HealthNCare App 2.0

Project Design Document

TEAM B - Java Juggernauts
Nathaniel Pellegrino nsp7786@rit.edu
Riley Basile-Benson rdb6619@rit.edu
Christian Berko ceb1810@rit.edu
Tryder Kulbacki thk9885@rit.edu

https://docs.google.com/document/d/16sufTzzHB_NhflvxKsnsOAVHUiXl_8TFUnm2thgDEo4/edit?usp=sharing

1 Project Summary

This project allows the user to track their food intake, exercises, and weight over time. This allows the user and their doctor to see a historical view of their health over time.

The app allows the user to add their own basic foods with nutritional info, recipes using those basic foods, and exercises. Using the nutritional info from the food and time spent exercising, the app can calculate the user's total and net calories for the day, along with an overview of other nutrients consumed throughout the day.

The user can set their calorie limit for the day, and be alerted if they exceed the limit. They can also track their weight, and see the correlation between calories, exercise, and their weight over time.

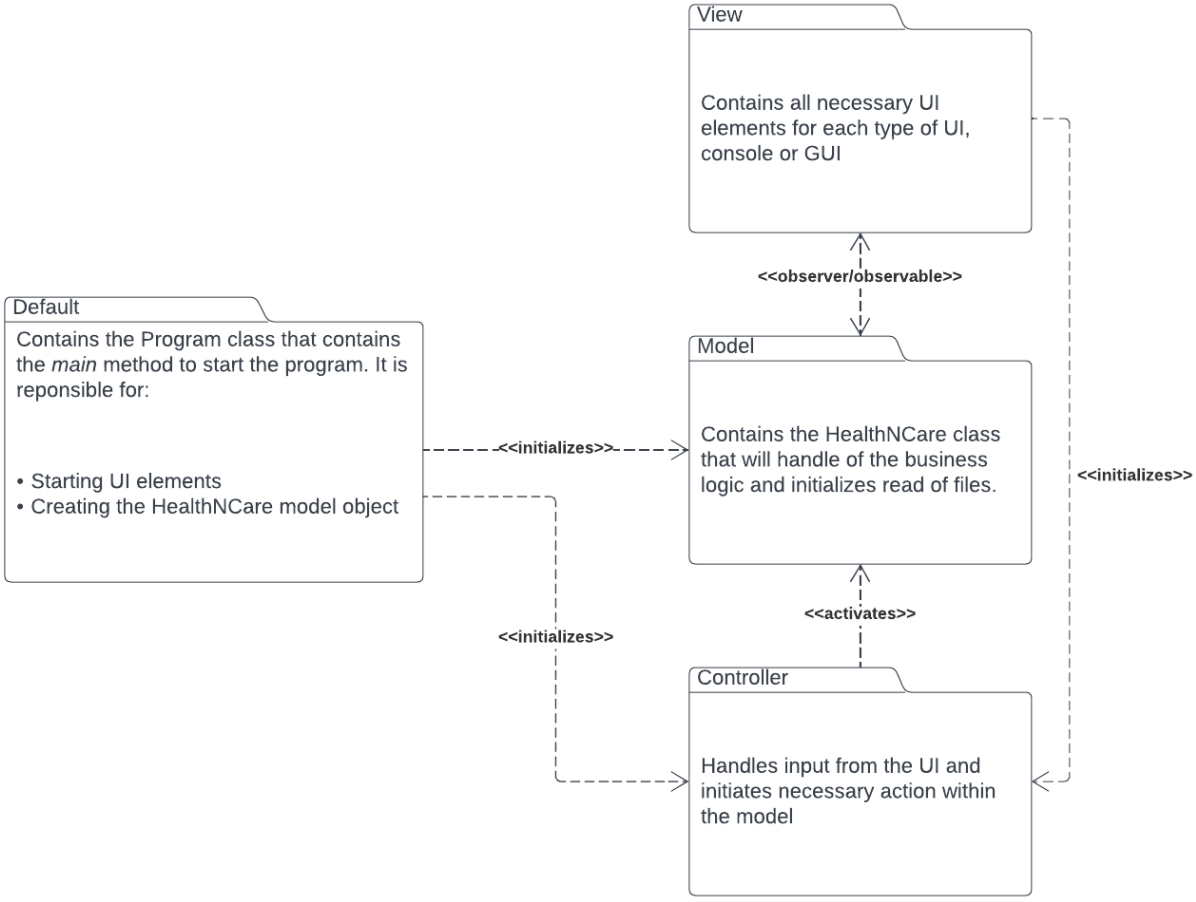
2 Design Overview

This app will follow the Model-View-Controller architectural pattern. It includes the Composite and Observer design patterns.

View

- The main window is broken into separate classes to simplify updating each part as data changes.

