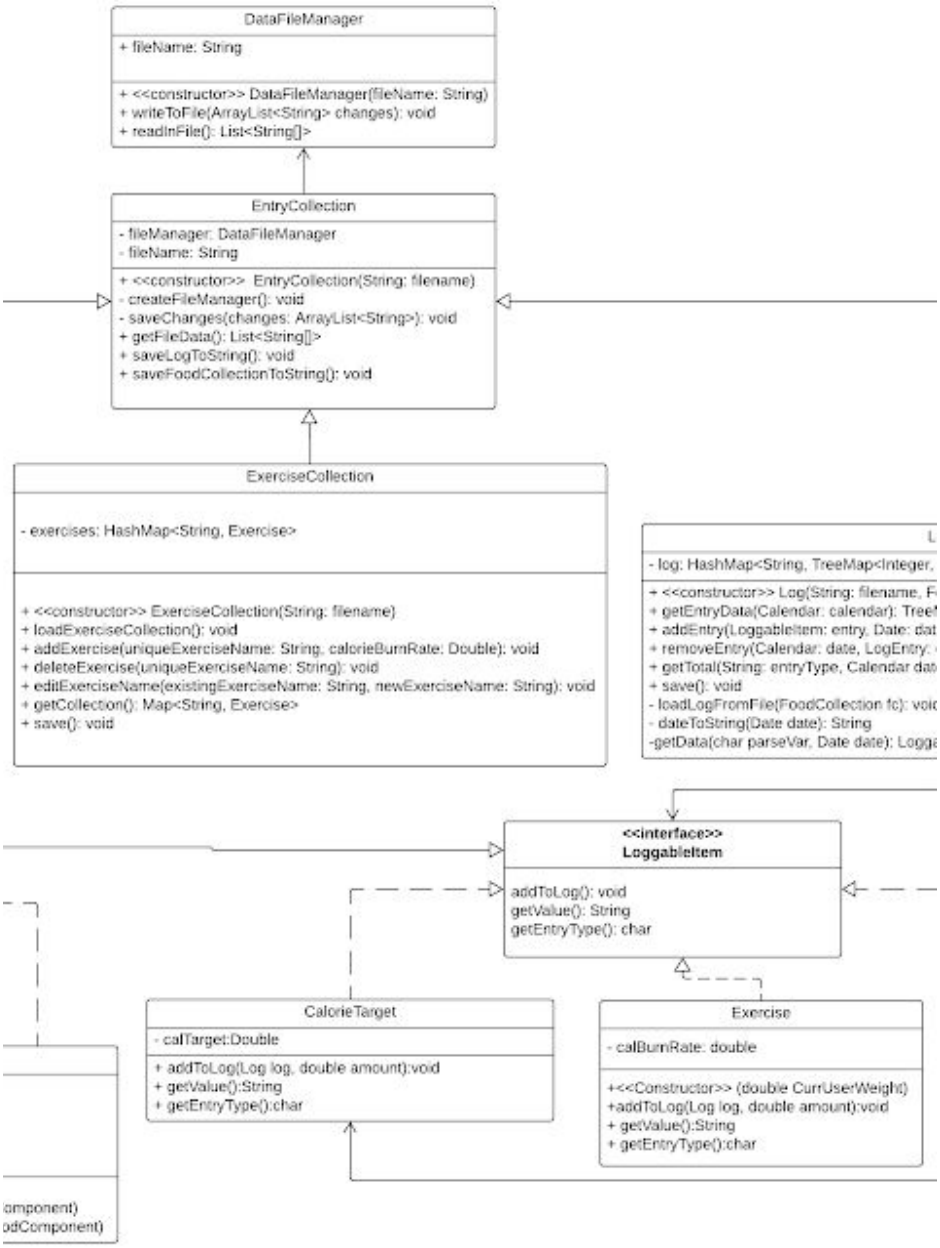
Class LoggableItem (interface)	
Responsibilities	Provide functionality for a food or recipe to add itself to the log. Anything meant to be logged must conform to this interface.
Collaborators (inheritance)	Used by the Log

