Design Sketch

March 6th, 2019

Team B

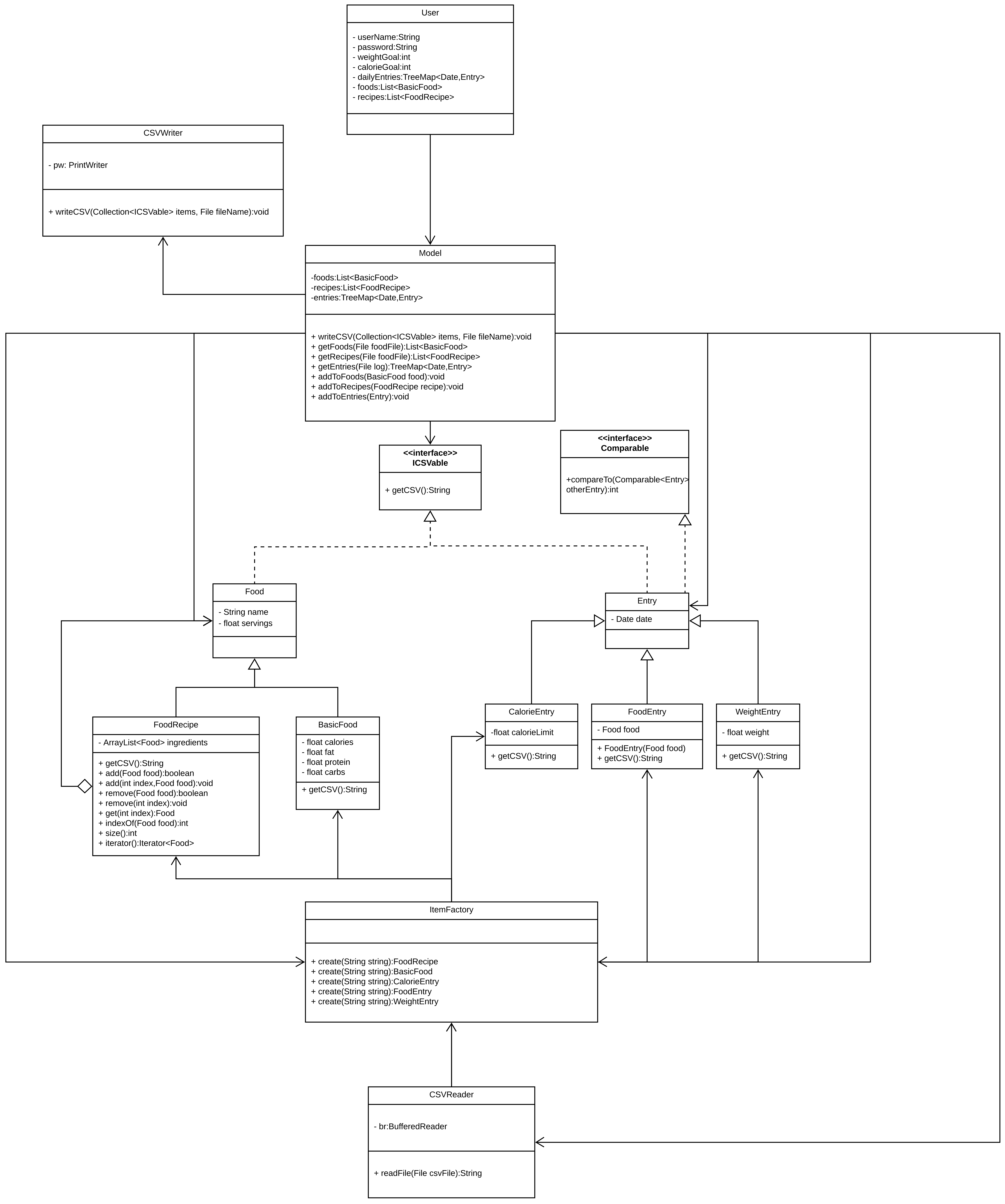
Zachary Lichvar

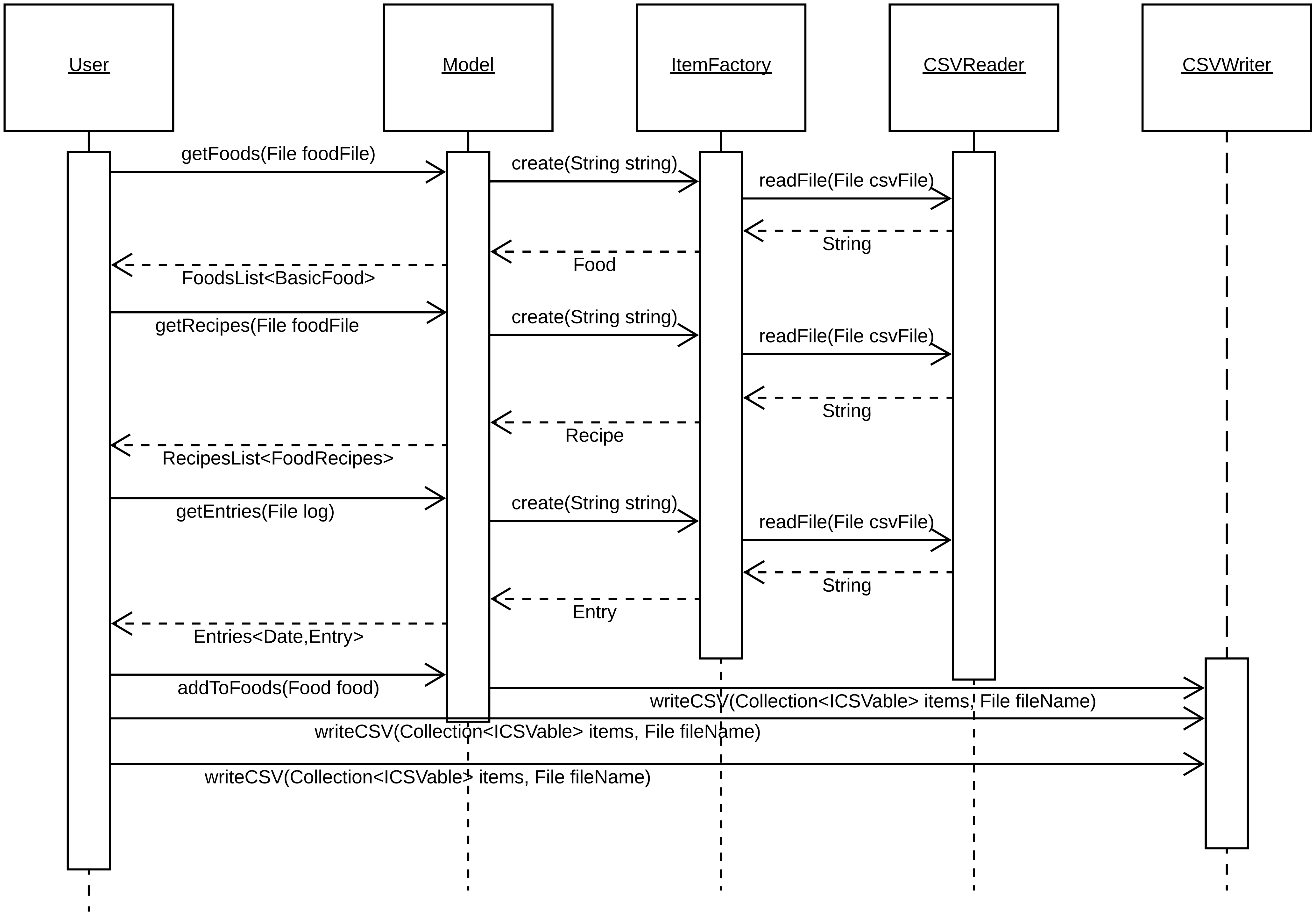
Brandon Connors

Sachi Nutulapati

Rosis Sharma

Brennan Jackson


User
- userName:String
- password:String
- weightGoal:int
- calorieGoal:int
- dailyEntries:TreeMap<Date,Entry>
- foods:List<BasicFood>
- recipes:List<FoodRecipe>
CSVWriter
- pw: PrintWriter
+ writeCSV(Collection<ICSVable> items, File fileName):void
Model
-foods:List<BasicFood>
-recipes:List<FoodRecipe>
-entries:TreeMap<Date,Entry>
+ writeCSV(Collection<ICSVable> items, File fileName):void
+ getFoods(File foodFile):List<BasicFood>
+ getRecipes(File foodFile):List<FoodRecipe>
+ getEntries(File log):TreeMap<Date,Entry>
+ addToFoods(BasicFood food):void
+ addToRecipes(FoodRecipe recipe):void
+ addToEntries(Entry):void
<<interface>> ICSVable
+ getCSV():String
<<interface>> Comparable
+compareTo(Comparable<Entry> otherEntry):int
Food
- String name
- float servings
Entry
- Date date
FoodRecipe
- ArrayList<Food> ingredients
+ getCSV():String
+ add(Food food):boolean
+ add(int index,Food food):void
+ remove(Food food):boolean
+ remove(int index):void
+ get(int index):Food
+ indexOf(Food food):int
+ size():int
+ iterator():Iterator<Food>
BasicFood
- float calories
- float fat
- float protein
- float carbs
+ getCSV():String
CalorieEntry
-float calorieLimit
+ getCSV():String
FoodEntry
- Food food
+ FoodEntry(Food food)
+ getCSV():String
WeightEntry
- float weight
+ getCSV():String
ItemFactory
+ create(String string):FoodRecipe
+ create(String string):BasicFood
+ create(String string):CalorieEntry
+ create(String string):FoodEntry
+ create(String string):WeightEntry
CSVReader
- br:BufferedReader
+ readFile(File csvFile):String


User
Model
ItemFactory
CSVReader
CSVWriter
getFoods(File foodFile)
create(String string)
readFile(File csvFile)
String
Food
