Diet Manager 2.0

Project Design Document

TEAM A
Matthew Marchinetti <mam5588@g.rit.edu>
Xin Liu <xl4998@g.rit.edu>
Zach Easley <zpe4421@g.rit.edu>
Amina Mahmood <axm6392@rit.edu>
Samuel Ilesanmi <soi6269@rit.edu>

Project Summary

Obesity is an ongoing epidemic in America and is often thought to be caused by the misunderstanding of caloric intake and dietary values of common foods. Diet Manager is an application that seeks to help users pursue their diet goals by keeping them informed on what they are eating daily. Users of the application can set their daily caloric limit and weight, add foods and exercises, log foods and exercises, and ultimately be able to view their daily summaries to make it more easy to watch their daily intake and balance their nutrition.

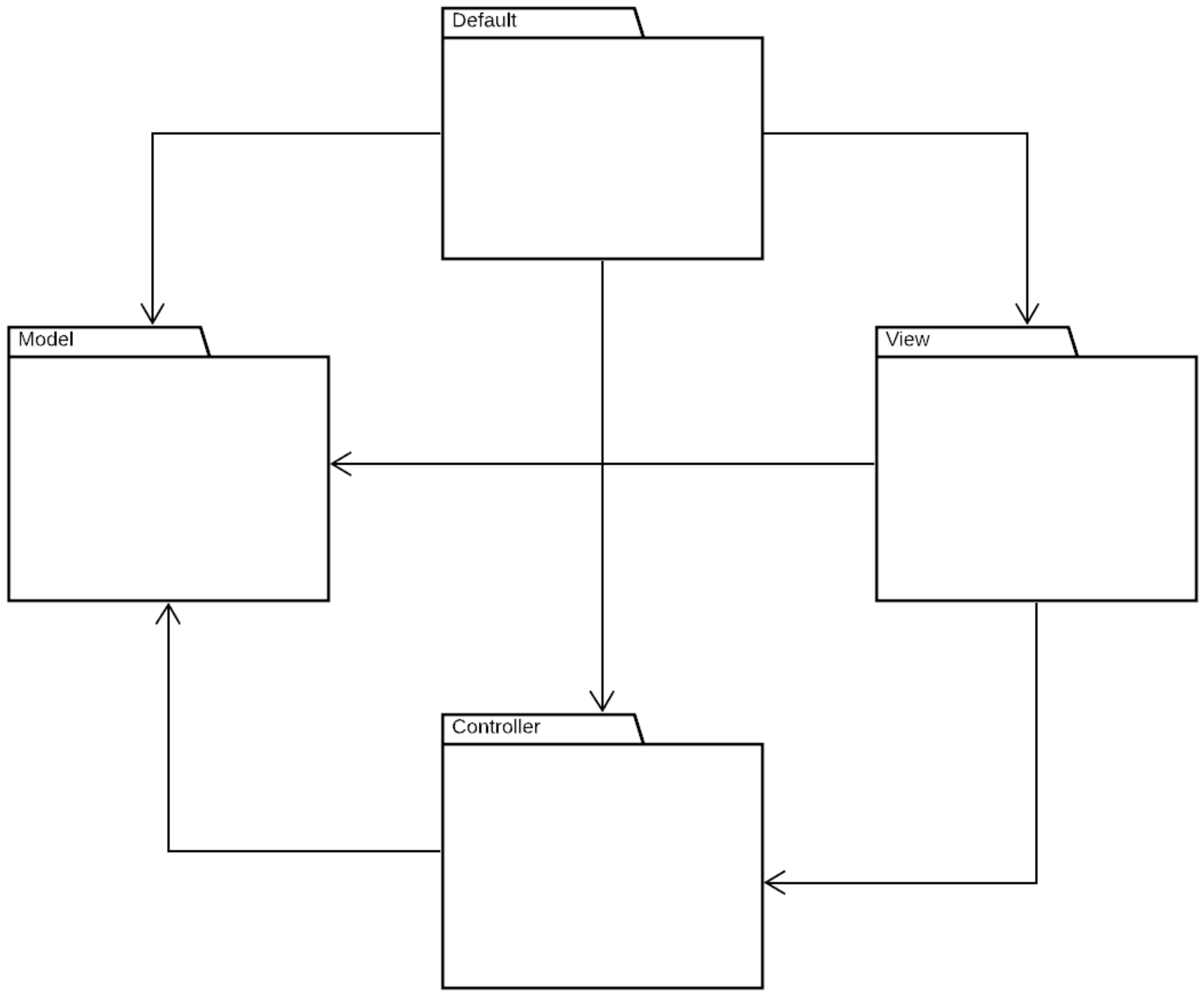
Design Overview

We designed this application with the main intention of maintaining a strong separation of concerns using the Model-View-Controller design pattern and this is what guided many of our major decisions.

We separated the desired functionality into three distinct types of operations: **creating new foods and exercises**, **logging foods and exercises to the daily log** and **calculating and displaying the daily totals and other data relevant to the user's selected day**. This allowed us to create separate panels (views) to make up the user interface: we needed panels for creating a new food, recipe, and exercise, a panel for logging foods or recipes and exercise, and a panel which displays updated summary data upon interaction with one of the other panels. By following this approach, our application will be built using high-cohesion and low-coupling; classes will only interact with classes that are needed to function properly. We designed the operations to act in a somewhat cyclical fashion so that, when a user interacts with a view, that view communicates to its controller. That controller then updates the models and the models send updates to any pertinent views before returning control to the user.

We had an issue with our original design of the LogCollection class as we did not realize that storing the consumed foods in a hashmap would not allow us to track multiple entries of the same food for the same day, however we revised the design to store it in a list instead and it was much easier to work with. We also originally designed the controllers to send updates to the views rather than simply allowing the model to be an observable. However, following the latter method allows for much lower coupling as well as separation of concerns as the views can pull in data from the models directly and not rely on the controller to effectively provide wrapper functions to the models.

Subsystem Structure



Default Subsystem - Starts the application, displays the user interface, uses IOHandler in the Controller Subsystem to read/write csvs and populates data in the Model Subsystem.

Model Subsystem - Stores all of the data and sends updates to views on data change.

