Design Sketch

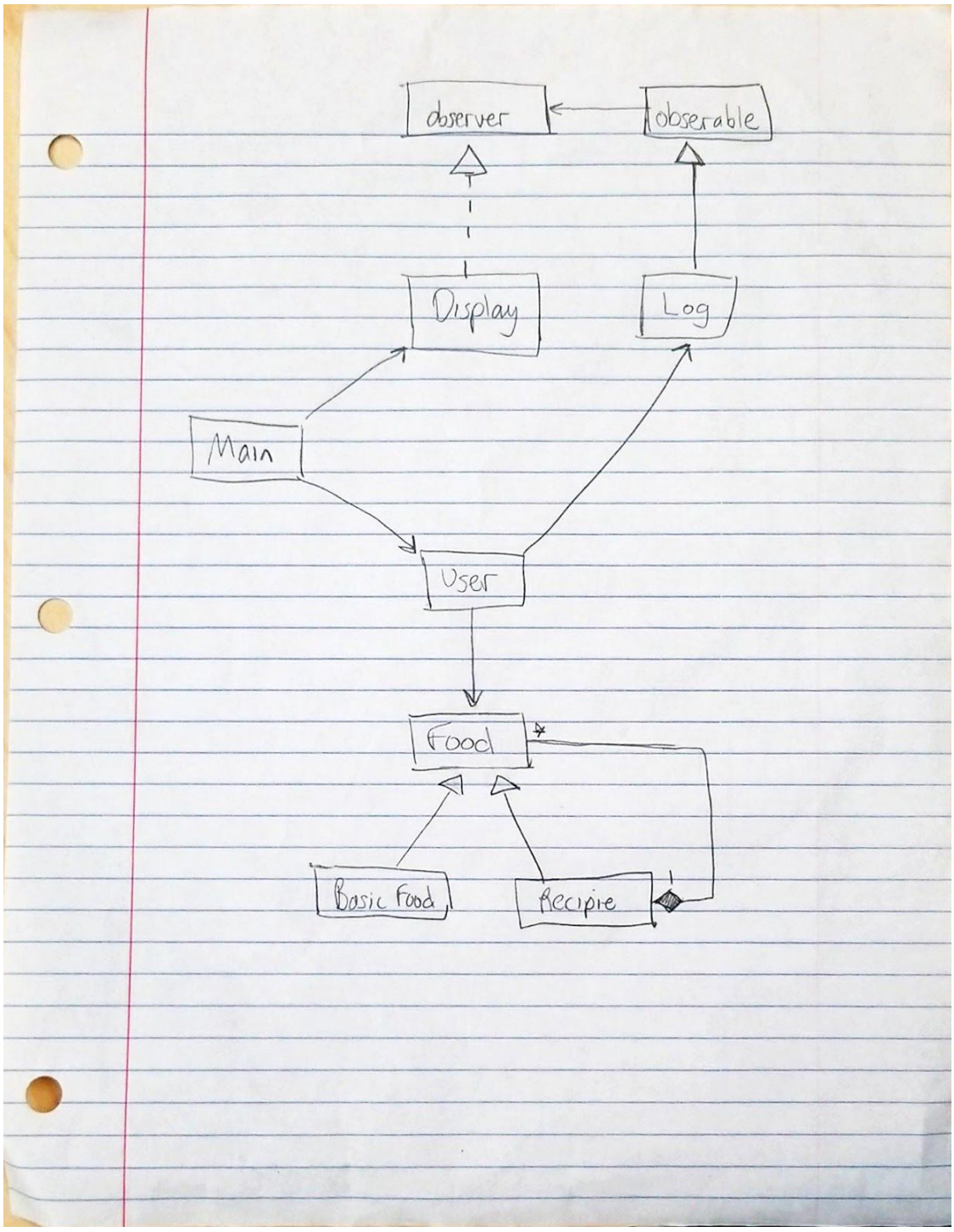
10/19/17

Team A

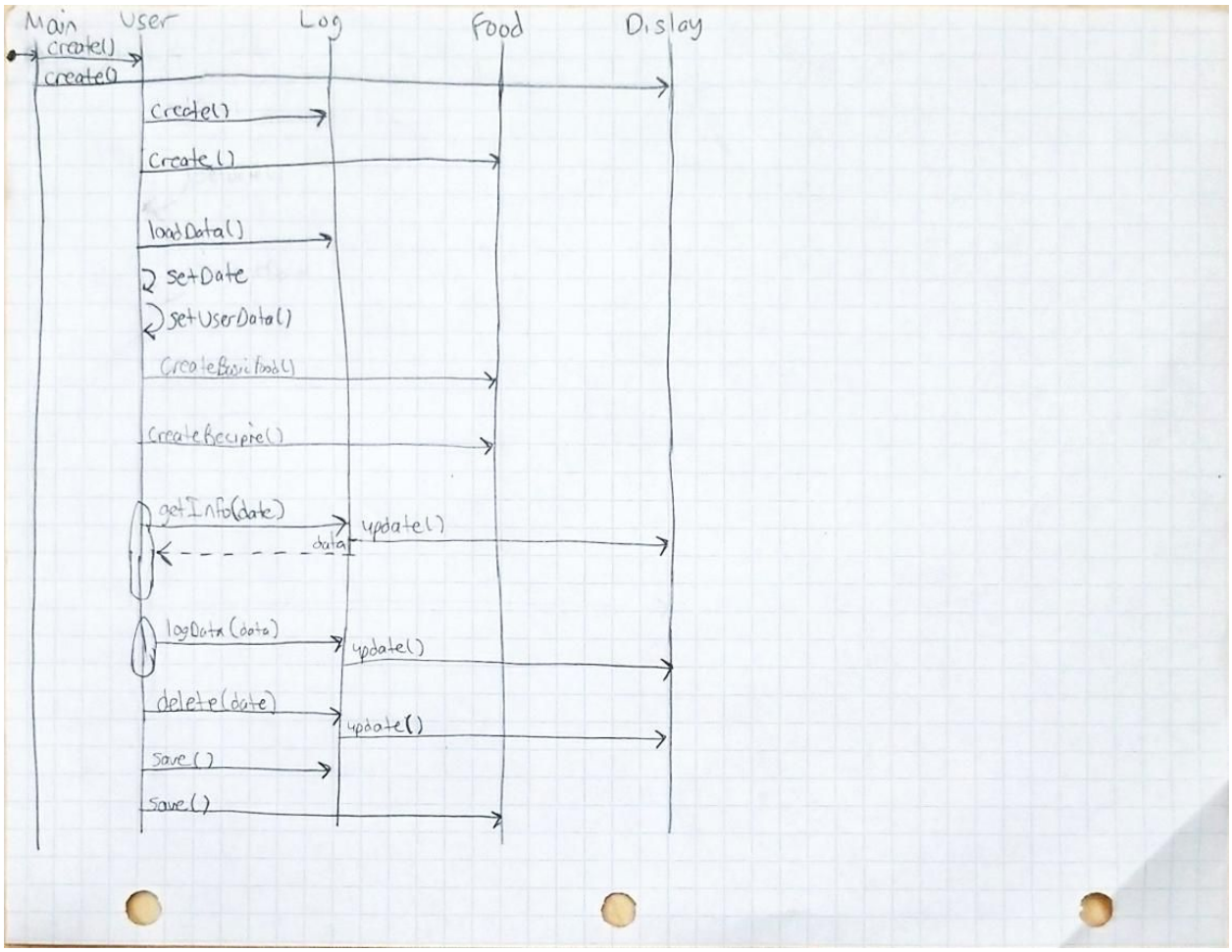
Brianna Buttaccio, Caitlyn Daly, Tiffany Ellis,

Kevin Reynolds, Roy Tran, Fu Quan Li

Class Diagram



Sequence Diagram



Class Description

Classes

Main – Main is responsible for creating a “Display” and a “User” object. Main knows about “Display” and “User”.

User – The user is responsible for storing the data about the user. It creates a “Basic Food”, “Recipe” and “Log” object. The user knows about “Log” and “Food”.

User can:

- Set a date for the log

Log – The log is responsible for keeping track of all the log data. Log can load the log file and store it. Log implements Observable so that the display can automatically update based on log data.

Log can:

- Return data about food consumed on a certain day, the number of servings, and total calories.
- Return the total number of calories consumed for a day
- Return whether they have exceeded their daily calorie limit given a specific day.
- Return the weight for a given day
- Return calorie limit for a given day
- Return a breakdown of nutrition in terms of the percentage of total grams of fats, carbohydrates, and protein, each to the nearest 1%. The total must be 100%
- Add a food item and the number of servings to the daily log on the currently selected date
- Add or update daily calorie limit for a given date
- Add or Update the weight for a given date
- Delete food items in the daily log for a specific food
- Save current state

Food – Food is an interface for “Basic Food” and “Recipe”. “Food” does not know about anything. It is the component in the composite pattern. This class is responsible

for defining the functionality of “Basic Food” and “Recipe”. “Food” holds the data for the calorie count and nutritional information.

Food Can:

- Load the food data and store it
- Create a new food
- Return data about food
- Save current state

