PROJECT / RELEASE

Project Design Document

GROUP 1

Luka Šebalj <ls1378@g.rit.edu>
Pavel Spirovski Ehmer <ps9583@g.rit.edu>
Denim Behn <dsb4749@g.rit.edu>
Tomas Martincic <tm4818@g.rit.edu>
David Kraljic <dk9612@g.rit.edu>
Mario Stura <mxs9672@g.rit.edu>
Matej Petric <mp2272@g.rit.edu>

Project Summary

The goal of our Diet-Manager program is to provide a way for people to keep track of their diets that they intake. The program is meant to handle multiple foods and recipes along with descriptive information for all the foods, such as how many carbohydrates and calories they have. With a pretty straightforward user interface and a functional diet manager displayed to the user, it is very easy for the user to log their consumptions and be presented with information regarding them. The feedback is relevant to the input of the user. The user will also be able to set a personal goal that they can view their progress towards at will.

Design Overview

For the final part of the project we decide to use the pattern MVC and Composite.

Our MVC is structured in a way that allows us to split our classes into three groups: Model, View and Controller which ultimately work together in unison. Our Model classes are tasked with the communication and dynamic changing of data inside of our .csv files based on user input. The Model works with the Controller to successfully read and alter the .csv files. Our View classes are tasked with displaying everything to the user for them to interact with. The controller classes are tasked with connecting the Model and View classes to one another, they aid the View in retrieving data from the Model and displaying it to the user, as well as sending the input from the View part up to the Model.

The Composite part of our project is used in regards to the Program to Interface implementation as it is perfectly suited to this project due to having an interface which uses other classes making it easier to maintain over time and include new additional features at future times. The code is structured and written using DRY principles to assure that the application has a clean looking code structure as well as reusable features throughout it.