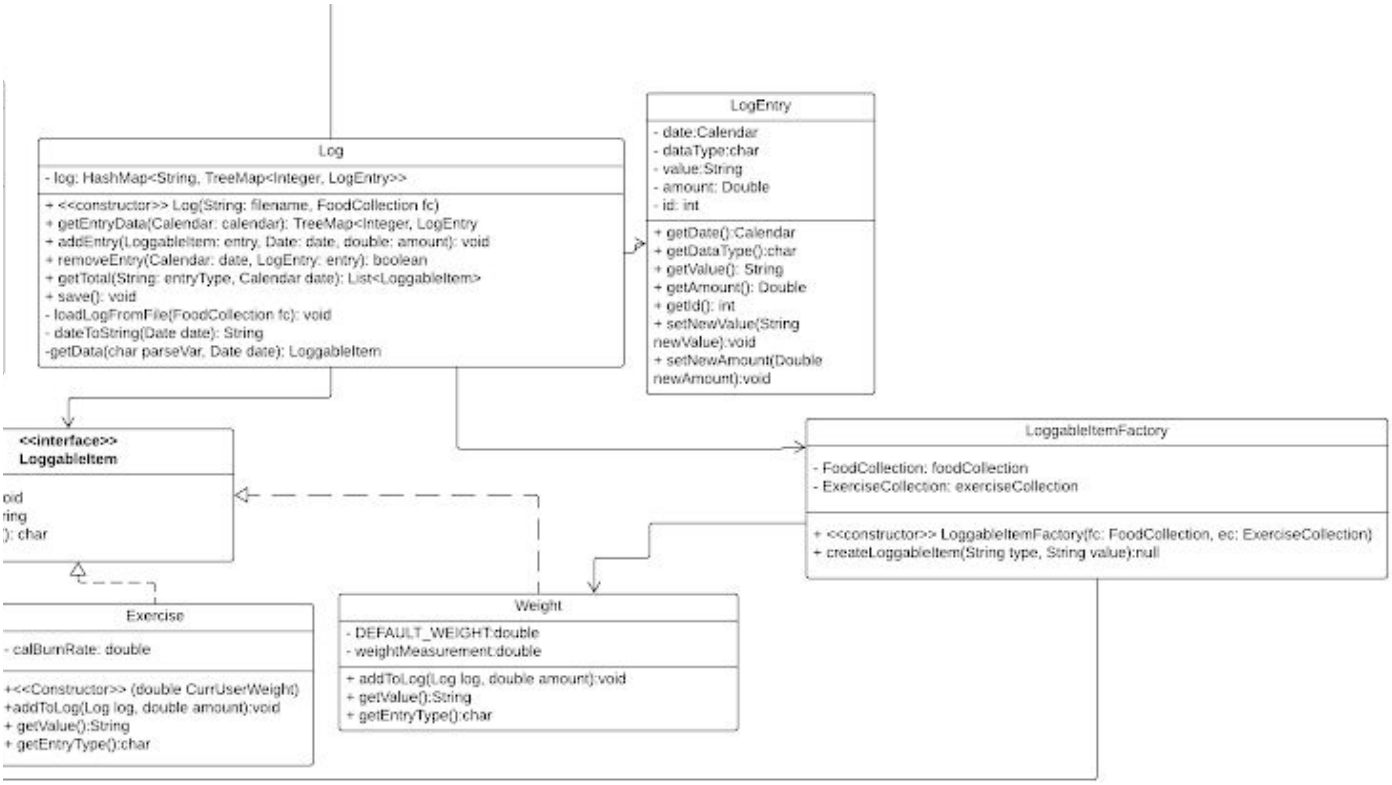
Class LoggableItemFactory	
Responsibilities	Creates LoggableItem objects that will be stored in the log. Used when reading in the log data file into an internal data structure. The exercise and food collections are injected.
Collaborators (inheritance)	Uses the injected collections, and creates Weight and CalorieTarget objects.

Class Recipe	
Responsibilities	Contain food objects and other recipe objects that contains data about the food or recipe, including the name and nutritional values. Add or remove components associated with that Recipe object.
Collaborators (inheritance)	Implements FoodInterface

Class Weight	
Responsibilities	Represents a weight which could be stored in the log. Stores relevant data about the weight value and type of loggable item.
Collaborators (inheritance)	Implements LoggableItem


FoodCollection
- foods: HashMap<String, FoodInterface>
+ <<constructor>> FoodCollection(String: filename)
+ loadFoodCollection(): void
+ getFoodList():FoodList
+ addFood(uniqueFoodName: String, food: FoodInterface): void
+ updateFood(name: String, food: FoodInterface): void
+ deleteFood(name: String)
+ getFood(name: String): FoodInterface
+ getCollection(): Map<String, FoodInterface>
+ save(): void
FoodFactory
+ bfCalories:double
+ bfCarbs:double
+ bfProtein:double
+ bfFat:double
+ createBasicFood(String name, String cals, Sting carbs, String protein, String fat):BasicFood
+ createRecipe(String name, Map<Interface, Double> Ingredients): Recipe
<<interface>>
FoodInterface
+ getName():string
+ getCalories():double
+ getFat():double
+ getCarbs():double
+ getProtein():double
+ getEntryType(): char
BasicFood
+ name:string
+ calories:double
+ fat:double
+ carb:double
+ protein:double
+ setName(name:string)
+ setCalories(calories:double)
+ setFat(fat:double)
+ setCarbs(carbs:double)
+ setProtein(protein:double)
+ addToLog(Log:log, Calender:date, Double: amount)
+ getValue(getName(): String)
Recipe
+ name:string
+ calories:double
+ fat:double
+ carb:double
+ protein:double
+getChild(index:int):FoodComponent
+addComponent(FoodComponent:FoodComponent)
+removeComponent(FoodComponent:FoodComponent)