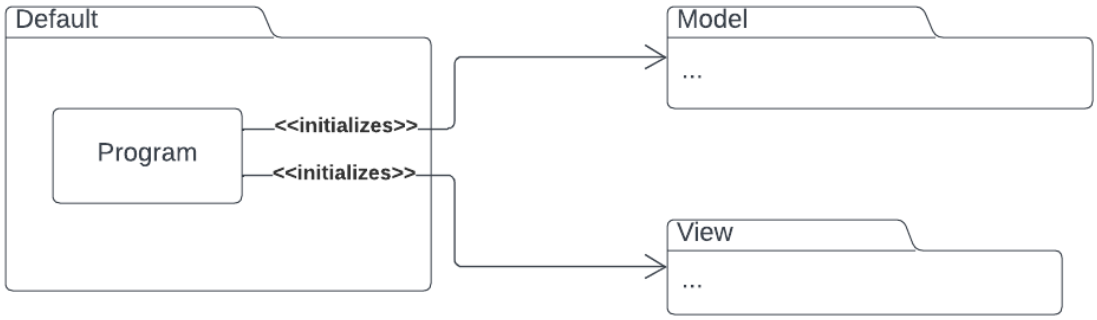
3 Subsystem Structure



4 Subsystems

4.1 Default Subsystem

Class Program	
Responsibilities	Create and manage the application's core objects and data models. Create the TextUI and, eventually GUI. (User Interface) Display the user interface(s) to enable interaction with the application Handle inputs and execute appropriate application logic.
Collaborators (uses)	**model.HealthNCare** - starts main business logic class **view.UserInterface** - starts one or more user interfaces



4.2 Model Subsystem

Class HealthNCare	
Responsibilities	● Notify observers of any changes ● Handle reading and writing to main data structures ● Uses business logic for interacting with the data
Collaborators (inherits)	**java.util.Observable** - so the program changes can be observed by others
Collaborators	**model.StorageManager** - interacts with the

(uses)	storage media and allows HealthNCare to write out or read the log and foods **java.util.Observer** - for notifications of log inputs to views. **java.util.HashMap** - stores the available foods **java.util.ArrayList** - stores the LogEntries

Class StorageManager	
Responsibilities	Interact with the storage media to read and write the data
Collaborators (uses)	**model.IFood** - Gets changes from the recipes and food selected. **model.ILog** - handles the log data **model.IRecipeLog** - handles the food data **java.util.HashMap** - stores the available foods **java.util.ArrayList** - stores the LogEntries

Class LogCSVFile	
Responsibilities	Writes the current log back to the CSV file. It will completely overwrite the existing file and replaces it with what the given object
Collaborators (inherits)	**ILog** interface for all log storage media
Collaborators (uses)	**org.apache.commons.csv.CSVReader** - reads in and parses a CSV file **org.apache.commons.csv.CSVReader** - writes out a CSV file **model.LogEntry** - contains information about each log entry

Class RecipeCSVFile

Responsibilities	Concrete implementation of link IRecipeLog for using a CSV file
Collaborators (uses)	**org.apache.commons.csv.CSVReader** - reads in and parses a CSV file **org.apache.commons.csv.CSVReader** - writes out a CSV file **model.IFood** - Gets a list of available recipes, food, and their corresponding nutrition values. **model.FoodItem** - holds nutritional information about a food **model.Recipe** - holds information about a recipe
Collaborators (inherits)	**model.IRecipeLog** - common interface for recipe storage media

Class ExerciseCSVFile	
Responsibilities	Concrete implementation of link IRecipeLog for using a CSV file
Collaborators (uses)	**org.apache.commons.csv.CSVReader** - reads in and parses a CSV file **org.apache.commons.csv.CSVReader** - writes out a CSV file **Model.exercise -** the exercise objects
Collaborators (inherits)	**model.IExerciseStorage** - common interface for all exercise storage types

Interface IFood	
Responsibilities	Common interface for all food items. Whether the item is a basic food item or a recipe it should be able to get the nutritional information from it.

Class Recipe	
Responsibilities	A collection of food and other recipes.
Collaborators (inherits)	**model.IFood** - component used to structure the collection
Collaborators (uses)	**java.util.ArrayList -** ArrayList to hold the food

Class FoodItem	
Responsibilities	An individual item of food that can be added to a recipe.
Collaborators (inherits)	**model.IFood** - component used to structure the object

Class Exercise	
Responsibilities	Tracks an individual type of exercise with a measure of calories per hour

Class CalorieLimitEntry	
Responsibilities	A concrete implementation of a LogEntry that tracks a person's calorie limit at a particular point in time
Collaborators (inherits)	LogEntry
Collaborators (uses)	**java.time.LocalDate -** Used to create a date Object

Class ConsumptionEntry	
Responsibilities	Concrete implementation of link LogEntry

	that logs a food consumed and a specific quantity
Collaborators (inherits)	LogEntry
Collaborators (uses)	**java.time.LocalDate -** date of the consumption entry

Class ILog	
Responsibilities	Common interface for any log storage medium. It should be able to read all the log entries and write them back out
Collaborators (uses)	**java.util.ArrayList -** stores a list of Log Entry