FoodsList<BasicFood>
getRecipes(File foodFile
create(String string)
readFile(File csvFile)
String
Recipe
RecipesList<FoodRecipes>
getEntries(File log)
create(String string)
readFile(File csvFile)
String
Entry
Entries<Date,Entry>
addToFoods(Food food)
writeCSV(Collection<ICSVable> items, File fileName)
writeCSV(Collection<ICSVable> items, File fileName)
writeCSV(Collection<ICSVable> items, File fileName)

Rationale:

The goal our design is to support high cohesion, separating our concerns in a way such that each component performs one function well. Our implementation of `ItemFactory` (using the factory pattern) supports this goal.

Dependency injection was used to allow for higher extensibility. We implemented a composite pattern to allow our `Food` classes to have a shared behavior and create a tree structure within the `FoodRecipe` class.

The initial design shown displays only the Model and Controller, with the model knowing only of itself and the controller handling all else.

The Dependency Inversion principle is demonstrated by our implementation of the ICSVable interface, ensuring our high level components are not reliant on our low level ones. We also use Dependency Inversion...

User

The User Class represents attributes of a user account. Including login information, goals, daily logs, and access to the lists of foods and recipes.

Model

The model class represents interactions the user could make with the lower level models. This includes getting foods, recipes and log entries from the CSV files.

CSVWriter

The CSVWriter class is a utility class that allows parts of the application to write to a specified CSV file.

CSVReader

The CSVReader class is a utility class that allows parts of the application read lines from a specified CSV file as Strings to build collections of `Items`. The item type will be determined at runtime by the `ItemFactory`.