DataFileManager
+ fileName: String
+ <<constructor>> DataFileManager(fileName: String)
+ writeToFile(ArrayList<String> changes): void
+ readInFile(): List<String[]>
EntryCollection
- fileManager: DataFileManager
- fileName: String
+ <<constructor>> EntryCollection(String: filename)
- createFileManager(): void
- saveChanges(changes: ArrayList<String>): void
+ getFileData(): List<String[]>
+ saveLogToString(): void
+ saveFoodCollectionToString(): void
ExerciseCollection
- exercises: HashMap<String, Exercise>
+ <<constructor>> ExerciseCollection(String: filename)
+ loadExerciseCollection(): void
+ addExercise(uniqueExerciseName: String, calorieBurnRate: Double): void
+ deleteExercise(uniqueExerciseName: String): void
+ editExerciseName(existingExerciseName: String, newExerciseName: String): void
+ getCollection(): Map<String, Exercise>
+ save(): void
- log: HashMap<String, TreeMap<Integer,
+ <<constructor>> Log(String: filename,
+ getEntryData(Calendar: calendar): Tree
+ addEntry(LoggableItem: entry, Date: dat
+ removeEntry(Calendar: date, LogEntry:
+ getTotal(String: entryType, Calendar dat
+ save(): void
- loadLogFromFile(FoodCollection fc): voi
- dateToString(Date date): String
-getData(char parseVar, Date date): Logg
<<interface>>
LoggableItem
addToLog(): void
getValue(): String
getEntryType(): char
CalorieTarget
- calTarget:Double
+ addToLog(Log log, double amount):void
+ getValue():String
+ getEntryType():char
Exercise
- calBurnRate: double
+<<Constructor>> (double CurrUserWeight)
+addToLog(Log log, double amount):void
+ getValue():String
+ getEntryType():char
omponent)
odComponent)