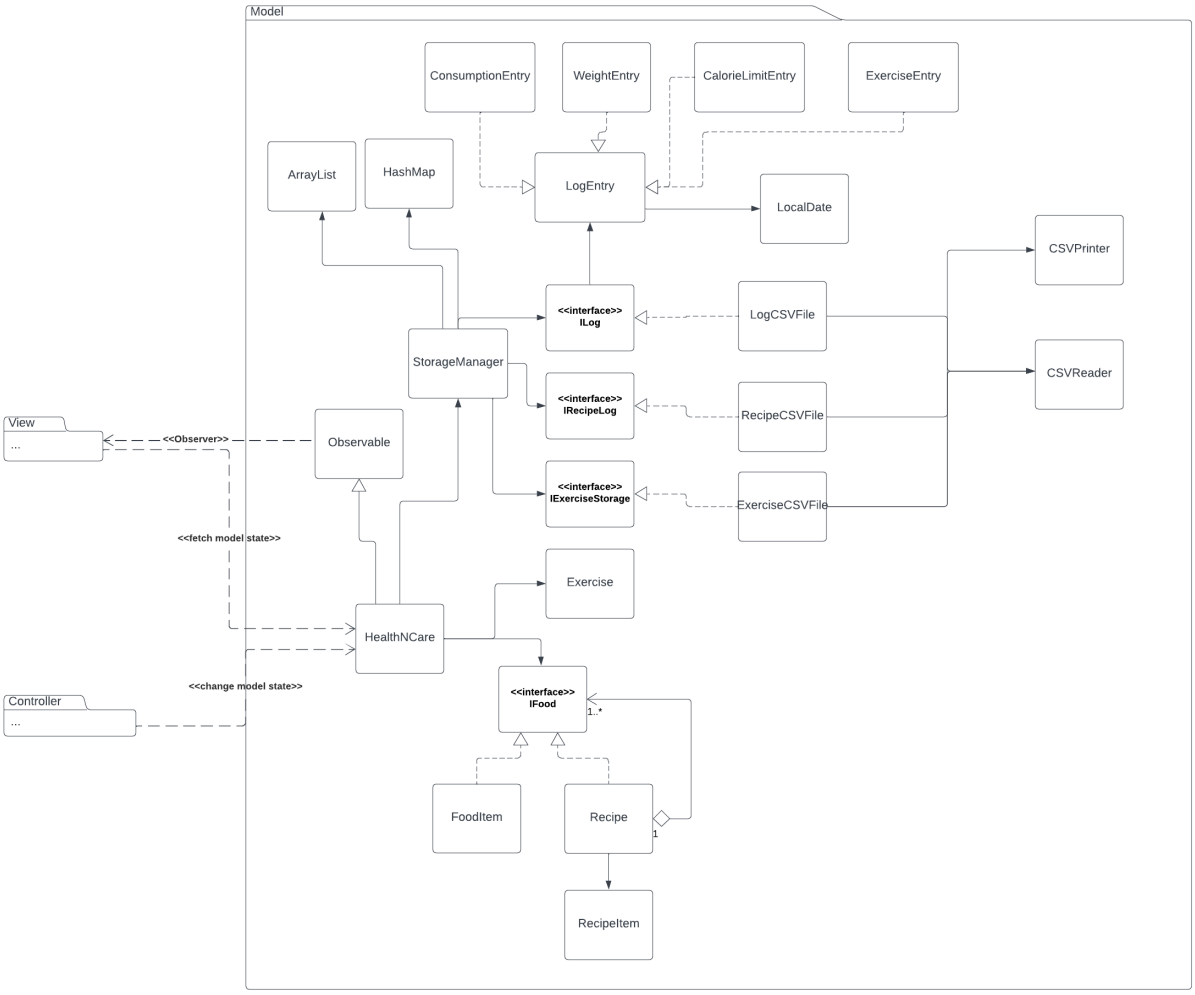
Class IRecipeLog	
Responsibilities	Interface definition for any Recipe storage object. It should be able to read all the items and write the items back out to the storage medium
Collaborators (inherits)	**java.util.HashMap -** holds recipes

Class IExerciseStorage	
Responsibilities	Interface definition for exercise storage medium
Collaborators (inherits)	**java.util.HashMap -** collection of exercises

Class StorageManager	
Responsibilities	Manages the storage of the log and recipes
Collaborators (uses)	**model.ILog** - Handles log entries **model.IRecipeLog** - Handles food entries **java.util.ArrayList** - holds log entries **java.util.HashMap** - holds food entries

Class WeightEntry	
Responsibilities	Manages the storage of the log and recipes
Collaborators (uses)	**LogEntry** - interface for all log entries
Collaborators (inherits)	**java.time.LocalDate -** date of the consumption entry

Interface LogEntry	
Responsibilities	Common interface for each type of log entry
Collaborators (uses)	**java.time.LocalDate -** date of the consumption entry



4.3 View Subsystem

Class TextUI	
Responsibilities	Handles the command line interface which takes in the user input and displays the data
Collaborators(inherits)	**java.util.Observable -** so that changes in the UI can be observed by other classes **java.util.Scanner** - accepts input from the terminal **java.lang.System** - prints output to a terminal window

Class SwingGUI	
Responsibilities	Creates the graphical user interface which takes in the user input and displays the data
Collaborators(inherits)	**model.Observable -** When the date changes it can be observed by other classes. **view.UserInterface** - Interface that breaks out required functions of a User Interface class. **view.Observer** - Updates some UI components if the model changes.

