PROJECT / RELEASE

Project Design Document

Royal Dabblers
Alex Leute <acl2809@rit.edu>
Erich Snell <eds5997@rit.edu>
Griffin Moose <gsm1298@rit.edu>
Julia Sarun <jns2613@rit.edu>
Edward<ess6446@rit.edu>

Project Summary

This project is a wellness management program that lets users leave daily logs of what they are eating. The program will have some basic foods available, and those basic foods can also be combined to create recipes. Users can also create their own recipes and foods. These recipes and foods will store nutritional information, such as calories and carbs.

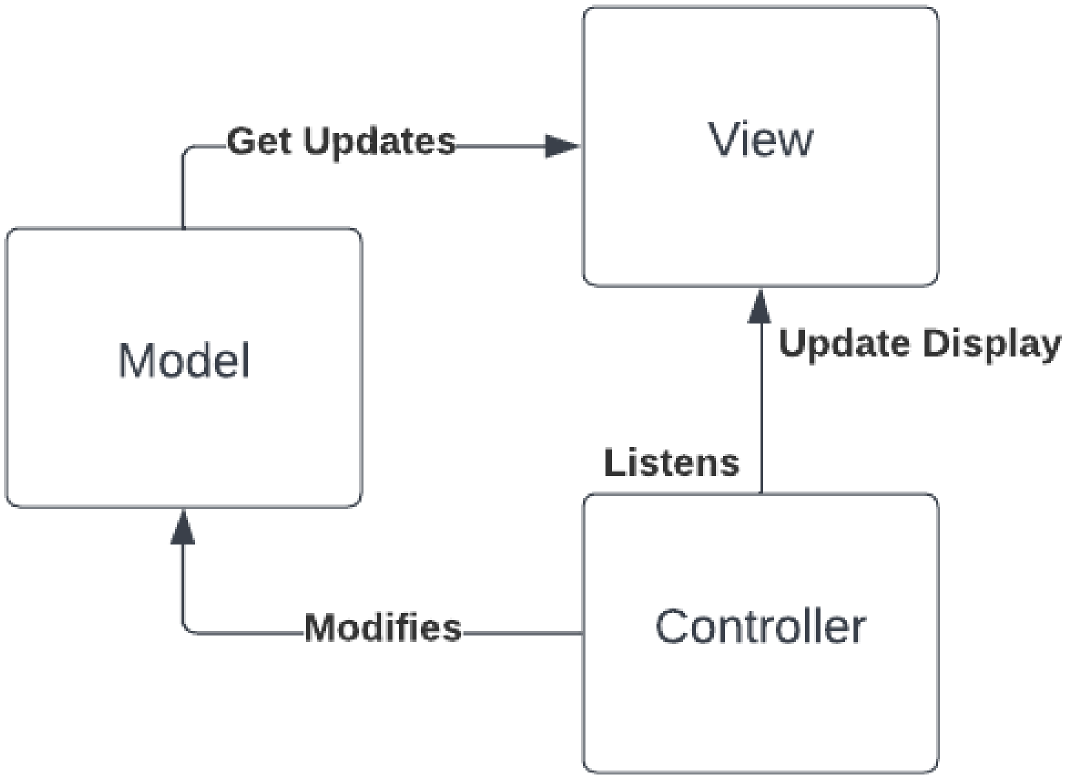
After users log what they are eating throughout the day, the software will help users track their intake and how well they are following their goals. It will also display nutrients per day, and graph distributions of what nutrients they are consuming daily. This will help users track their weight over time.

Design Overview

Before we did any design sketches, subsystem diagrams, or UML, we sat down and combed through the assignment document and recorded all of the classes we believed we would need to complete the task. In brainstorming these classes, we consider that we wanted to limit coupling as much as possible while promoting high levels of cohesion. Furthermore, after completing the list of classes, we compiled another list of potential methods we would use. At this point, we began making rough sketches of a UML class diagram detailing the relationships between all of our different classes.