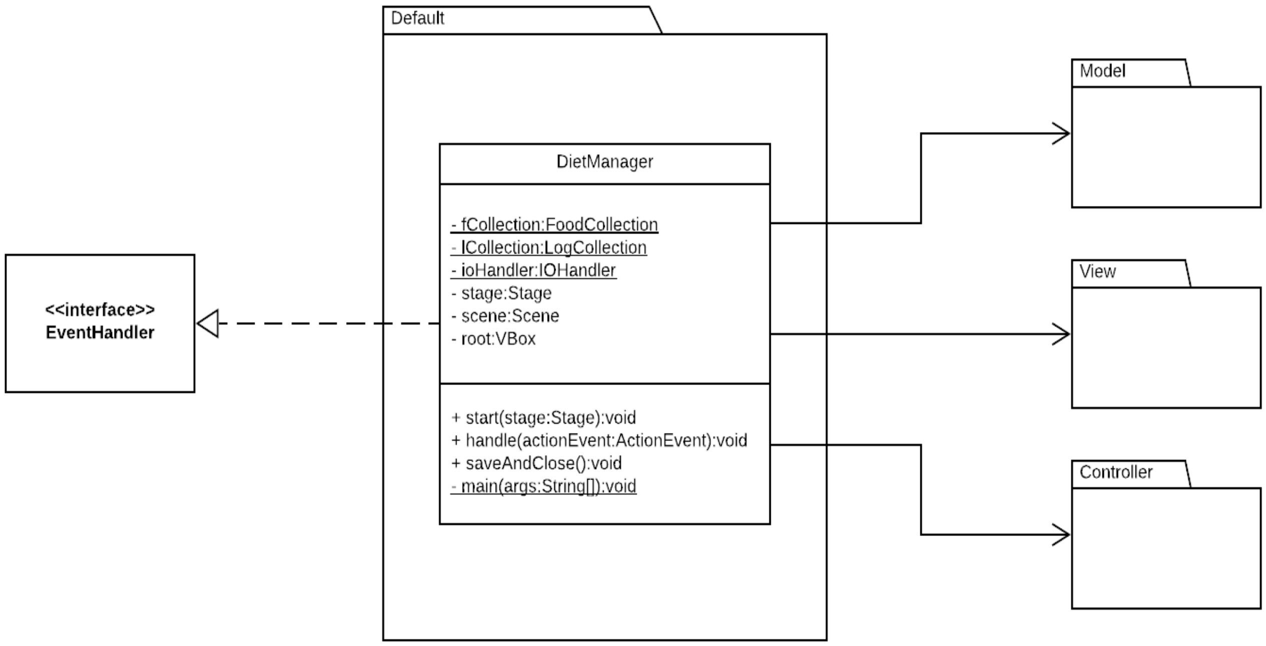
View Subsystem - Provides user interface for altering and displaying data to the user.

Controller Subsystem - Validates user input and prepares data for storage in the models.

Subsystems

Default Subsystem

Class DietManager	
Responsibilities	The “main” class; starts the JavaFx GUI and CLI.
Collaborators (uses)	FoodCollection, ExerciseCollection, LogCollection, IOHandler


Default
DietManager
- fCollection:FoodCollection
- lCollection:LogCollection
- ioHandler:IOHandler
- stage:Stage
- scene:Scene
- root:VBox
+ start(stage:Stage):void
+ handle(actionEvent:ActionEvent):void
+ saveAndClose():void
- main(args:String[]):void
<<interface>>
EventHandler
Model
View
Controller

View Subsystem

Class: NutritionTrackerPanel	
Responsibilities	Displays summary data to the user for the current day's total nutritional values. Listens for updates from the LogCollection to update its UI.
Collaborators (uses)	NutritionTrackerController, GraphPanel, FoodCollection, LogCollection, ExerciseCollection, Observer, EventHandler<ActionEvent>

Class: GraphPanel	
Responsibilities	Prepares a graph for the NutritionTrackerPanel to utilize.
Collaborators (uses)	N/A

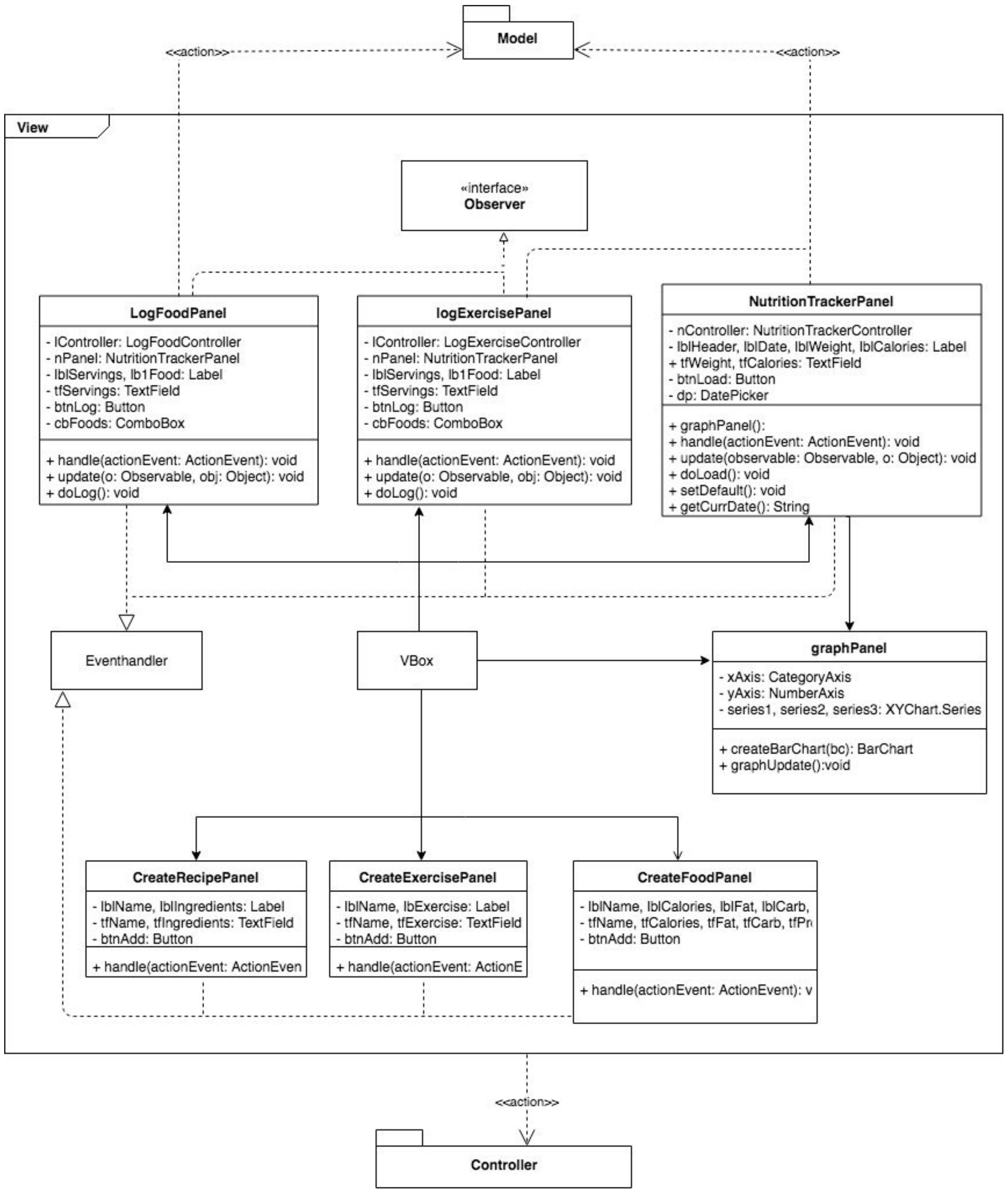
Class: CreateFoodPanel	
Responsibilities	Enables users to create new basic food items by entering nutrition information.
Collaborators (uses)	FoodController, FoodCollection, Observer, EventHandler<ActionEvent>

Class: CreateRecipePanel	
Responsibilities	Enables users to create new recipe items by entering nutrition information.
Collaborators (uses)	FoodController, FoodCollection, Observer, EventHandler<ActionEvent>