Class SetLimits	
Responsibilities	Handles the user input to set their calorie and weight limits for the selected date
Collaborators(inherits)	

Class SettingsDialog	
Responsibilities	Handles the panel that allows users to add foods and recipes, as well as set their calorie/weight value for exercise calculations.
Collaborators(inherits)	**view.Observer -** so that any changes in the UI can be observed and used to refresh the content

Class FoodConsumedTable	
Responsibilities	Handles the tables that display the food eaten on the selected date

Collaborators(inherits)	**view.Observer -** so that any changes in the UI can be observed and used to refresh the content

Class ExerciseTable	
Responsibilities	Handles the table that displays the exercise done on the selected date.
Collaborators(inherits)	**view.Observer -** so that any changes in the UI can be observed and used to refresh the content

Class CalorieCalculator	
Responsibilities	Displays the calorie calculations for the selected date, along with the weight.
Collaborators(inherits)	**view.Observer -** so that any changes in the UI can be observed and used to refresh the content

Class NutritionBarChart	
Responsibilities	Displays a bar chart that shows the nutrition values consumed for the selected date.
Collaborators(inherits)	**view.Observer -** so that any changes in the UI can be observed and used to refresh the content

Class BarCanvas	
Responsibilities	Draws the bar of the Bar Chart

Collaborators(uses)	**java.awt.graphics -** draws the canvas of the Bar Chart

Class BasicFoodList	
Responsibilities	Presents the list of basic foods available to modify
Collaborators(uses)	**JPanel -** displays a panel of available foods in a separate window.

Class DailyLogTrees	
Responsibilities	Displays the food currently in the log
Collaborators(uses)	**JTree -** displays the foods in a list-like structure **view.Observer -** so that any changes in the UI can be observed and used to refresh the content

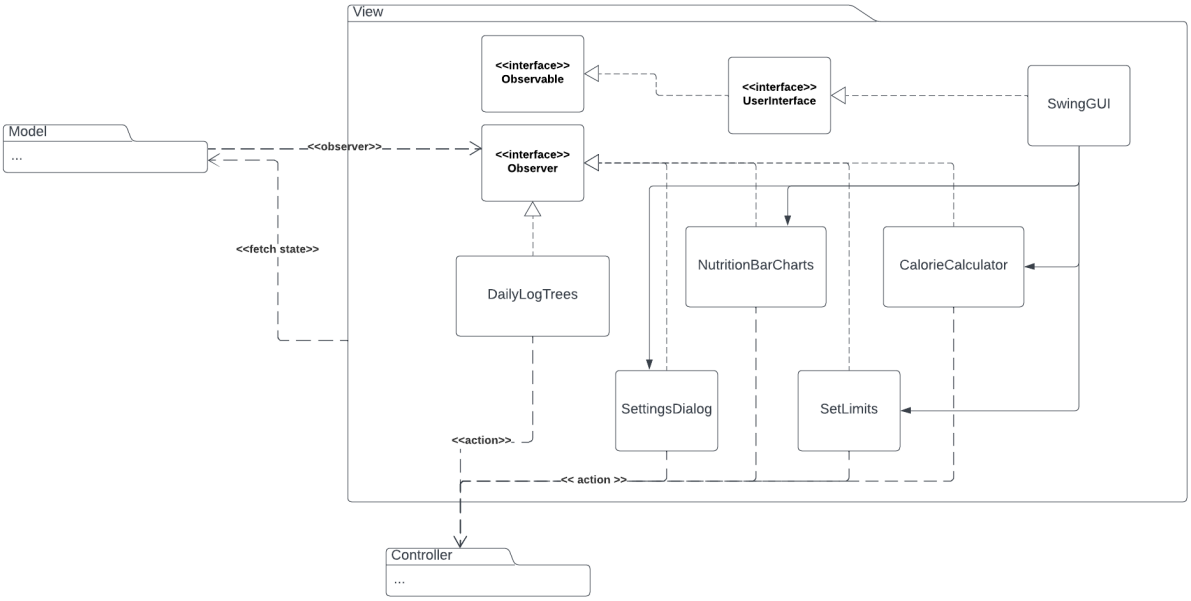
Class DarkModeManager	
Responsibilities	Toggles Dark Mode of the UI
Collaborators(uses)	N/A

Class ExerciseList	
Responsibilities	Presents the list of exercises available to modify
Collaborators(uses)	N/A

Class GridBagConstraintDefaults	
Responsibilities	Sets the defaults for UI GridBagLayout constraints across the app
Collaborators(uses)	**java.awt.GridbagConstraints** **java.awt.Insets**

Class RecipeList	
Responsibilities	Displays the available recipes for modification
Collaborators(uses)	**javax.swing.tree.DefaultMutableTreeNode** **javax.swing.tree.DefaultTreeModel**

Class UpdateDailyLogDialog	
Responsibilities	Creates a button to add food/recipes/exercise to the daily log.
Collaborators(uses)	**javax.swing.tree.DefaultMutableTreeNode** **javax.swing.tree.DefaultTreeModel**



4.4 Controller Subsystem

Class NutritionChartUpdater	
Responsibilities	updates and calculates the percentages
Collaborators(uses)	**model.HealthNCare** **java.time.localDate**

Class AddCalorieLimitListener	
Responsibilities	Responsible for adding new CalorieLimit
Collaborators (uses)	**model.HealthNCare**

Class RemoveEntryListener	
Responsibilities	Responsible for handling the addition of a weight entry.