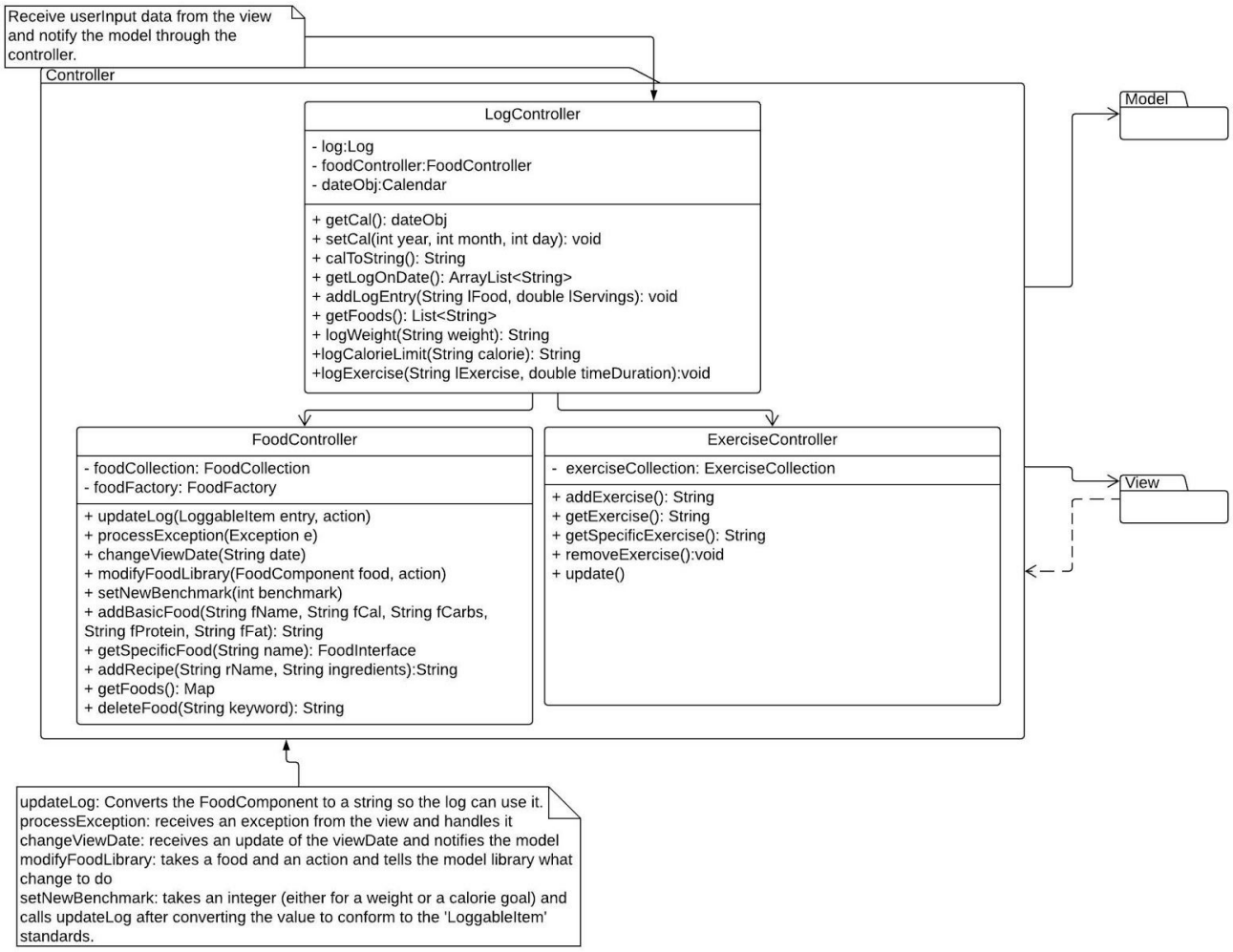
Subsystem - Controller

Class FoodController	
Responsibilities	Receive state change information from the view subsystem and update the appropriate model objects. Format user entered data so that model objects can use it. (ex. a user inputs a food name and the food library will know how to interpret that) Handle exceptions/errors that the user may run into.
Collaborators (uses)	The view subsystem will use the controller to update the model with changes. The controller will collaborate with the model when it needs to update it with changes from the view. The controller will update the view under certain circumstances.

Class LogController	
Responsibilities	Receive state change information from the view subsystem and update the appropriate model objects. Format user entered data so that model objects can use it. (ex. a user inputs a food name and the food library will know how to interpret that)

	Handle exceptions/errors that the user may run into.
Collaborators (uses)	FoodController - LogController allows for logging of foods from the foodController

Class ExerciseController	
Responsibilities	Handles the actions and logic from CreateExercisePanel. Retrieves and adds data to ExerciseCollection.
Collaborators (uses)	ExerciseCollection, LogController



Subsystem - View

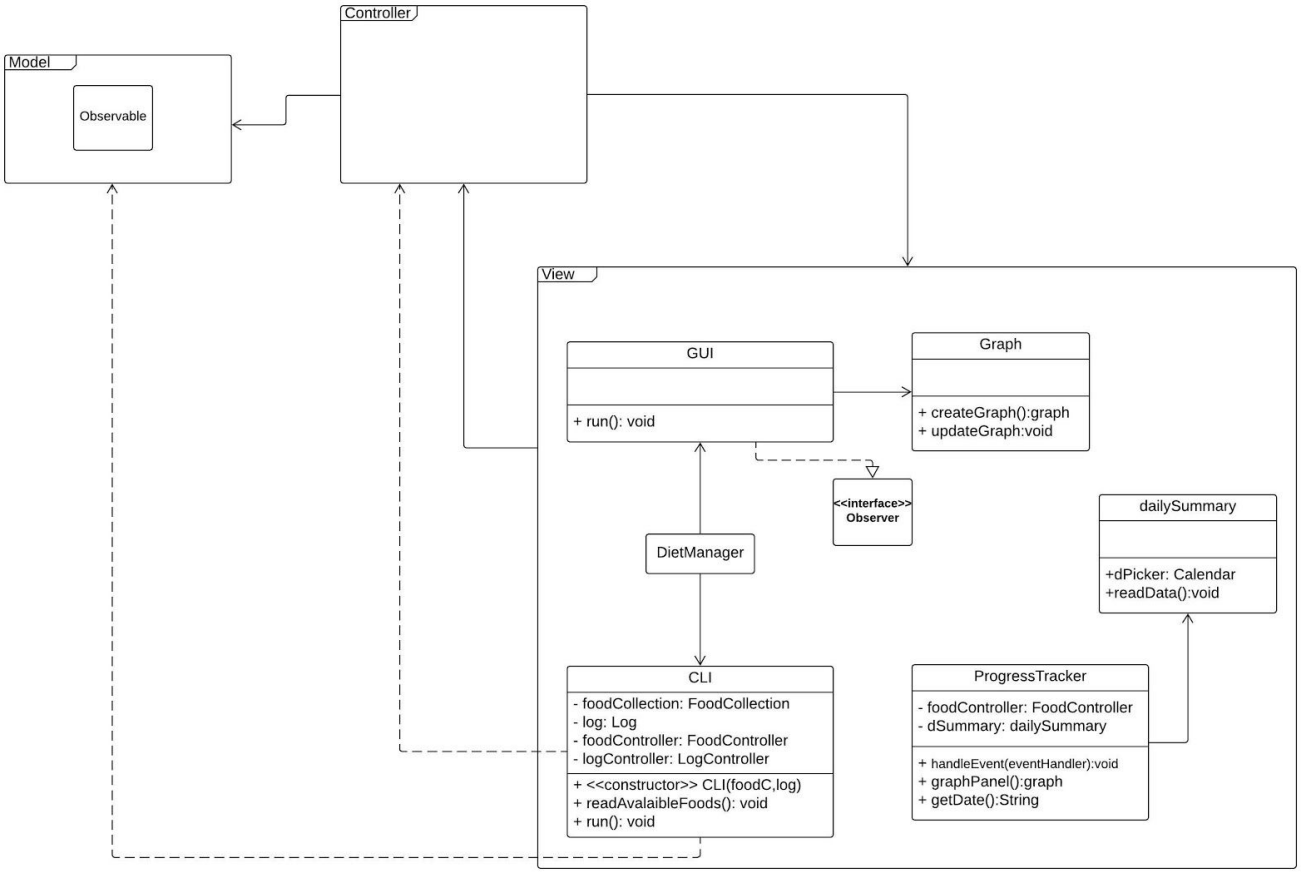
Class LogPanel	
Responsibilities	Receives user input of weight, calorie goal, or a food to add or delete, to their log.
Collaborators (uses)	EventHandler - Implements EventHandler interface to check for user input when user is interacting with panel.