Class: CreateExercisePanel	
Responsibilities	Enables users to create new exercise by entering exercise information.
Collaborators (uses)	ExerciseController, Observer, EventHandler<ActionEvent>

Class: LogFoodPanel	
Responsibilities	Enables users to log, delete food or recipes for the day as well as allowing the user to view all logged foods for the selected day.
Collaborators (uses)	NutritionTrackerPanel, LogController, FoodCollection, Observer, EventHandler<ActionEvent>

Class: LogExercisePanel	
Responsibilities	Enables users to log, delete an exercise for the day as well as allowing the user to view all logged exercises for the selected day.
Collaborators (uses)	NutritionTrackerPanel, LogController, ExerciseCollection, Observer, EventHandler<ActionEvent>


Model
<<action>>
<<action>>
View
«interface»
Observer
LogFoodPanel
- lController: LogFoodController
- nPanel: NutritionTrackerPanel
- lblServings, lb1Food: Label
- tfServings: TextField
- btnLog: Button
- cbFoods: ComboBox
+ handle(actionEvent: ActionEvent): void
+ update(o: Observable, obj: Object): void
+ doLog(): void
logExercisePanel
- lController: LogExerciseController
- nPanel: NutritionTrackerPanel
- lblServings, lb1Food: Label
- tfServings: TextField
- btnLog: Button
- cbFoods: ComboBox
+ handle(actionEvent: ActionEvent): void
+ update(o: Observable, obj: Object): void
+ doLog(): void
NutritionTrackerPanel
- nController: NutritionTrackerController
- lblHeader, lblDate, lblWeight, lblCalories: Label
+ tfWeight, tfCalories: TextField
- btnLoad: Button
- dp: DatePicker
+ graphPanel():
+ handle(actionEvent: ActionEvent): void
+ update(observable: Observable, o: Object): void
+ doLoad(): void
+ setDefault(): void
+ getCurrDate(): String
Eventhandler
VBox
graphPanel
- xAxis: CategoryAxis
- yAxis: NumberAxis
- series1, series2, series3: XYChart.Series
+ createBarChart(bc): BarChart
+ graphUpdate():void
CreateRecipePanel
- lblName, lblIngredients: Label
- tfName, tfIngredients: TextField
- btnAdd: Button
+ handle(actionEvent: ActionEven
CreateExercisePanel
- lblName, lbExercise: Label
- tfName, tfExercise: TextField
- btnAdd: Button
+ handle(actionEvent: ActionE
CreateFoodPanel
- lblName, lblCalories, lblFat, lblCarb,
- tfName, tfCalories, tfFat, tfCarb, tfPr
- btnAdd: Button
+ handle(actionEvent: ActionEvent): v
<<action>>
Controller

Model Subsystem

Class: LogCollection	
Responsibilities	Stores the collective information of logged foods, logged exercises, weights and calorie limits by date using maps.
Collaborators (uses)	Observable

Class: FoodCollection	
Responsibilities	Stores the collective information of all the foods read in from food.csv as a key/value pair using maps.
Collaborators (uses)	IFood <<Interface>>, Food, Recipe, Observable

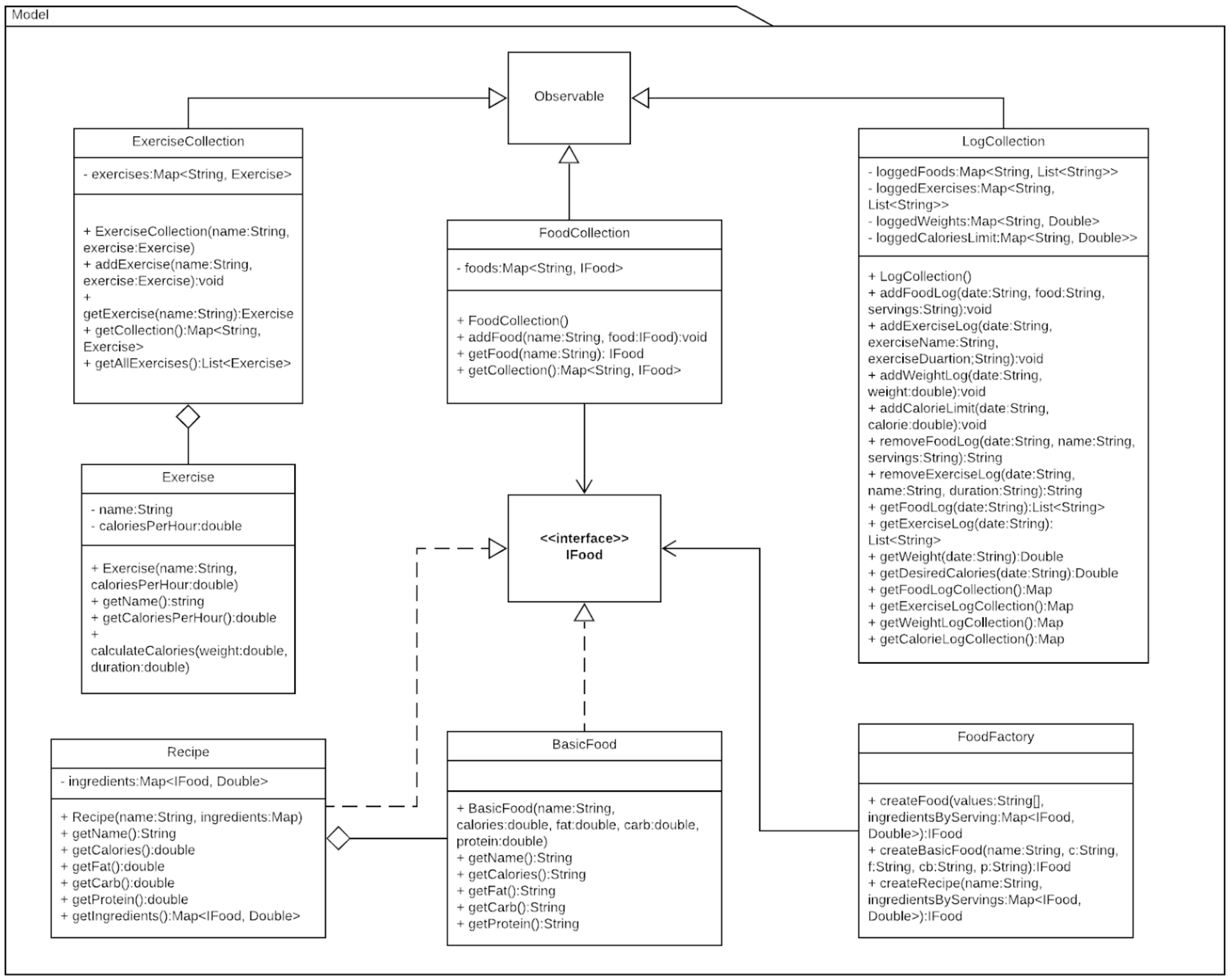
Class: BasicFood	
Responsibilities	Stores all the nutritional information of a food.
Collaborators (uses)	IFood <<Interface>>

Class: Recipe	
Responsibilities	Stores all total nutritional information of the recipe based on its ingredients, and a map of its ingredients (key) and servings (value).
Collaborators (uses)	IFood <<Interface>>, BasicFood, Recipe