Using the rough sketches we could make subsystems based on many of the classes and further break them down into a more cohesive and extensible form. The view has an interface that would allow for different UIs to be later added without changing any existing code. We did this under the assumption that a more user friendly UI would need to be implemented later. We used the composite pattern with basic food and recipe with the interface Food to allow easy searching of both the recipes and basic foods. DailyLog has been split into different logs that can each be modified with a controller. Lastly we made data use an interface to allow for different data files to be used later with no changes to the code. We did this under the assumption that users may want different data files to be able to be used in saving and loading. The date is also only linked to the DailLog and Food to reduce the coupling.

Subsystem Structure



Subsystems

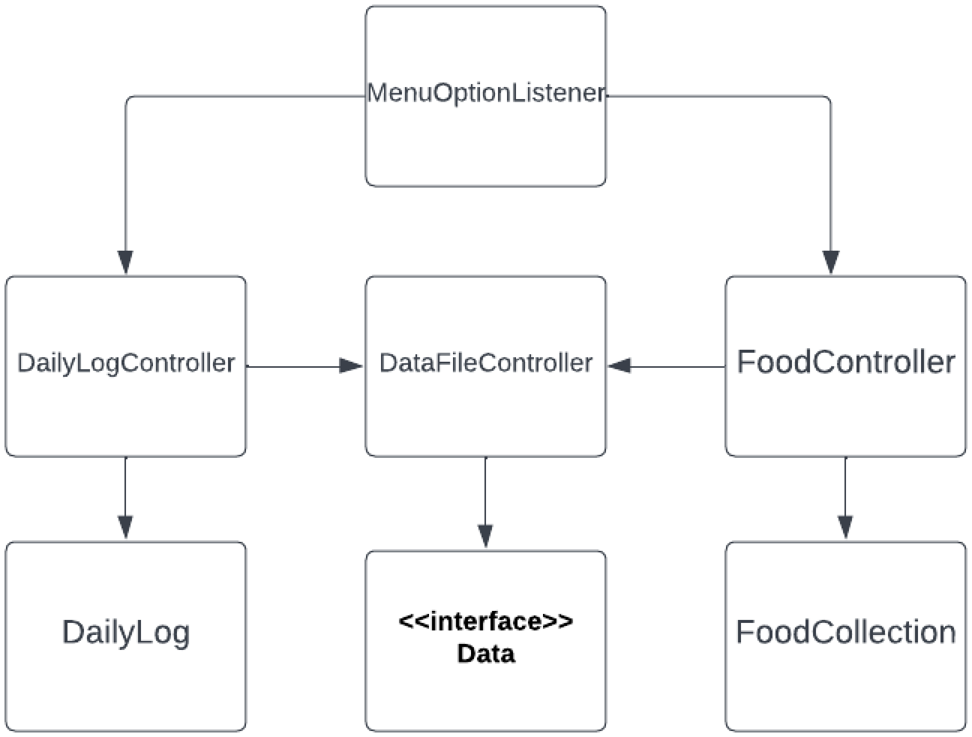
CONTROLLER

Class DailyLogController	
Responsibilities	Called by the MenuOptionListener Give instructions to the dailyLog in memory what to do and reports back to the MenuOptionListener
Collaborators	DailyLog DataFileController

Class DataFileController	
Responsibilities	Called by the FoodController and DailyLogController Give instructions to a Data type and reports back to the MenuOptionListener
Collaborators	Data

Class FoodController	
Responsibilities	Called by the MenuOptionListener Give instructions to the the FoodCollection and reports back to the MenuOptionListener
Collaborators	FoodCollection DataFileController

Class MenuOptionListener	
Responsibilities	Called by the UI Gives instructions to the other Controllers Updates view for prompts Listens for user input
Collaborators	FoodController DailyLogController UI



MODEL

Class Data (interface)	
Responsibilities	Provide a generic interface to all classes that interact with a data file. Can receive a formatted data file (Load). Can write a formatted data file (Save).