ICSVable

The ICSVable interface provides a shared behavior for all classes that can be written to a CSV file. It forces all implementers to create a `toCSV()` method, which will allow the `CSVWriter` to just call each items method to get the information it needs to write it to a file.

ItemFactory

The ItemFactory class uses a factory pattern to create concrete Food and Entry objects during runtime. These items are collected into lists, which are located in the `Model` class.

Food

The Food abstract class provides inheritance for the lower level `FoodRecipe` and `BasicFood` classes. It implements a composite pattern allowing for `FoodRecipe` to contain any child of the Food parent class

Entry

The `Entry` abstract class provides inheritance for the lower level `CalorieEntry`, `FoodEntry` and `WeightEntry` classes. The objects from this class are comparable to each other so that they can be stored in a sorted order making for quick and easy calculations later.

FoodRecipe

The `FoodRecipe` class is a composite component of a composite pattern for `Food` and its child classes. It holds a collection of `Food` objects that can be iterated through, added to or removed from.

BasicFood

The `BasicFood` class is a leaf component of a composite pattern for `Food` and its child classes. It represents the most simple form of the `Food` family having attributes relating to nutritional value.

CalorieEntry

The `CalorieEntry` class is a child of the `Entry` class, it is a model used to represent the CSV string of a calorie entry for a daily log.

FoodEntry

The `FoodEntry` class is a child of the `Entry` class, it is a model used to represent the CSV string of a `Food` entry for a daily log.

WeightEntry

The `WeightEntry`class is a child of the Entry class, it is a model used to represent the CSV string of a weight entry for a daily log.