Collaborators (uses)	**model.HealthNCare** **model.ConsumptionEntry**

Class ItemListResponder	
Responsibilities	Responsible for responding to requests for the list of foods
Collaborators (uses)	**model.HealthNCare** **model.FoodItem** **model.IFood** **model.Recipe**

Class DailyStatsResponder	
Responsibilities	Responds to requests for daily statistics
Collaborators (uses)	**model.HealthNCare**

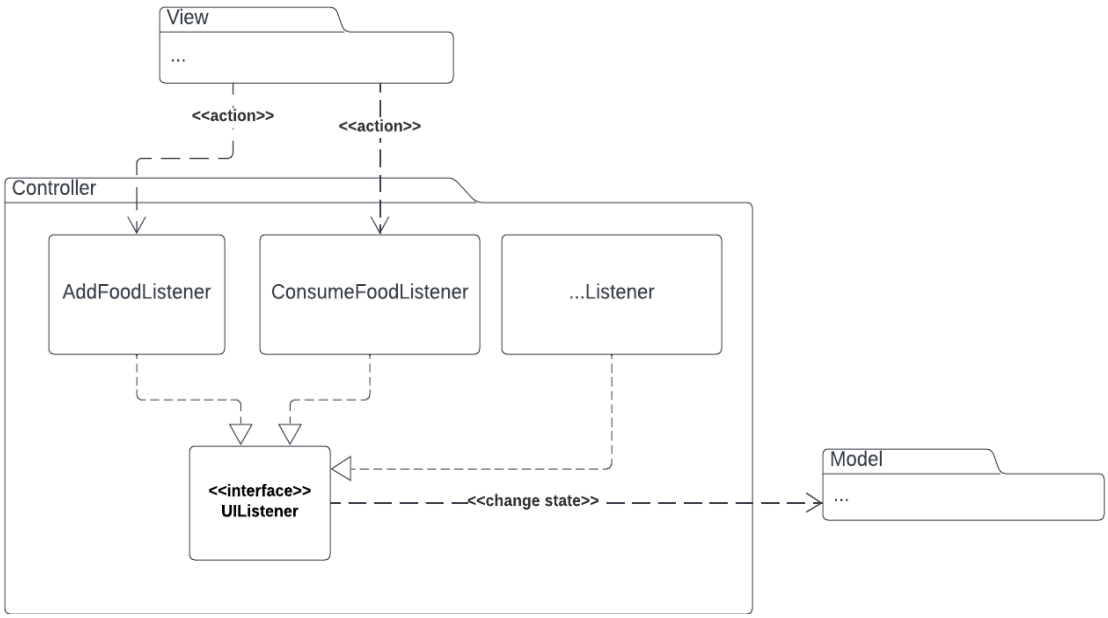
Class RemoveFoodEntryListener	
Responsibilities	Responsible for handling the removal of a food item from the Log Entry
Collaborators (uses)	**model.HealthNCare** **model.Consumption** **model.UserInterface** **java.time.LocalDate**

Class SaveDataListener	
Responsibilities	Responsible for saving the data in the program.
Collaborators (uses)	**model.HealthNCare**

Class CheckCaloriesResponder	
Responsibilities	Used to check the calories consumed, calories expended from exercise, and the calorie limit
Collaborators (uses)	**model.HealthNCare** **java.time.LocalDate**

Class ExerciseCRUDListener	
Responsibilities	Allows for CRUD operations on exercises
Collaborators (uses)	**model.HealthNCare** **view.UserInterface**

Class FoodCRUDListener	
Responsibilities	Allows for CRUD operations on food items
Collaborators (uses)	**model.HealthNCare** **view.UserInterface**



Exception Package

Class ExerciseAlreadyExistsException	
Responsibilities	Raises an exception if the user tries to add an exercise with the same
Collaborators (uses)	

Class ExerciseNotExistsException	
Responsibilities	Throws an error if the user searches for an exercise that does not exist
Collaborators (uses)	

Class FoodAlreadyExistsException	
Responsibilities	Throws an error if the user tries to add a food with the same name to log
Collaborators (uses)	

Class FoodExistsException	
Responsibilities	Throws an error if the user tries to remove a food that does not exist
Collaborators (uses)	

Class HasDependencyException	
Responsibilities	Throws an error if the user tries to remove a food that depends on another item
Collaborators (uses)	