Class CSV	
Responsibilities	Can receive a formatted CSV file (Load). Can write a formatted CSV file (Save).
Collaborators (inheritance)	Data

Class FoodCollection	
Responsibilities	Support access to the foods or recipes in the food collection. Add, find, delete an existing food or recipe in the collection. Save/load to and from a data file
Collaborators	BasicFood

	Recipe

Class Food (interface)	
Responsibilities	Provide a generic interface to all food and recipe types. Includes unique name of food or recipe.

Class BasicFood	
Responsibilities	Represents a basic food in the collection. Provides access to calories, fat, carb, and protein in a serving of the food
Collaborators (inheritance)	Food
Class Recipe	
Responsibilities	Represents a recipe in the collection. Provides access to all basic foods and their servings in the recipe Provides access to the calories, fat, carb, and protein in terms of the foods stated serving amounts in the recipe.
Collaborators (inheritance)	Food

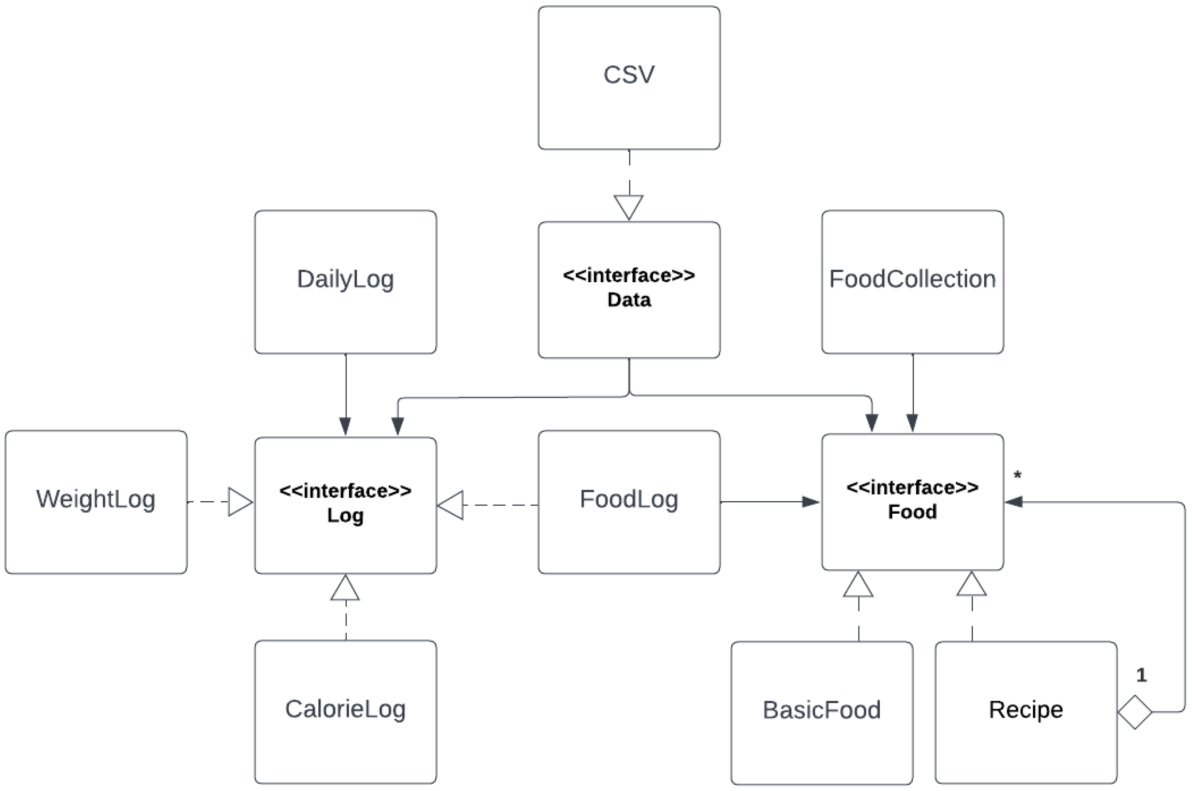
Class DailyLog	
Responsibilities	Represents a daily log Provides access to calorie limit, foods eaten, and servings consumed on a date. Save/load to and from a data file
Collaborators	WeightLog CalorieLog FoodLog