Class CLI	
Responsibilities	Receives user input of Adding a new food to food library, removing a food from food library
Collaborators (uses)	EventHandler - Implements EventHandler interface to check for user input when user is interacting with panel.

Class DailySummary	
Responsibilities	Displays daily summary of data to user including: Calorie intake goal, current weight, and current amount of daily calories consumed
Collaborators (uses)	

Class ProgressTracker	
Responsibilities	A dynamic view of the changing user data such as weight and calories consumed/ goal.
Collaborators (uses)	dailySummary, EventHandler - implements EventHandler interface to check for user input when user is interacting with panel.

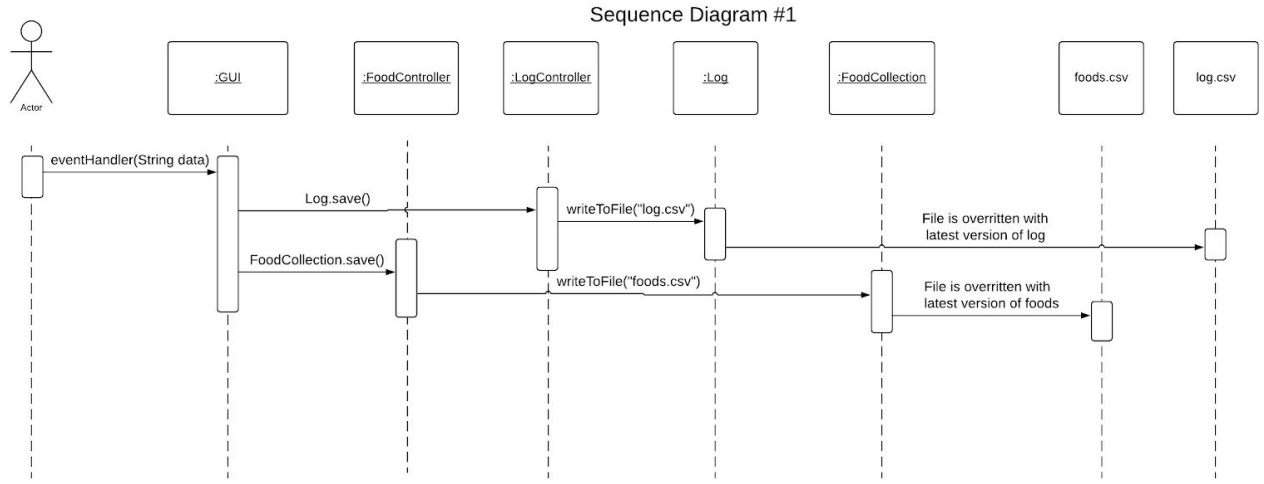
Class EventHandler (Interface)	
Responsibilities	Takes in a listener for user input on view panels. Event listener must conform to the EventHandler interface.


Controller
Model
Observable
View
GUI
+ run(): void
Graph
+ createGraph():graph
+ updateGraph:void
<<interface>>
Observer
DietManager
dailySummary
+dPicker: Calendar
+readData():void
CLI
- foodCollection: FoodCollection
- log: Log
- foodController: FoodController
- logController: LogController
+ <<constructor>> CLI(foodC,log)
+ readAvalaibleFoods(): void
+ run(): void
ProgressTracker
- foodController: FoodController
- dSummary: dailySummary
+ handleEvent(eventHandler):void
+ graphPanel():graph
+ getDate():String