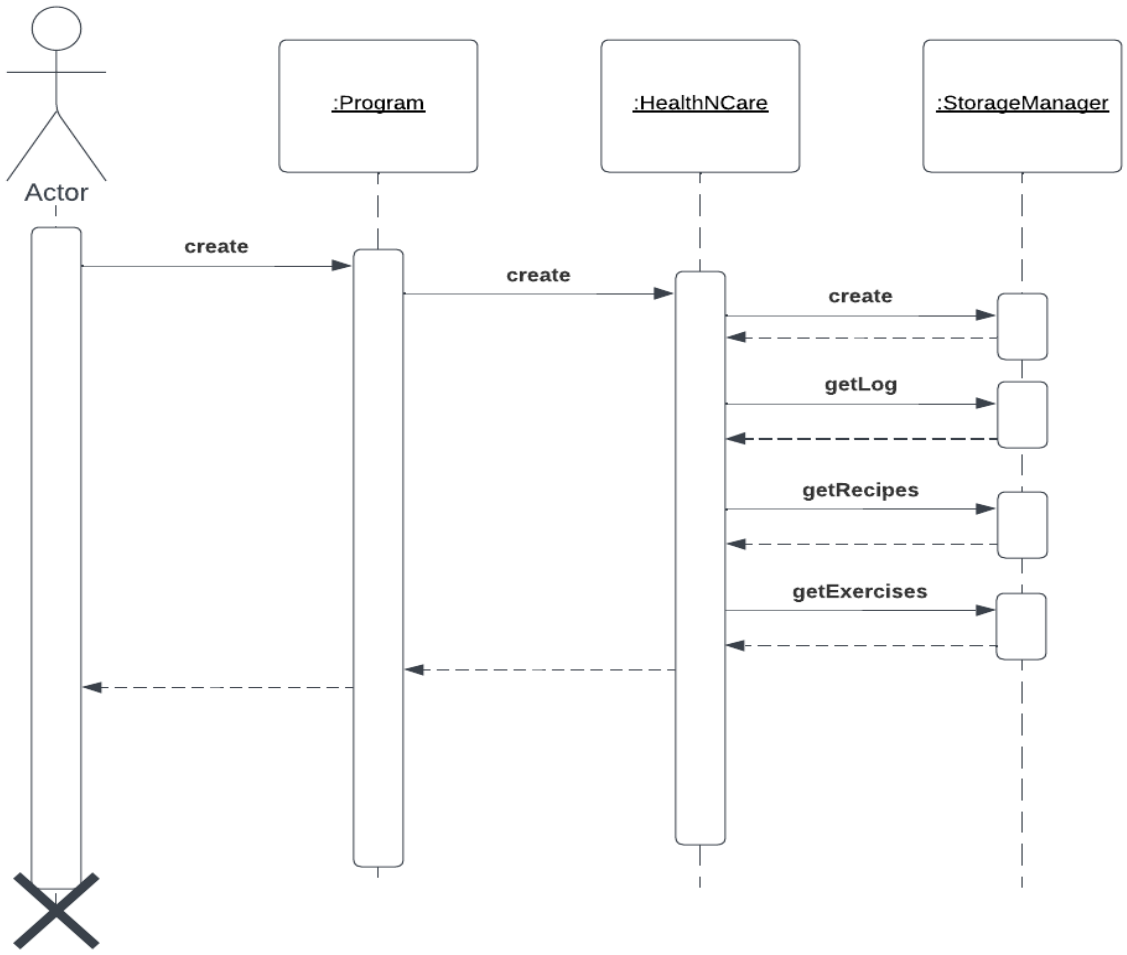
Class ItemUsedInLogException	
Responsibilities	Throws an error if the user tries to delete an item that is used in the log entry
Collaborators (uses)	

Class PastEntryConstraintException	
Responsibilities	Throws an error if the user tries to add an entry for a date that has already past

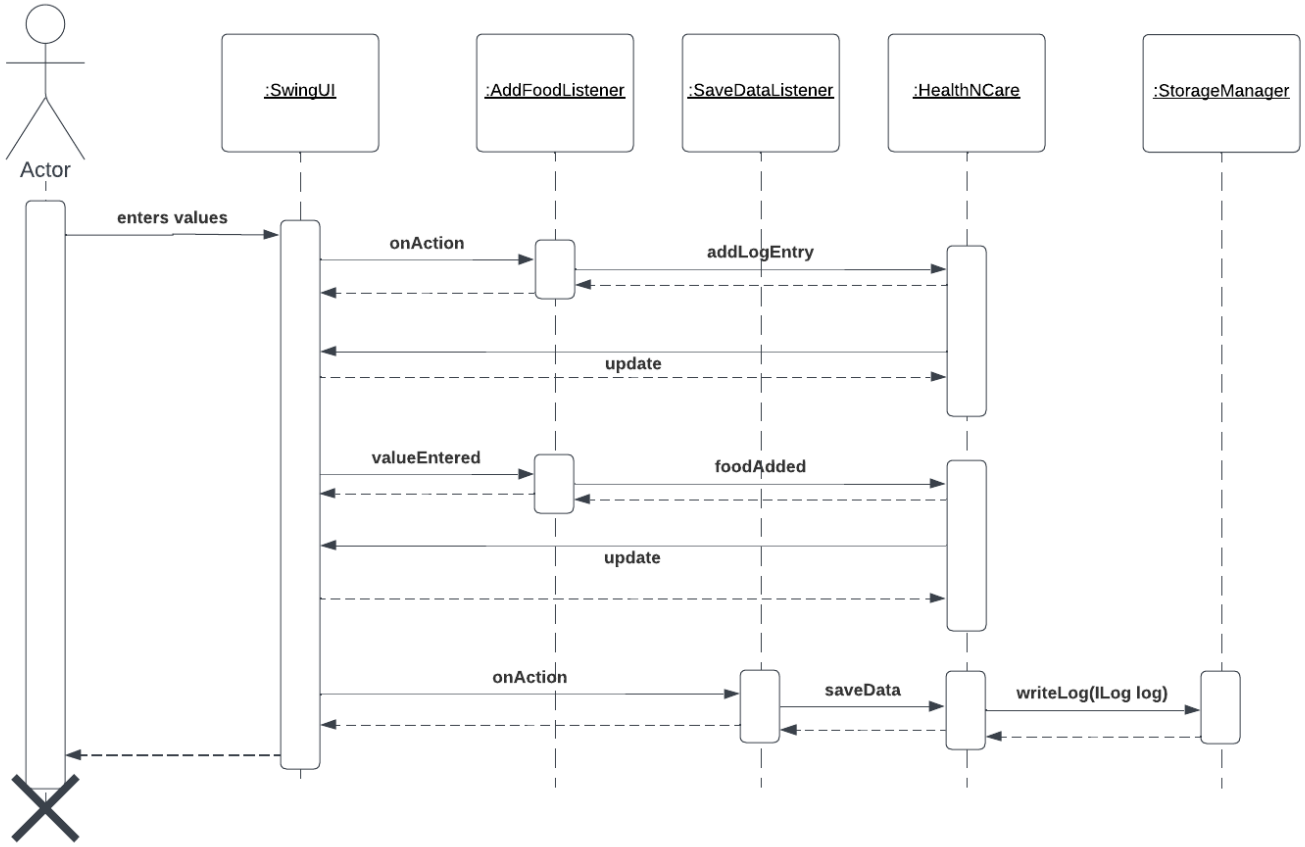
Collaborators (uses)	