Class Log (interface)	
Responsibilities	Provides generic interface for log types. Allows for adding and getting date and value.

Class FoodLog	
Responsibilities	Represents a FoodLogin the collection. Access to get and set food intake and date
Collaborators (inheritance)	Log

Class WeightLog	
Responsibilities	Represents a WeightLog in the collection. Access to get and set weight and date
Collaborators (inheritance)	Log

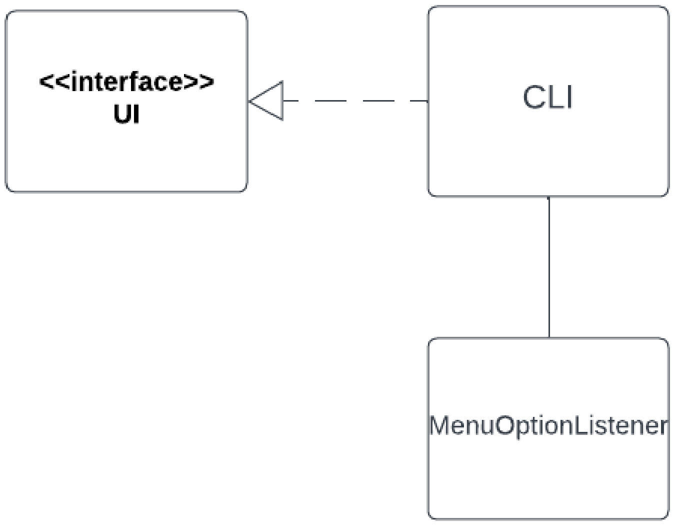
Class CalorieLog	
Responsibilities	Represents a CalorieLog in the collection. Access to get and set calorie limit and date
Collaborators (inheritance)	Log