Sequence Diagrams

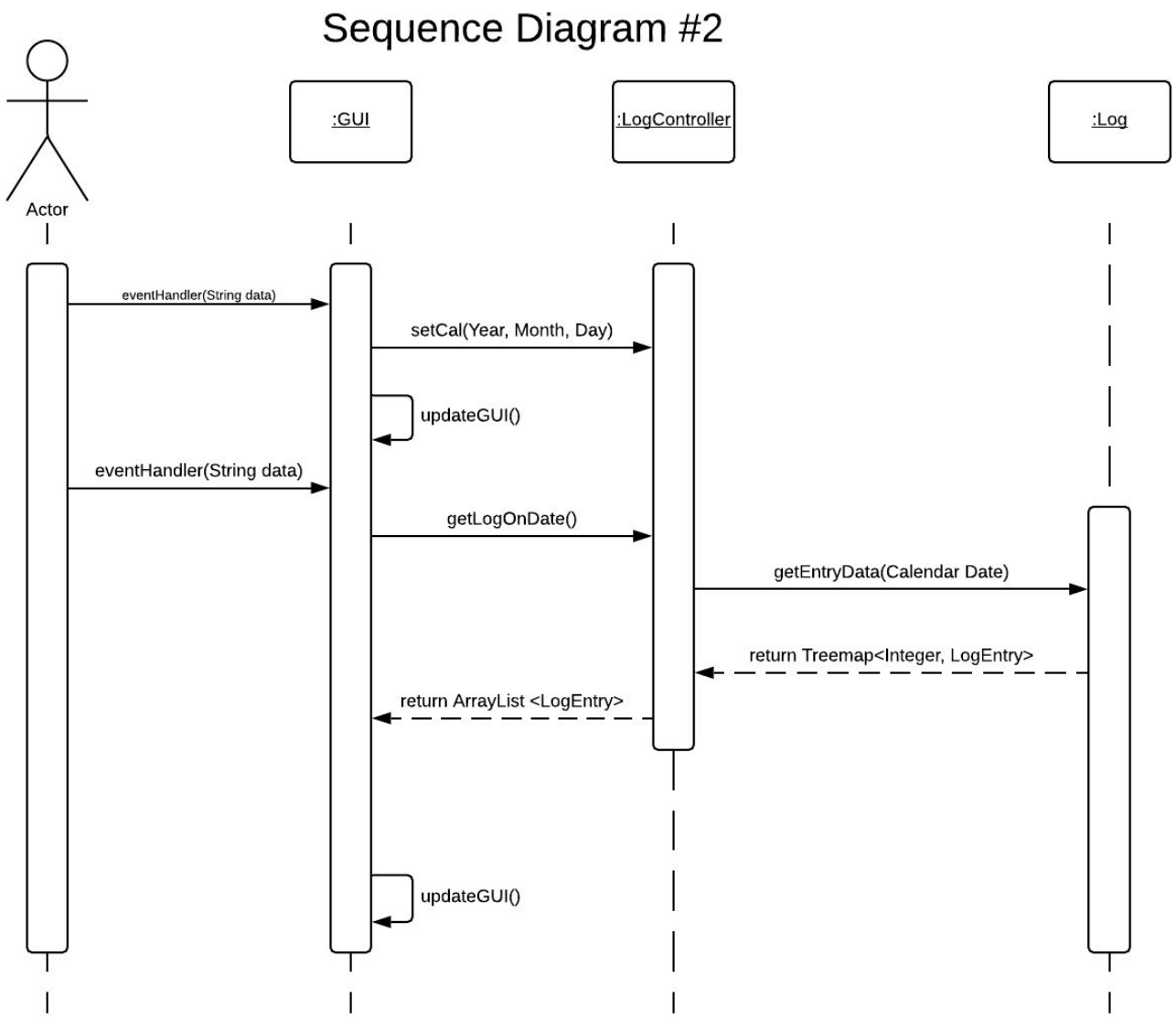
Sequence Diagram #1 Description

The user is saving all changes made since the last save to the appropriate data files.