Class: IFood <<Interface>>	
Responsibilities	The interface that all recipes and food items will follow for creation and to promote extensibility.
Collaborators	N/A

Class: ExerciseCollection	
Responsibilities	Stores the collective information of exercises.
Collaborators (uses)	Exercise, Observable

Class: Exercise	
Responsibilities	Stores data describing an exercise including name and calories per hour. Additionally, it calculates the calories given weight and duration.
Collaborators (uses)	N/A


Model
Observable
ExerciseCollection
- exercises:Map<String, Exercise>
+ ExerciseCollection(name:String, exercise:Exercise)
+ addExercise(name:String, exercise:Exercise):void
+ getExercise(name:String):Exercise
+ getCollection():Map<String, Exercise>
+ getAllExercises():List<Exercise>
FoodCollection
- foods:Map<String, IFood>
+ FoodCollection()
+ addFood(name:String, food:IFood):void
+ getFood(name:String): IFood
+ getCollection():Map<String, IFood>
LogCollection
- loggedFoods:Map<String, List<String>>
- loggedExercises:Map<String, List<String>>
- loggedWeights:Map<String, Double>
- loggedCaloriesLimit:Map<String, Double>>
+ LogCollection()
+ addFoodLog(date:String, food:String, servings:String):void
+ addExerciseLog(date:String, exerciseName:String, exerciseDuartion;String):void
+ addWeightLog(date:String, weight:double):void
+ addCalorieLimit(date:String, calorie:double):void
+ removeFoodLog(date:String, name:String, servings:String):String
+ removeExerciseLog(date:String, name:String, duration:String):String
+ getFoodLog(date:String):List<String>
+ getExerciseLog(date:String): List<String>
+ getWeight(date:String):Double
+ getDesiredCalories(date:String):Double
+ getFoodLogCollection():Map
+ getExerciseLogCollection():Map
+ getWeightLogCollection():Map
+ getCalorieLogCollection():Map
Exercise
- name:String
- caloriesPerHour:double
+ Exercise(name:String, caloriesPerHour:double)
+ getName():string
+ getCaloriesPerHour():double
+ calculateCalories(weight:double, duration:double)
<<interface>>
IFood
Recipe
- ingredients:Map<IFood, Double>
+ Recipe(name:String, ingredients:Map)
+ getName():String
+ getCalories():double
+ getFat():double
+ getCarb():double
+ getProtein():double
+ getIngredients():Map<IFood, Double>
BasicFood
+ BasicFood(name:String, calories:double, fat:double, carb:double, protein:double)
+ getName():String
+ getCalories():String
+ getFat():String
+ getCarb():String
+ getProtein():String
FoodFactory
+ createFood(values:String[], ingredientsByServing:Map<IFood, Double>):IFood
+ createBasicFood(name:String, c:String, f:String, cb:String, p:String):IFood
+ createRecipe(name:String, ingredientsByServings:Map<IFood, Double>):IFood

Controller Subsystem

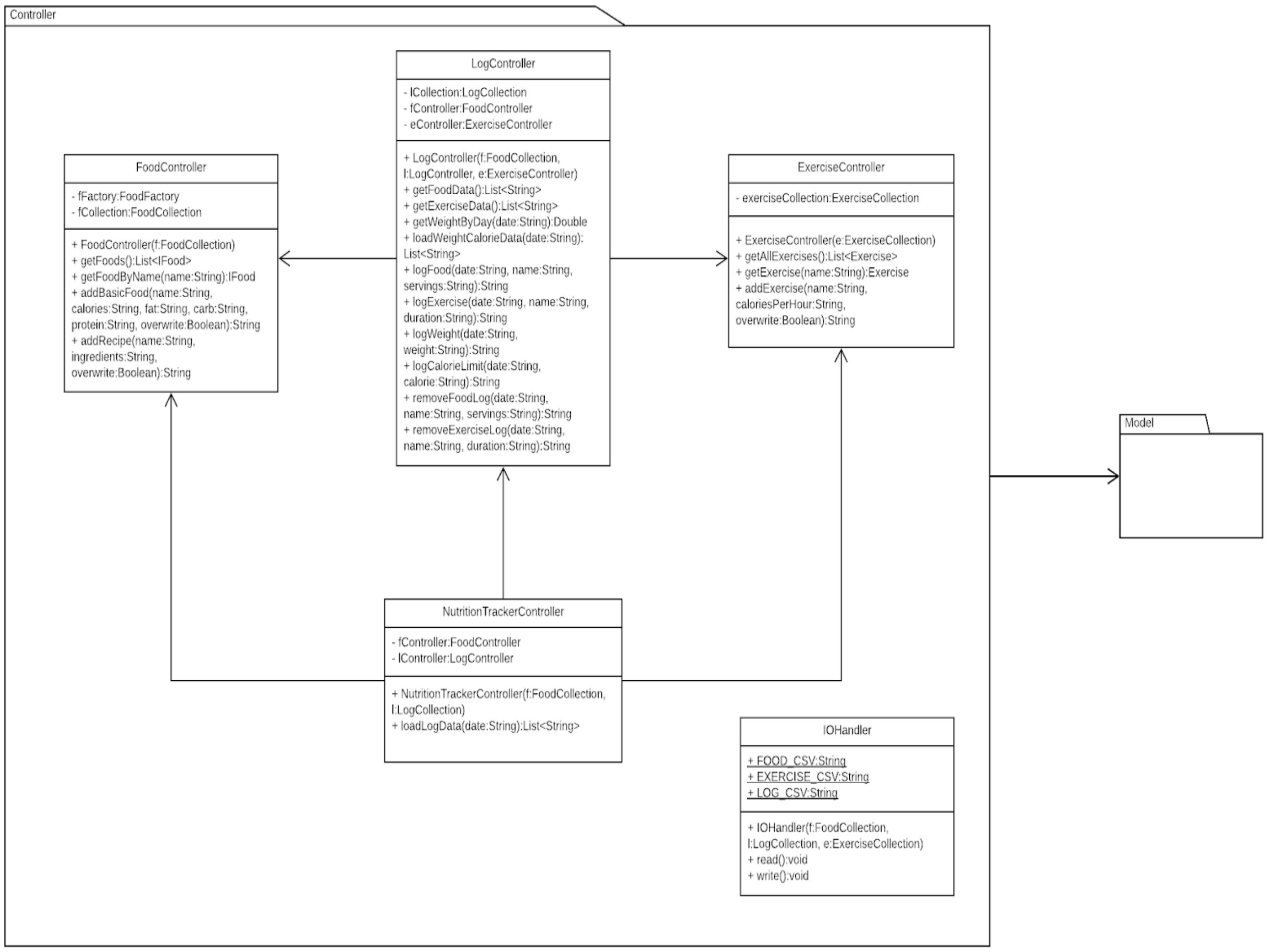
Class FoodController	
Responsibilities	Handles the actions and logic from the food related view classes. Retrieves and adds data to FoodCollection.
Collaborators (uses)	FoodFactory, FoodCollection, CreateFoodPanel, CreateRecipePanel