VIEW

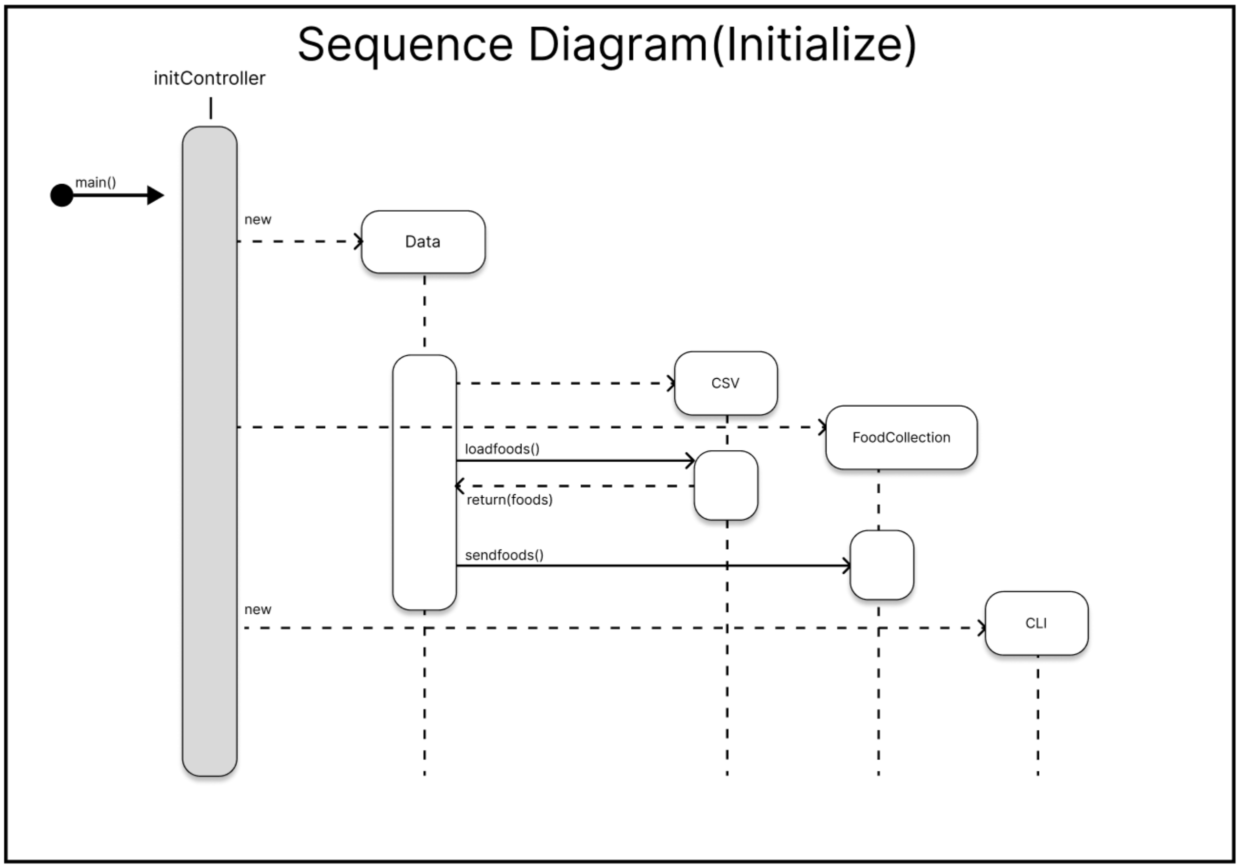
Class UI (interface)	
Responsibilities	Provide a generic interface to all UI types. Allows users to add, find, delete an existing food or recipe in the collection using the UI. Save/load to and from a data file using the UI.

Class CLI	
Responsibilities	Represents a basic CLI UI. Allows users to add, find, delete an existing food or recipe in the collection using commands. Save/load to and from a data file using commands.
Collaborators	UI MenuOptionListener