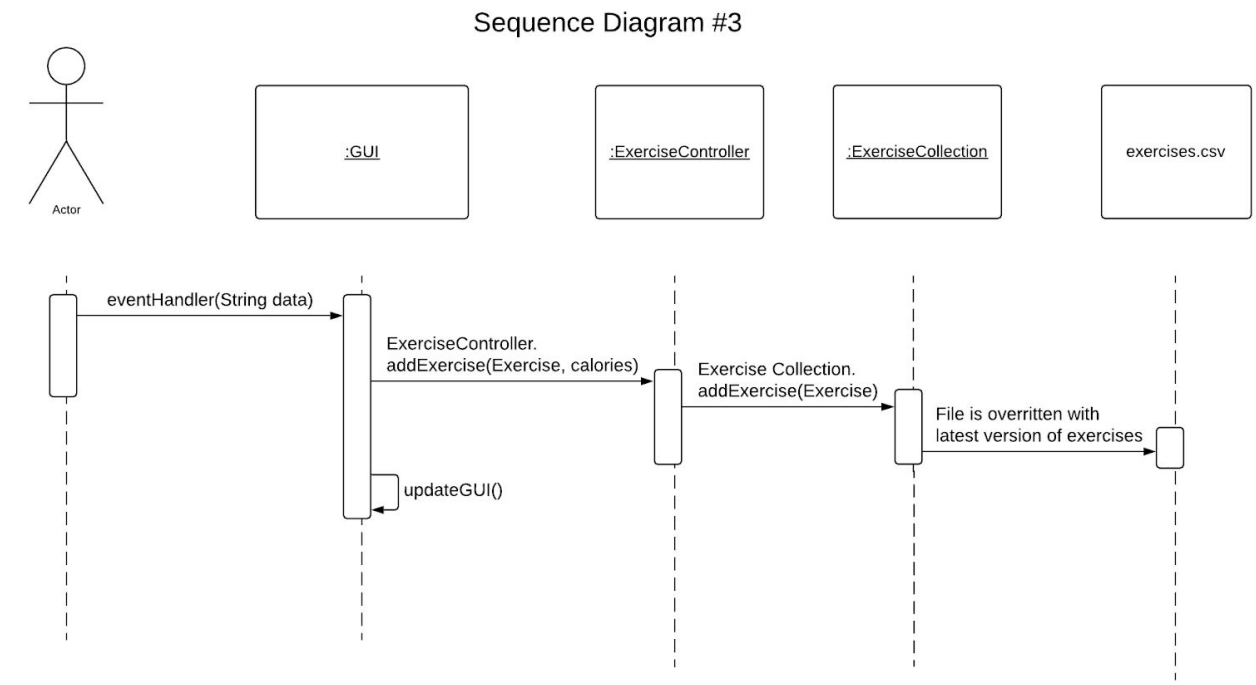
Sequence Diagram #2 Description

The user has initiated a date change to pull up log data from a previous date.



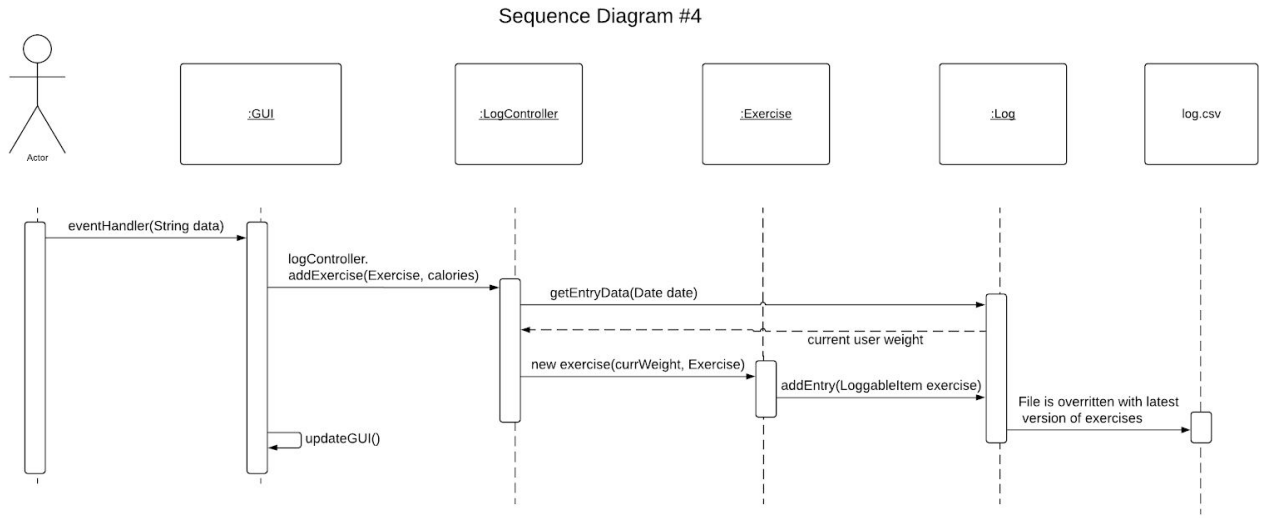
Sequence Diagram #3 Description

The user adds a new exercise into the collection.



