### Team B - Used Napkin

Diet Manager - Version 2.0

Josh Schrader

Kevin Gleason

Neil Hiranandani Patrick Lennon Sean Decker

### Since Last Time

- The driving force behind our method of GUI implementation was to incorporate the already working CLI
- With this in mind, we would end up with a working GUI and CLI
- Seeing how effectively tabs could be used in a GUI of this nature, we decided to use a tabbing system for the functionality and have extra panels displaying the current information

### Default

Contains the DietManager class that has the public static void main() method.

\* Initializes the UIView

<=initializes>>

Initializes the controller first.

View

Presents the user with a GUI where they can choose what actions to perform.

Uses a mediator to connect actions in the controller with buttons in the GUI.

Is updated through the controller with state changes in the model.

Model

### Contains the Abstract Food class

- \* Generic representation of food
- \* Provides methods for any class of this type

### Contains the BasicFood class

- \* Contains nutritional information of the food
- \* Has ability to build own CSVFormat
- \* Has ability to copy itself

### Contains the Recipe class

- \* Contains a list of (Abstract) Food
- \* < Abstract > Food is aggregated by Recipe
- \* Has ability to build own CSVFormat
- \* Has ability to copy itself

### Contains the DietLog Class

- \* Contains hashmap of food consumed for each day
- \* Contains functionality to get total nutritional information for the selected day

### Contains the User Class

- \* Stores user information and desired goals
- \* Contains a DietLog object

### Contains the Foodl ist Class

\* Stores a list of available Food objects

### Contains the ExerciseList Class

\* Stored a list of all available Exercise Objects as a HashMap<String, Exercise>

### Contains ExerciseClass

- \* Represents exercise object
- \* Contains information about a specific exercise (name, calories burned per hour of a 100 pound person, and duration of exercise)

<<initializes>>

### Controller

- Initializes the CSVParser
- Uses CSVParser to read in csv files on startup and populate the respective internal model structures needed.
- Constructs User Model
- Constructs FoodList
- Performs any changes to the model based off of user changes.

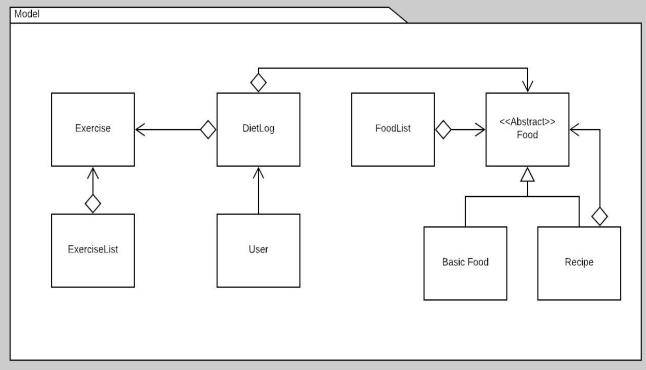
## Subsystem Structure

The Default package has the DietManager that initializes the DietView

The DietView initializes the DietController

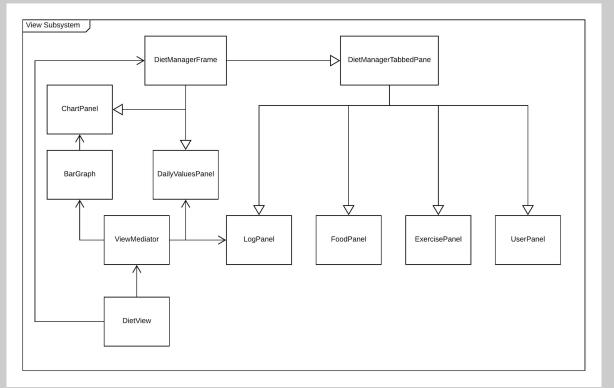
The DietController initializes the Model and the CSVParser, the parser populates the FoodList and ExerciseList after reading in the csvs

The Model contains all the classes, with DietLog containing most functionality for nutritional information calculations