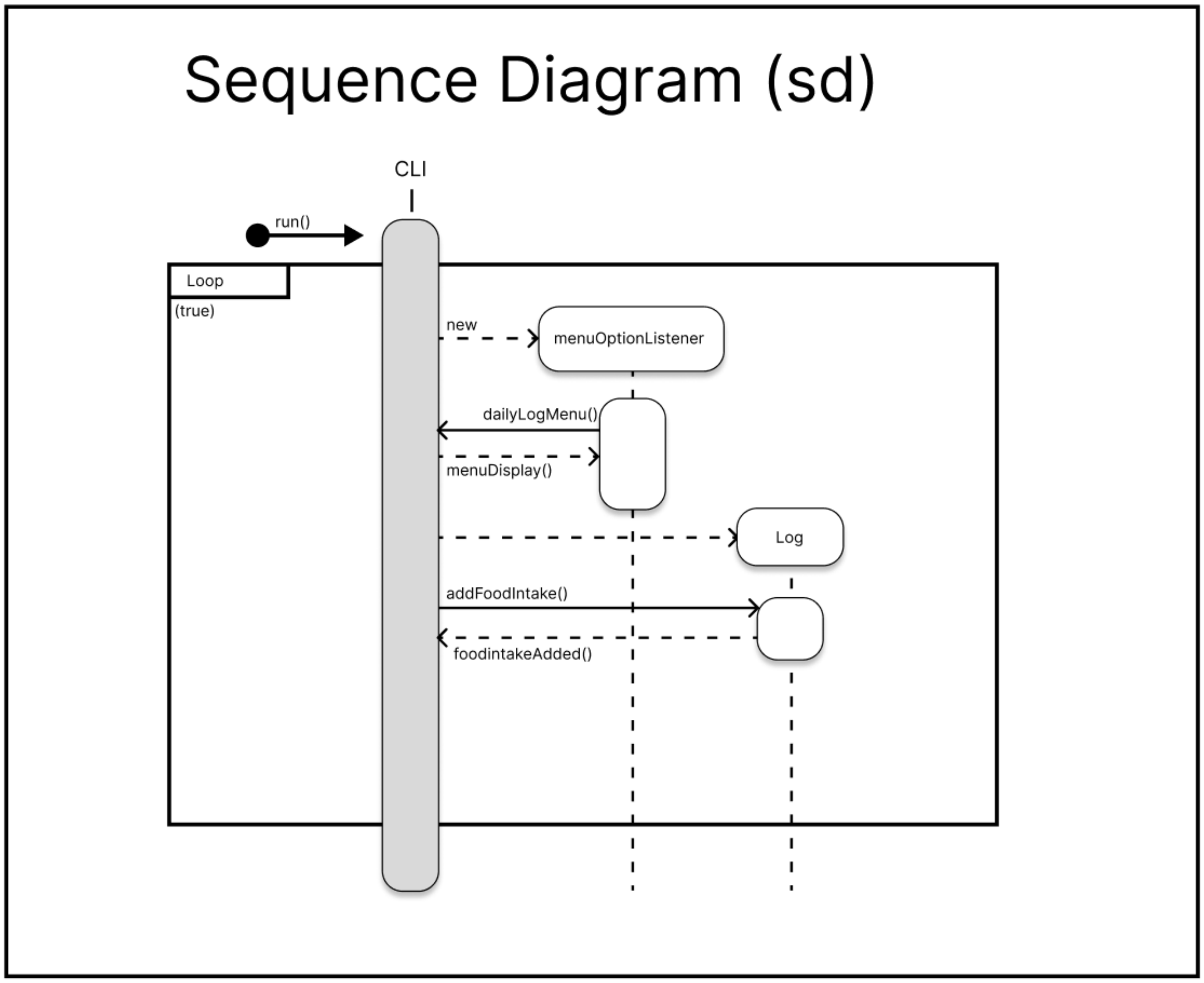


Sequence Diagrams

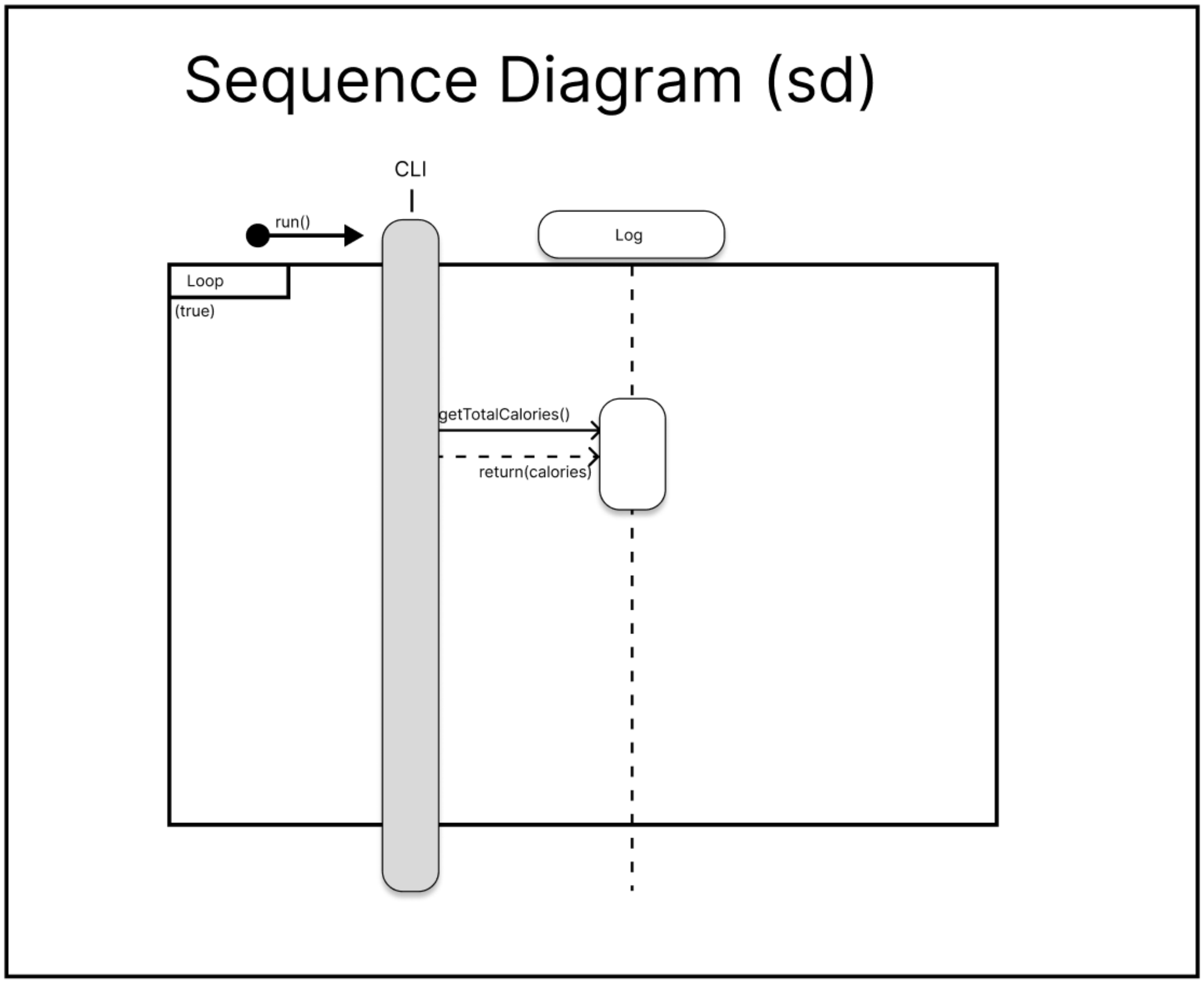
This diagrams shows the initialization of the initController and the creation of the CLI. To start, the intiController creates and instance of the Data class as a new CSV. Then the data object calls the method loadFoods() to the CSV, which returns the foods in the CSV file. After this is done, the initController creates a new FoodCollection instance and passes the loadedfoods into its parameters. Lastly, the intiController creates a new CLI and calls the mainMenu() method to display the menu options in the command line interface for the user.



This next sequence diagram (the sample) illustrates the process of adding a new food intake to the dailyLog. The first thing that happens is the CLI creates a new menuOptionListener that listens to the user input and displays the corresponding menu. In this example, the user selects "dailyLog options" at which point the menuOptionListener calls for the CLI to display the dailyLogMenu. At this point, the CLI also creates a new instance of the Data class to call the addFoodIntake() method on. After this, the users input for the food is added to the daily log.



This sequence diagram is a simple illustration of getting the total calorie count from the days log. The CLI calls the getTotalCalories() method and the Log returns the total for ALL of the foods that were added on the current day.



Pattern Usage

Composite Pattern

Composite Pattern	
Component	Food
Leaf	BasicFood
Composite	Recipe

MVC Pattern

MVC Pattern	
Model	BasicFood CSV DailyLog Data FoodCollection Log Recipe
View	CLI
Controller	DailyLogController DataFileController FoodController MenuOptionListener

RATIONALE

We decided to use the MVC pattern to decrease coupling, and to separate the concerns from the UI and the Model.

We are using the composite pattern for Food because Recipes are made up of BasicFoods, but have all of the same properties as BasicFoods