Class LogController	
Responsibilities	Handles the actions and logic from log related view classes. Retrieves, removes and adds data to LogCollection.
Collaborators (uses)	LogCollection, FoodController, ExerciseController, LogFoodPanel, LogExercisePanel

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

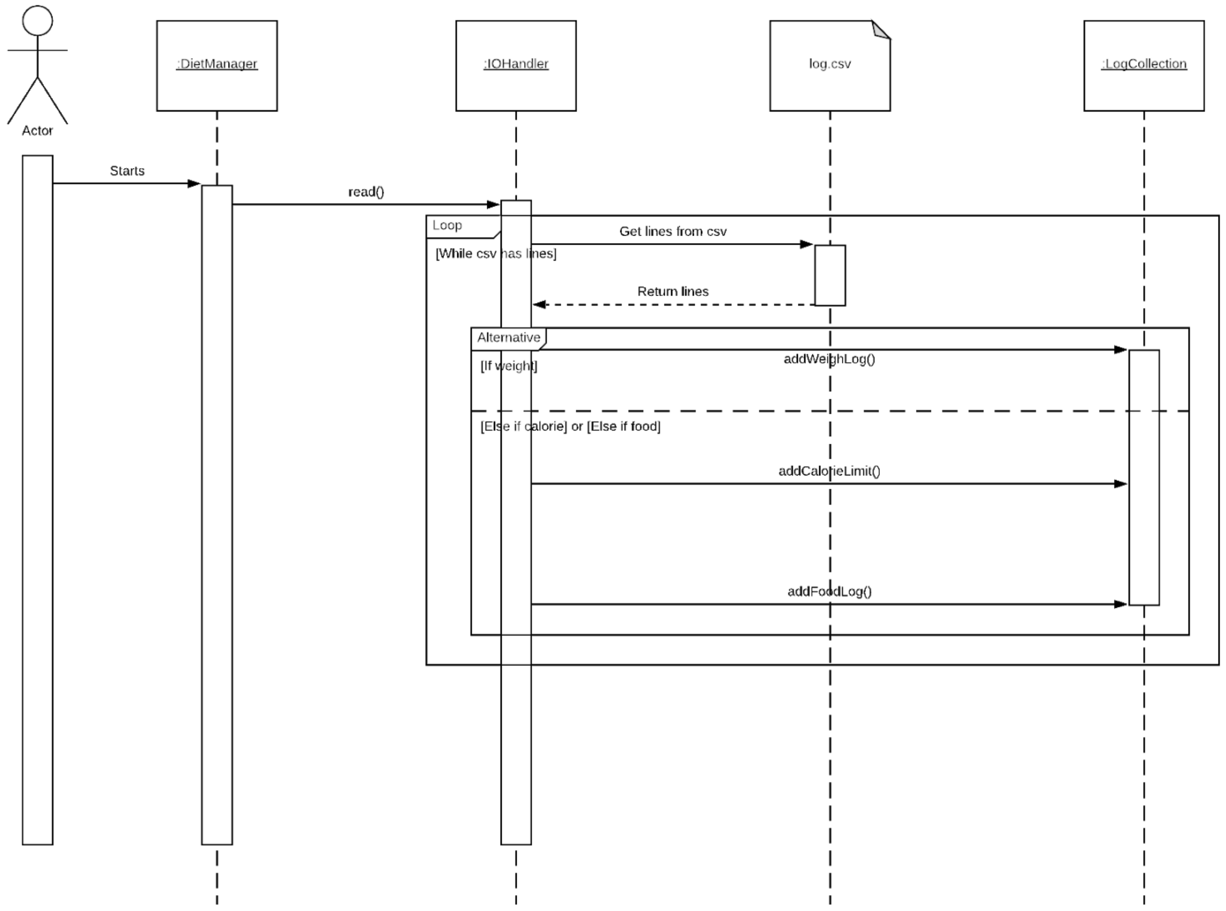
Class NutritionTrackerController	
Responsibilities	Handles the actions and logic from the NutritonTrackerPanel. Retrieves, calculates and set data for the display.
Collaborators (uses)	FoodController, LogController, ExerciseController, NutritionTrackerPanel

Class IOHandler	
Responsibilities	Reads from food.csv, log.csv, exercise.csv Writes to food.csv, log.csv, exercise.csv Prepares the models using data from the csv files.
Collaborators (uses)	FoodFactory, FoodCollection, LogCollection, ExerciseCollection