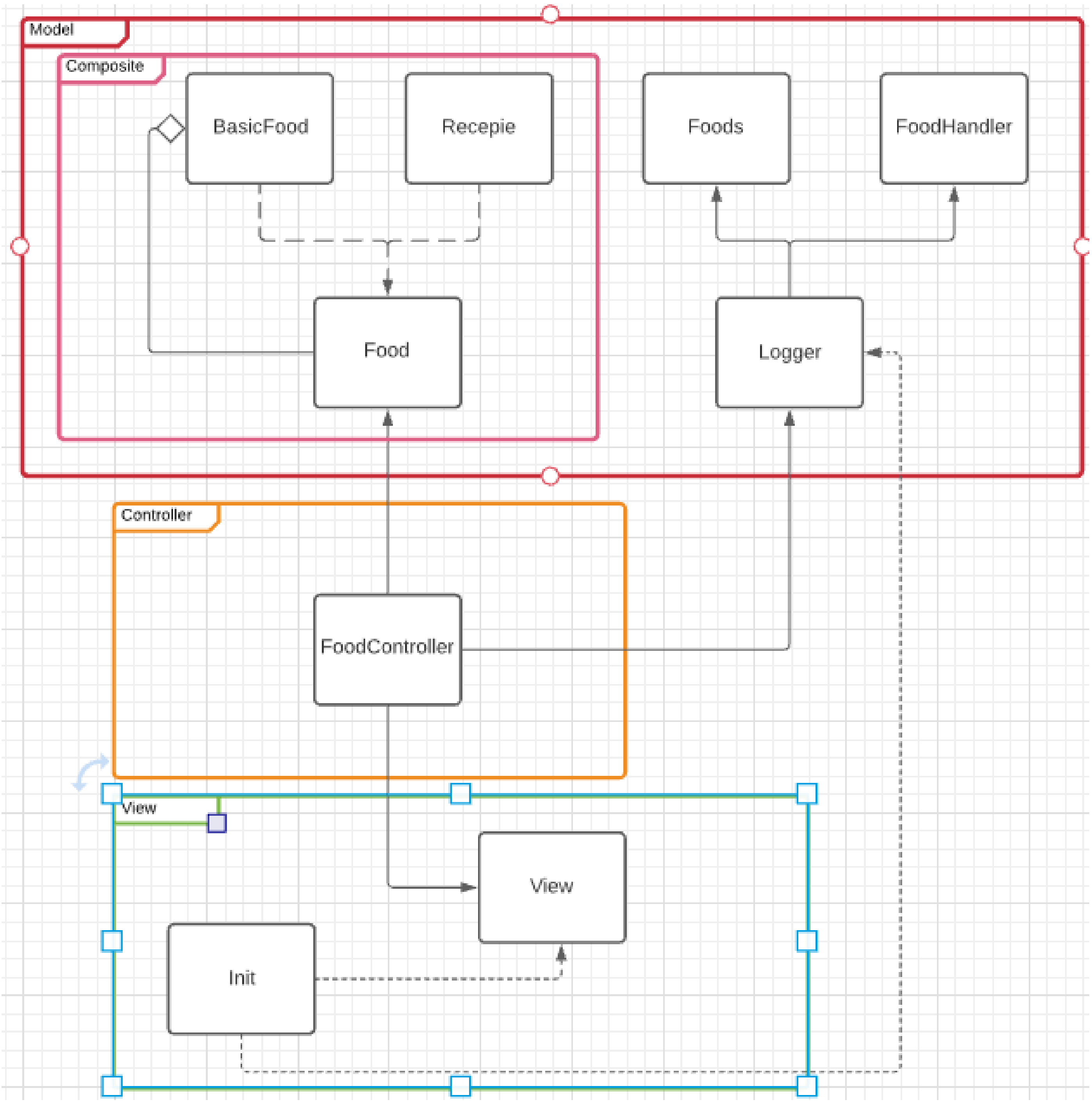
Overall System Structure

Our program consists of three subsystems. Model (and Composite sub-subsytem), View, Controller.

Our Controller is directly connected to all other subsystems as it connects all of them. The Controller's main class is the FoodController class which is responsible for sending data from the model to the view to be displayed to the user.

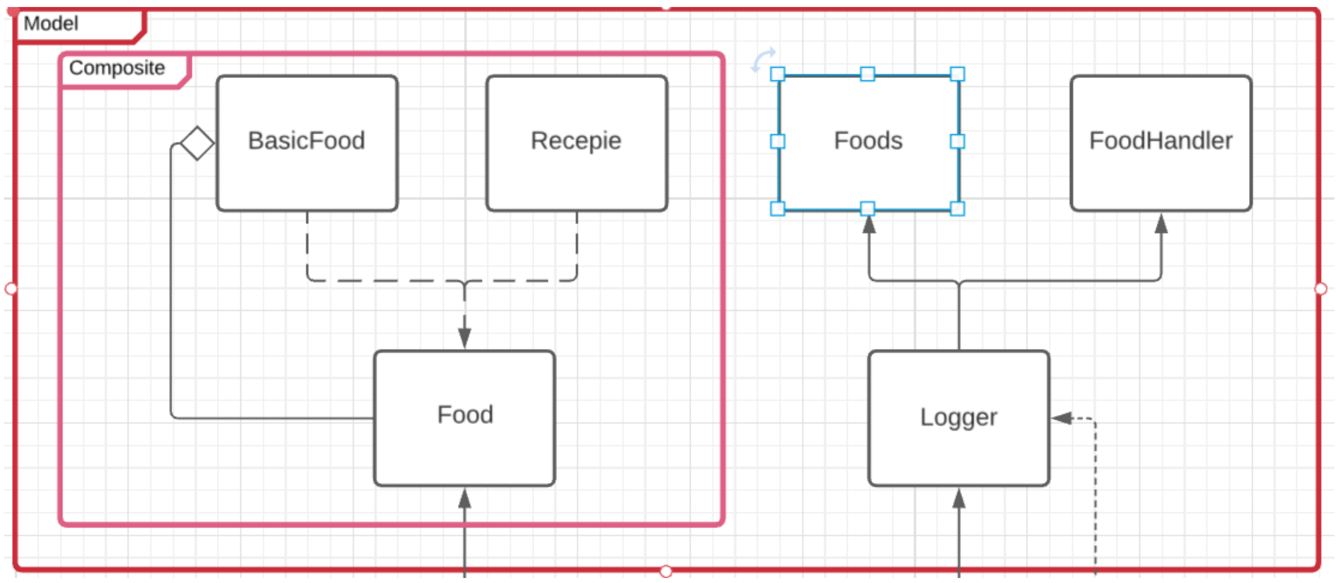
The Controller and Model have a has-a relationship. The data retrieved from the Model comes from the Food interface and Logger class. The Logger class logs all the information about the foods, directly connected to the FoodController. Recepie and Foods classes are models. The Recepie class provides and manages the data and logic of the recipes while the Foods class provides and manages the data and logic of the food. In the Composite sub-subsystem we have a class BasicFood representing a specific food item from the list. Each item that is read from the file will be constructed as a Food model object. FoodHandler is referenced by Logger and is responsible for reading the foods.csv and to add new lines into the file after user input.