5 Sequence Diagrams

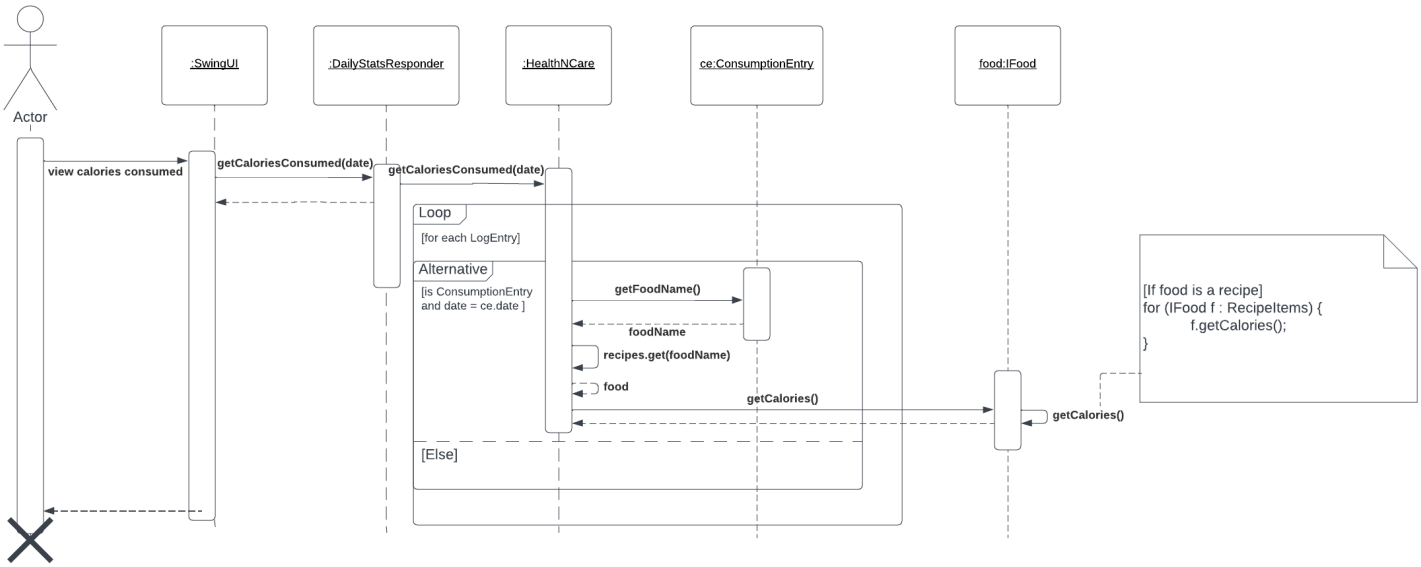
5.1 Initial reading of data from persistent storage



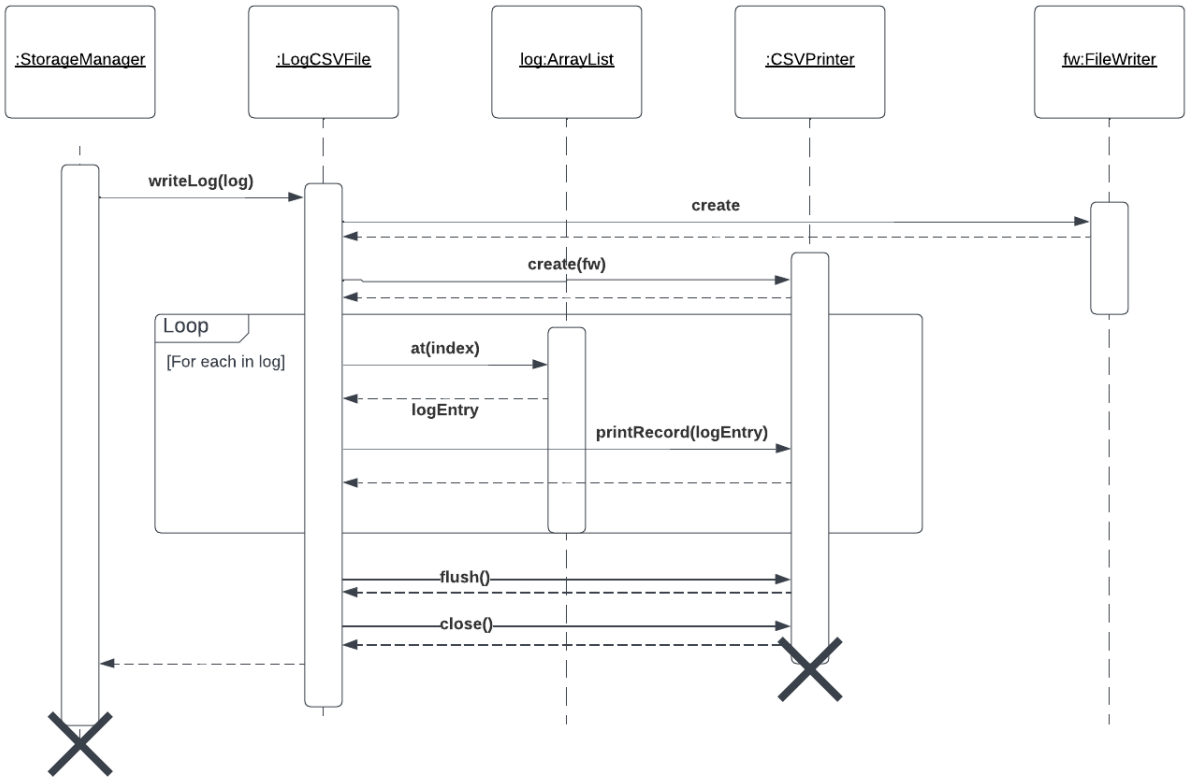
5.2 Add a serving of food to the log and saving the data