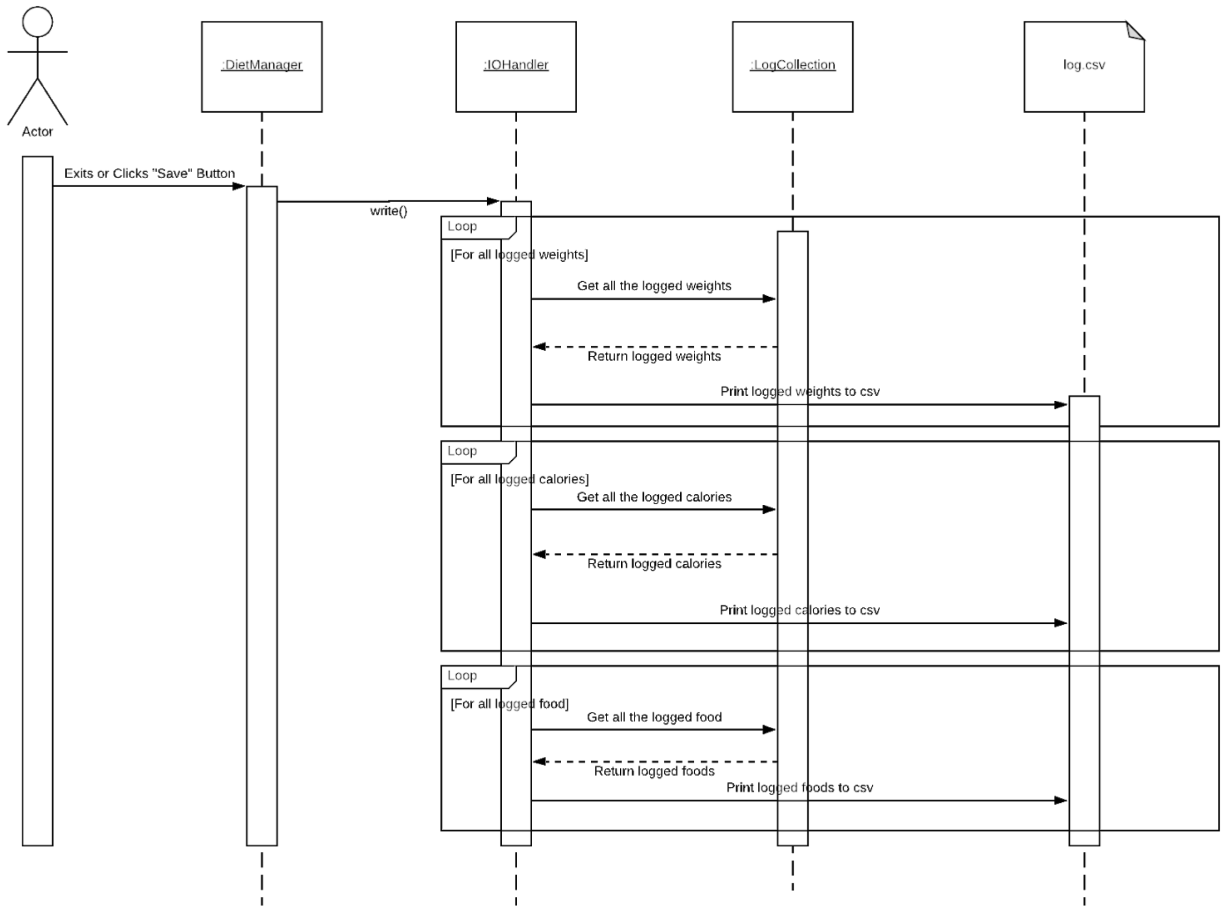

Controller
LogController
- lCollection:LogCollection
- fController:FoodController
- eController:ExerciseController
+ LogController(f:FoodCollection, l:LogController, e:ExerciseController)
+ getFoodData():List<String>
+ getExerciseData():List<String>
+ getWeightByDay(date:String):Double
+ loadWeightCalorieData(date:String): List<String>
+ logFood(date:String, name:String, servings:String):String
+ logExercise(date:String, name:String, duration:String):String
+ logWeight(date:String, weight:String):String
+ logCalorieLimit(date:String, calorie:String):String
+ removeFoodLog(date:String, name:String, servings:String):String
+ removeExerciseLog(date:String, name:String, duration:String):String
FoodController
- fFactory:FoodFactory
- fCollection:FoodCollection
+ FoodController(f:FoodCollection)
+ getFoods():List<IFood>
+ getFoodByName(name:String):IFood
+ addBasicFood(name:String, calories:String, fat:String, carb:String, protein:String, overwrite:Boolean):String
+ addRecipe(name:String, ingredients:String, overwrite:Boolean):String
ExerciseController
- exerciseCollection:ExerciseCollection
+ ExerciseController(e:ExerciseCollection)
+ getAllExercises():List<Exercise>
+ getExercise(name:String):Exercise
+ addExercise(name:String, caloriesPerHour:String, overwrite:Boolean):String
Model
NutritionTrackerController
- fController:FoodController
- lController:LogController
+ NutritionTrackerController(f:FoodCollection, l:LogCollection)
+ loadLogData(date:String):List<String>
IOHandler
+ FOOD_CSV:String
+ EXERCISE_CSV:String
+ LOG_CSV:String
+ IOHandler(f:FoodCollection, l:LogCollection, e:ExerciseCollection)
+ read():void
+ write():void