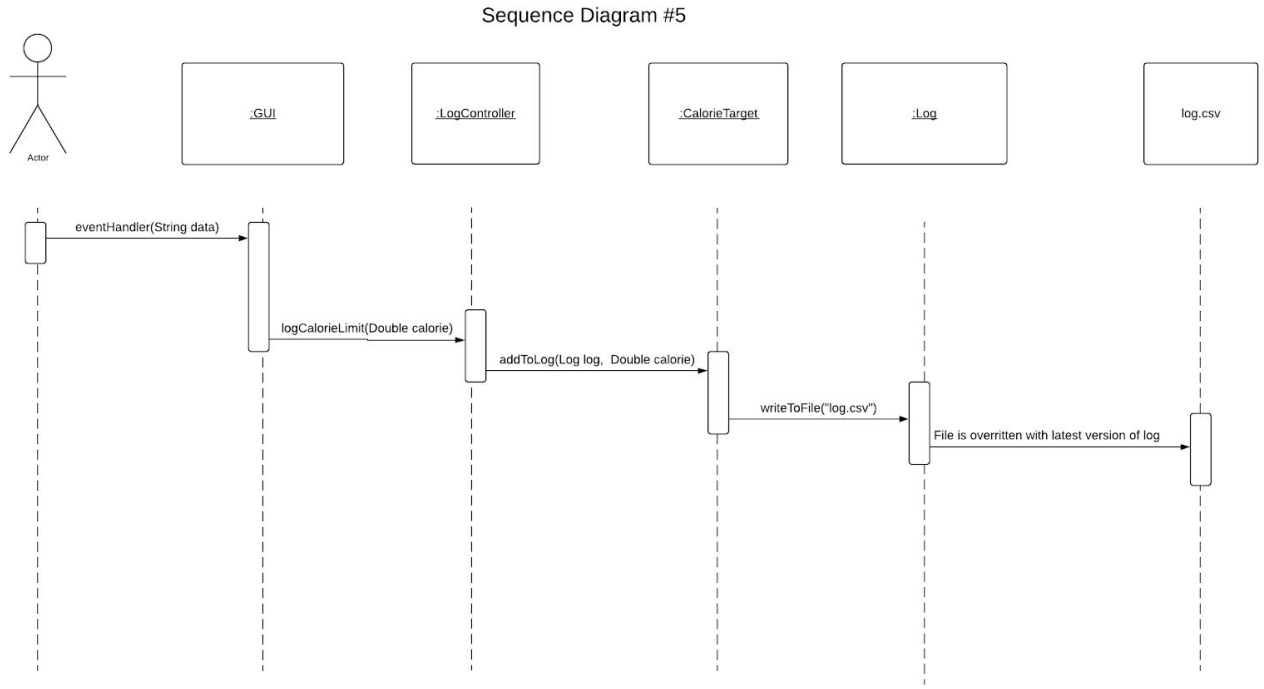
Sequence Diagram #4 Description

The user adds a new exercise entry into the log.



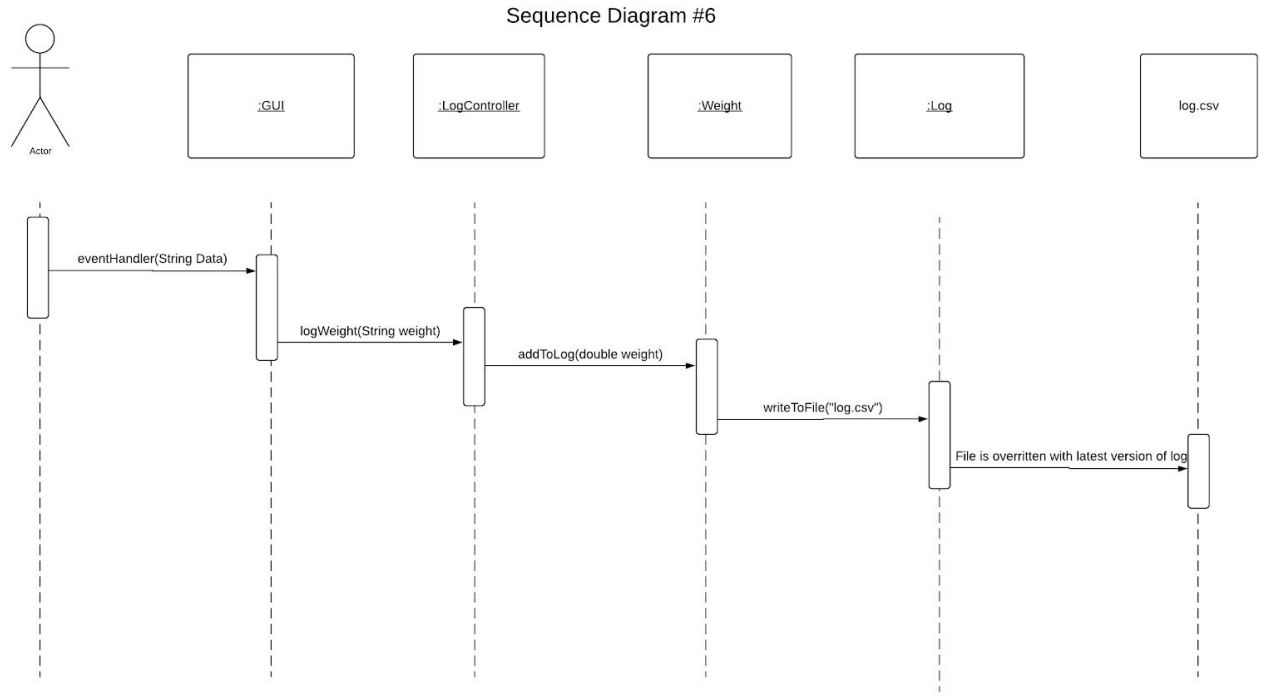
Sequence Diagram #5 Description

The user specify calorie limit



Sequence Diagram #6 Description

The user specify his/her weight



Pattern Usage

Pattern #1 MVC

MVC Pattern	
Model	BasicFood CalorieTarget DataFileManager EntryCollection FoodCollection ExerciseCollection FoodFactory FoodInterface Log LogEntry LogEntryFactory LoggableItem Exercise Recipe Weight
View	CLI GUI
Controller	FoodController LogController

Pattern #2 Composite

Composite Pattern	
Component	FoodInterface
Composite	Recipe
Leaf	Food