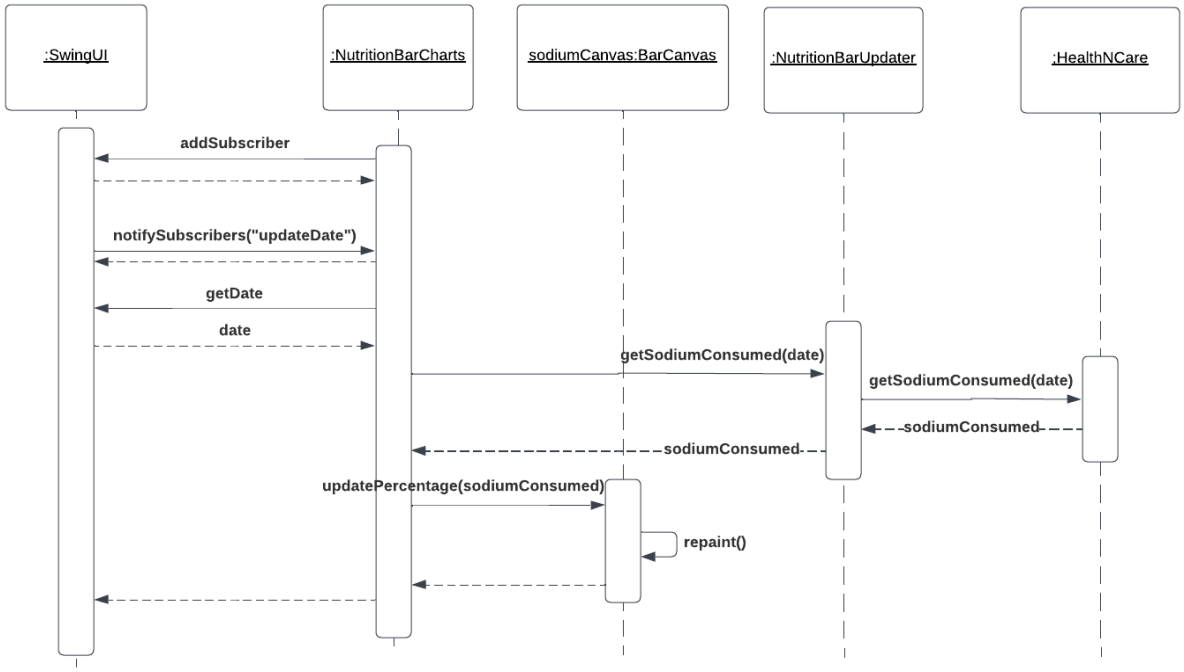
5.3 Compute the total calories for a given day



5.4 Writing log to CSV file



5.5 Updating the selected date



6 Pattern Usage

6.1 Observer

Observer Pattern	
Observer(s)	TextUI SwingUI
Observable(s)	HealthNCare

6.2 Composite

Composite Pattern	
Composite	Recipe
Component	IFood
Leaf	FoodItem

6.3 MVC

MVC Pattern	
Model	HealthNCare
View	SwingUI TextUI
Controller	LogEntryCRUDListener FoodCRUDListener ExerciseCRUDListener CheckCaloriesResponder DailyStatsResponder ItemListResponder NutritionChartUpdater SaveDataListener

7 RATIONALE

11/20/2023: We created this design document to keep track of our plans and decisions.

12/1/2023: We implemented the observer pattern within the view as well to notify components when the selected date has changed.

- We chose to use the Model View Controller architectural pattern because it allows us to keep the main categories of classes separate, maximizing extensibility and cohesion while minimizing coupling between classes.
- We are using the Observer pattern to allow automatic updating of the view when the state in the model changes. This allows the view to always be up to date without the need for the view to be constantly checking the model for the newest data. We also used the observer pattern to notify elements when the user selects a different date so that the data can be retrieved for that day.
- Chose to use the Composite pattern to allow us to represent recipes and food in part-whole relationships. This significantly decouples the recipe and food classes by having them both use a unified interface