Sequence Diagrams

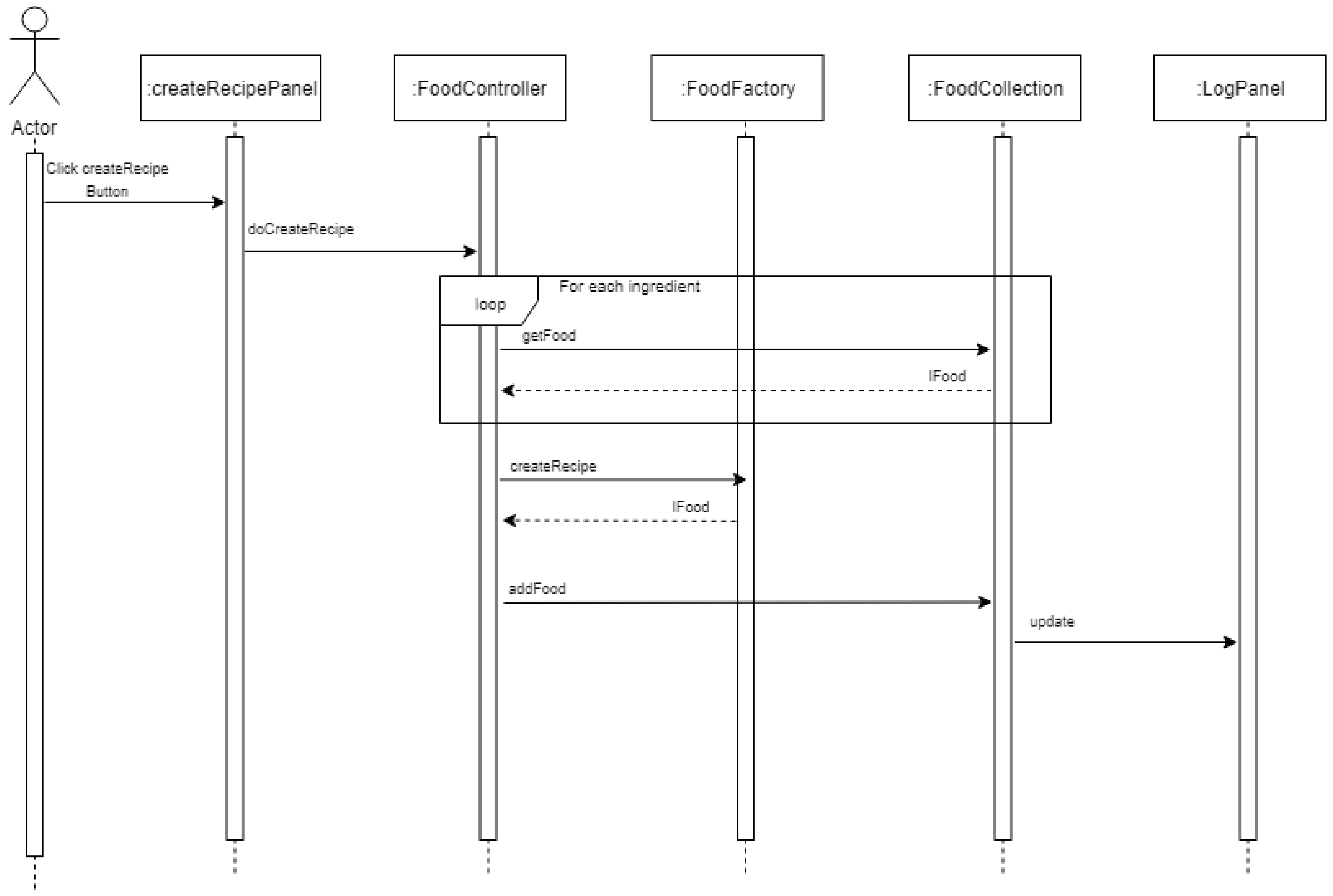
Reading from log.csv



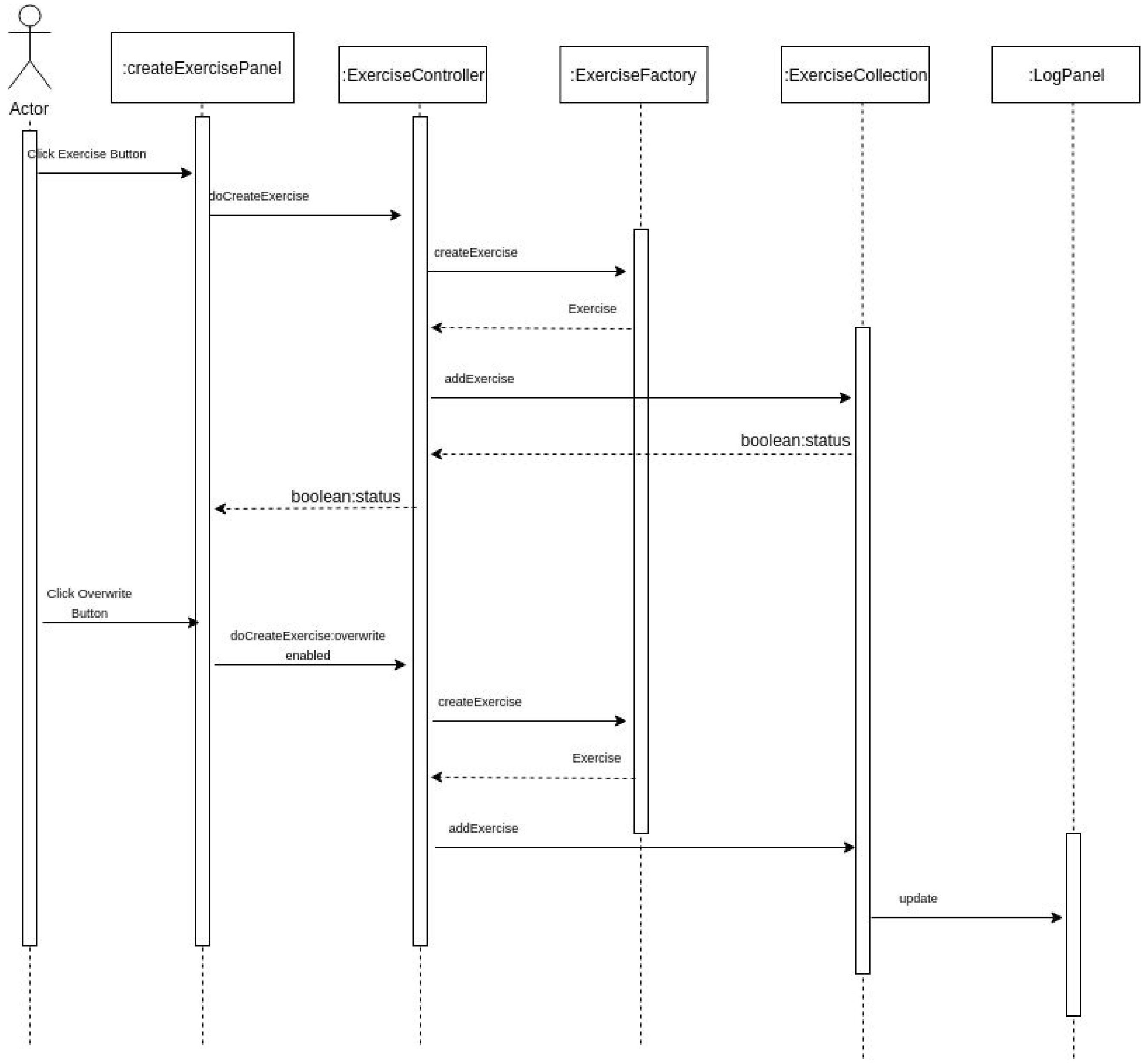
Writing to log.csv



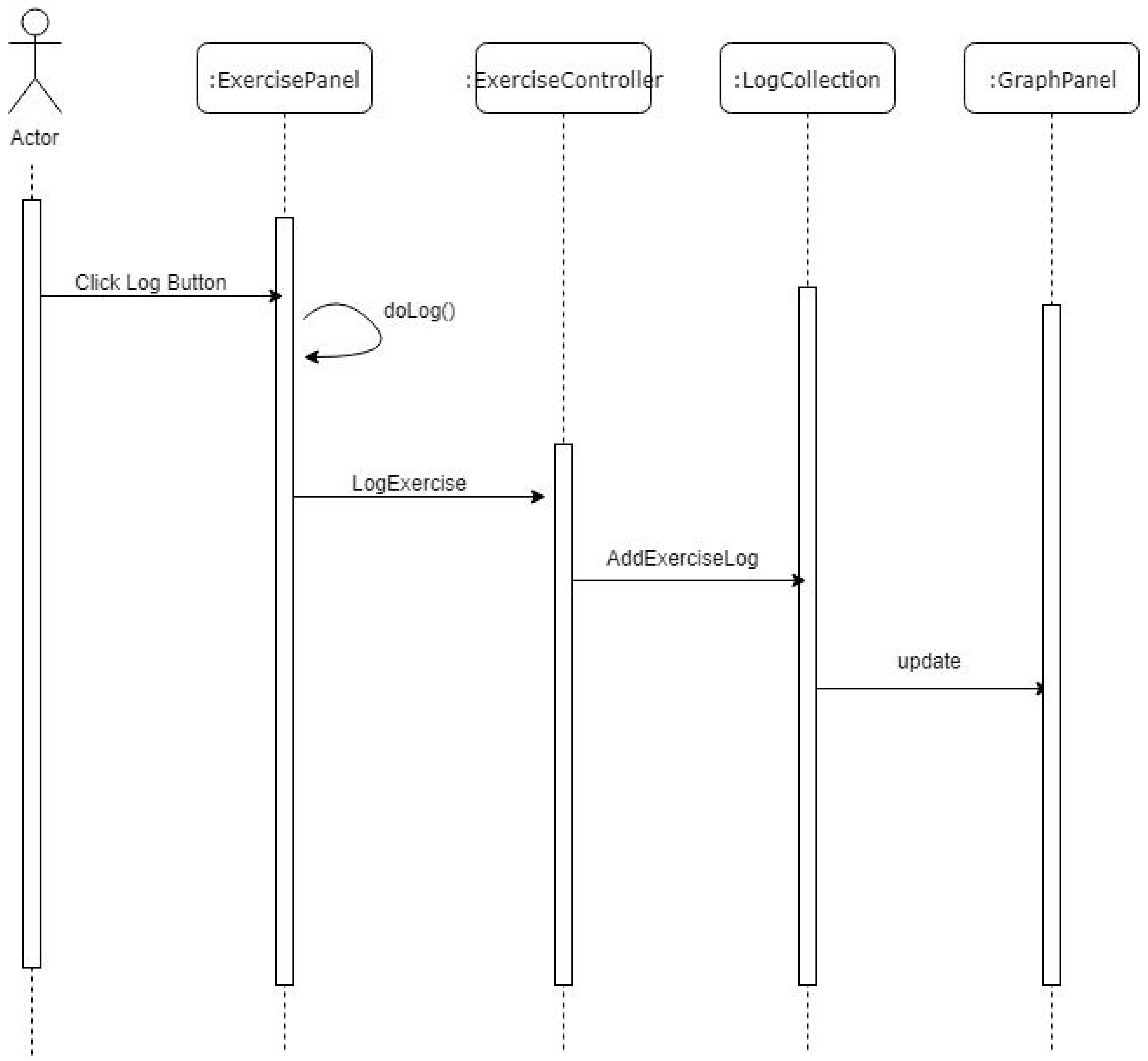
Creating a new recipe



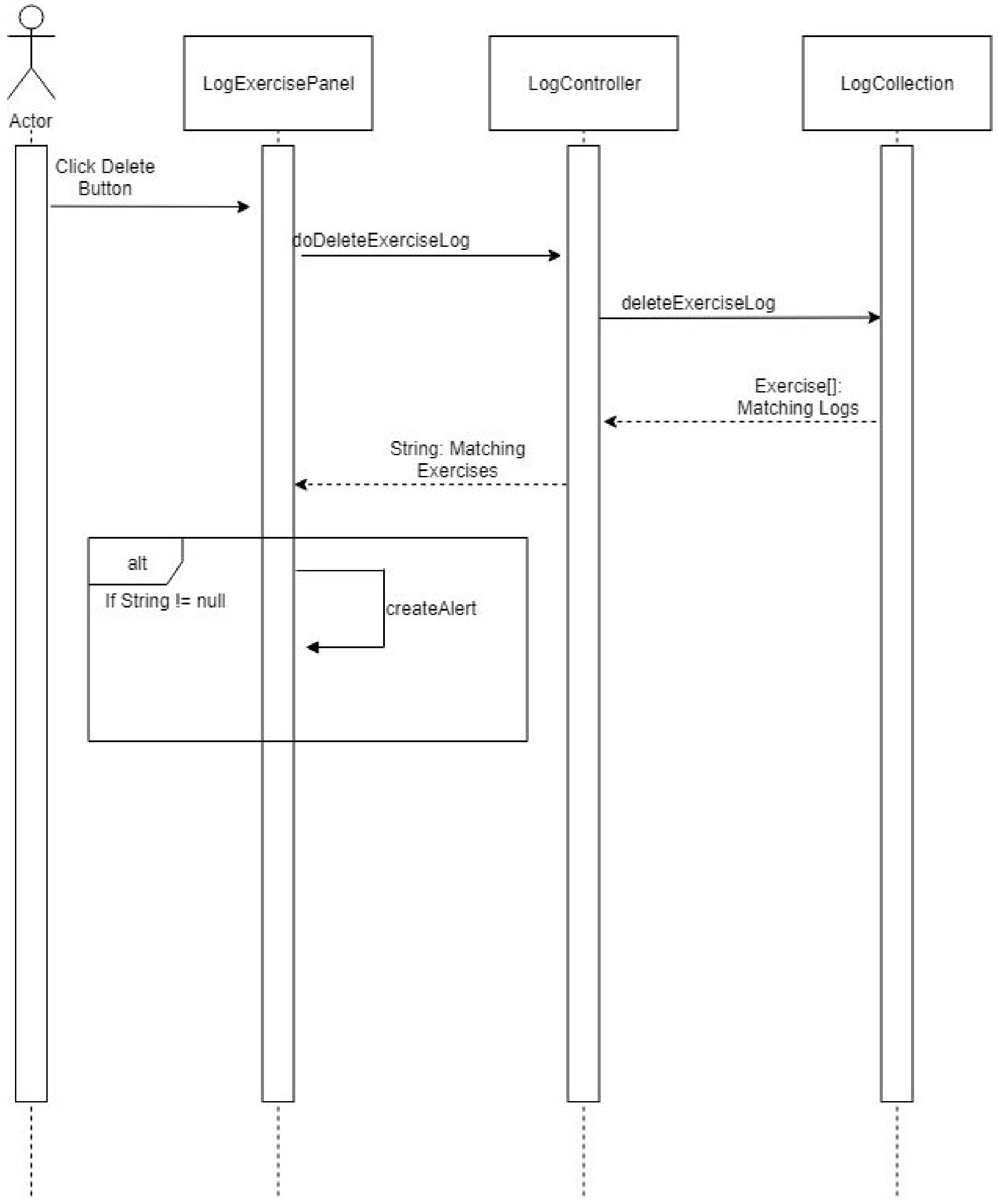
Creating an Exercise



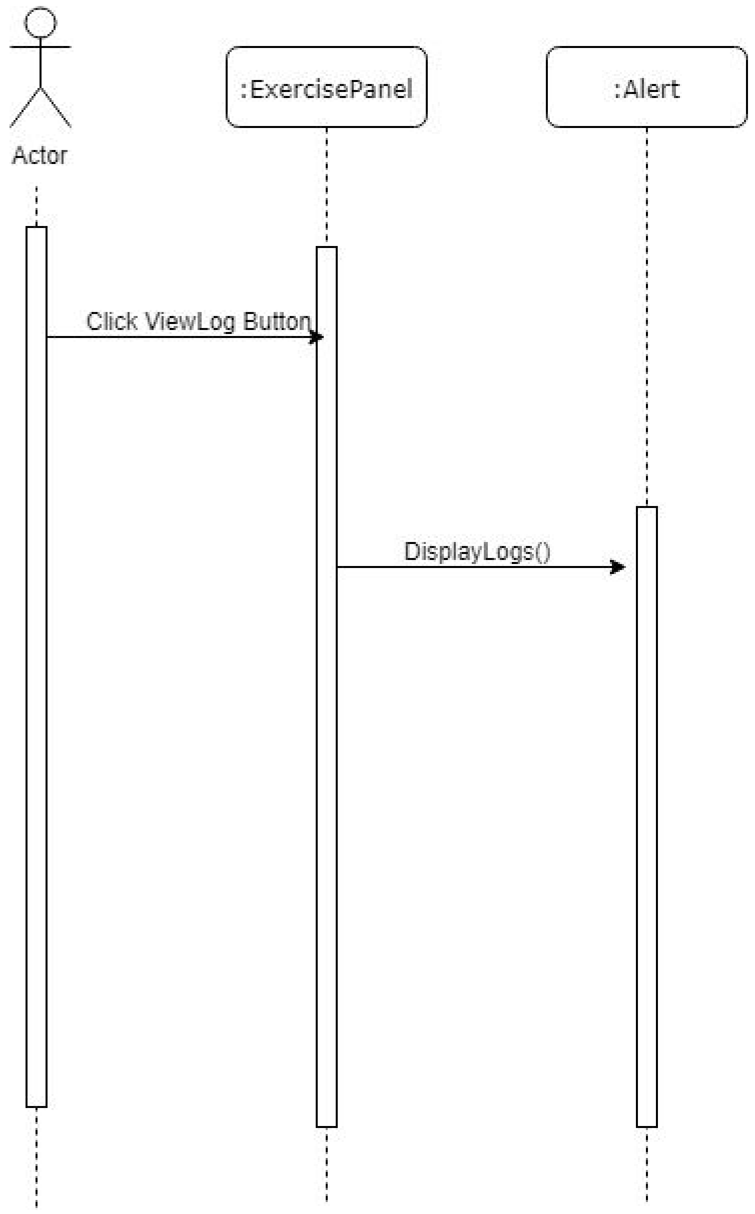
Logging an Exercise



Delete an Exercise Log



Viewing all Logged Exercises



Pattern Usage

Composite Pattern	
Abstract	IFood
Composite	Recipe
Leaf	Food

Observer Pattern	
Observer(s)	NutritionTrackerPanel
Observable(s)	LogCollection

Observer Pattern	
Observer(s)	LogFoodPanel
Observable(s)	FoodCollection

Observer Pattern	
Observer(s)	LogExercisePanel
Observable(s)	ExerciseCollection

Factory Pattern	
Factory	FoodFactory
Output(s)	IFood