# **MVC - Model Subsystem**

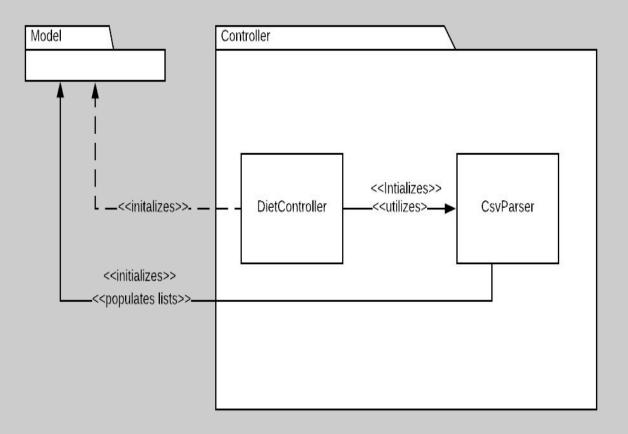
- Exercise and ExerciseList were treated similarly to Food and FoodList.
- Made our composites into aggregations.
- DietLog given more functionality to calculate calories burned through exercises.



- DailyValuesPanel displays an updated list of text information as foods and exercises are logged
- ChartPanel displays the bar graph that updates as food is logged

# **MVC - View Subsystem**

- DietManagerFrame broken up into 3 main parts - The DietManagerTabbedPane, DailyValuesPanel, and the ChartPanel
- DietManagerTabbedPane contains the User, Food, Exercise, and Log Panels



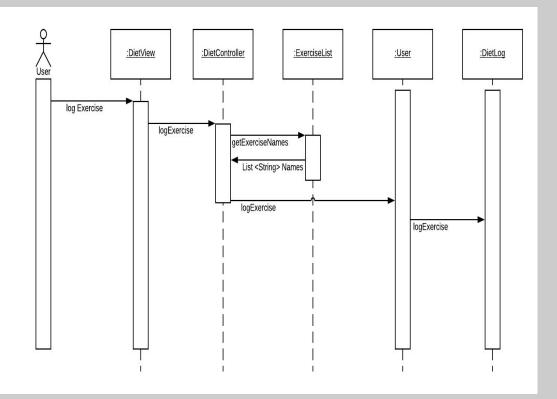
### MVC - Controller Subsystem

- Made the CSVParser know of the model to populate FoodList and ExerciseList.
- DietController knows of both the CSVParser and the Model, initializes both.

### :DietView :DietController :Exercise :ExerciseList add exercise command addExercise create Exercise Exercise addExercise

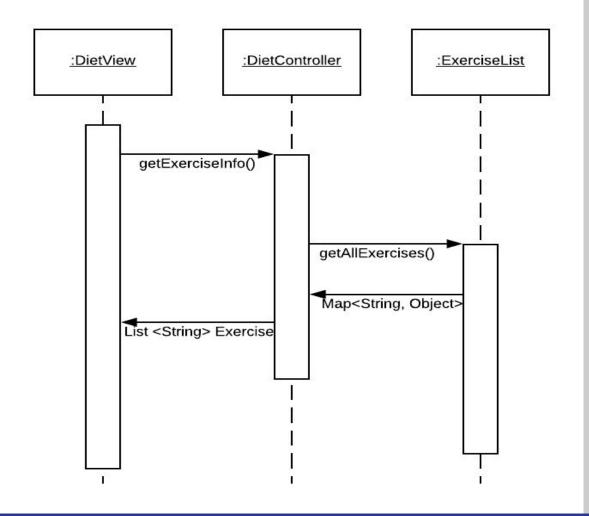
## Sequence Diagram - Add an Exercise

- The view tells the controller an Exercise needs to be made with the user input
- The controller makes a new Exercise object with the values
- The new Exercise is added to the ExerciseList



## Sequence Diagram - Log an Exercise

- The view tells the DietController to log one of the chosen Exercises.
- The controller checks the list of exercises for the name and gets the object associated with it.
- The controller tells the user to log the exercise object to their personal list
- The user updates the DietLog with the logged exercise



## Sequence Diagram - Show Exercise List

- DietView tells the DietController to get the names of all the exercises
- The controller gets all the exercises from the ExerciseList as a HashMap.
- The controller gets all the names from the key set.
- The controller passes the list of Strings to the view.

### The Good

- More group meetings than during R1
- Improved team coordination
- Solid use of github, feature branching, and pull-requests
- Implementation of UI in a way that required minimal editing of CLI
- CLI still functioning alongside the GUI, improves debugging of GUI

### The Bad

- UI implementation
- Code delays due to debugging
- IntelliJ (pathing issues)
- IntelliJ's built in UI builder produced an unusable product, forcing rework
- Sporadic Meeting times
- Time management with the Holiday
- Allocating tasks to members
- Team members not updating Trello tickets to reflect current status

### **DEMO TIME**