Basic Food – “Basic Food” is the leaf in the composite pattern. All functionality is defined in “Food”. “Basic Food” knows about “Food”.

Recipe - “Recipe” is the composite in the composite pattern. All functionality is defined in “Food”. “Recipe” knows about “Food”. “Recipe” can also store “Food”.

Overview

We organized the system in a way to separate responsibilities in a way that makes sense. We used the Observer pattern for the display and the log data so that the display can automatically update when the log data is updated. How we display data is irrelevant to the actual data so we designed it so the display is completely independent of the data. If the requirements for the display change, we can easily change only the display class, which makes maintenance easier. Log tracks all of the data for the user, such as the food, calorie limit, and weight. Log is also responsible for doing some calculations. A disadvantage to this is that the log class will be fairly large and covers a broad spectrum of ideas. It may be better for us to split some of its functionality into multiple classes to have more focused classes, rather than the one large class.

We used the composite pattern for our basic food and recipe classes, with basic food being a leaf, recipe being the composite, and the component abstract being food. This allow us to include basic foods in our recipes, without having to re-make the food in the recipe class. We can also use this to only interact with the food abstraction, and not the recipe and basic food classes directly. The user can easily add a recipe or basic food to the log and the log will be treating them the same way. This is because basic food and recipes both have the same attributes such as calories and nutritional values. This design is advantagous as we can easily add, delete, or modify basic foods and recipes. It can also be a disadvantage as we could accidentally delete a basic food that a recipe is relying on, causing errors in the system.