The View subsystem has two classes, the Init tester class and the View class - the GUI. The View class is used to receive data from the Controller (which is taken from the Model) and display the user interface to the user. Its Init class is used to execute the program. The View and Controller also have a has-a relationship.


Model
Composite
BasicFood
Recepie
Foods
FoodHandler
Food
Logger
Controller
FoodController
View
View
Init

Subsystems

Subsystem Model

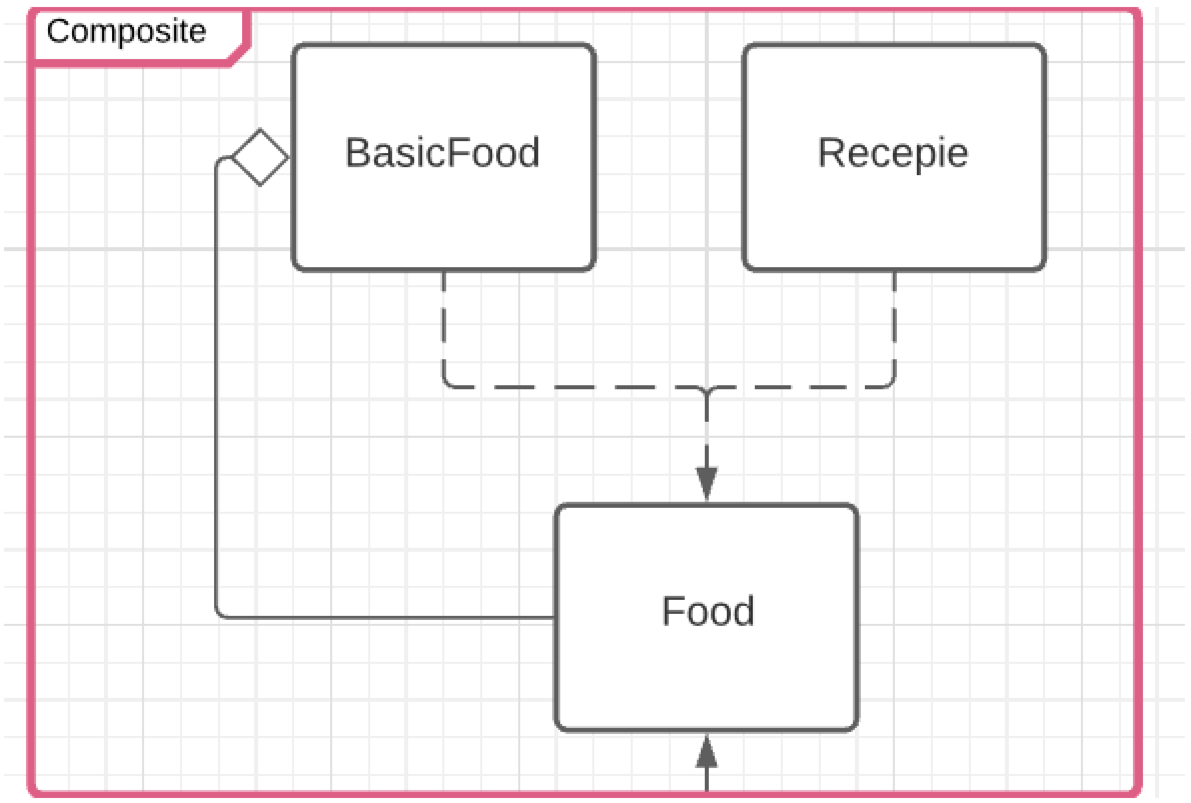


Class: Foods	
Responsibilities This class is responsible for reading and interpreting foods.csv. This will be used to access the file and return food model or recepie model which will be further used within the controller	**Collaborators** Logger

Class: Logger	
Responsibilities The class is responsible for logging the information about the foods.	**Collaborators** FoodController, Foods, FoodHandler

Class: FoodHandler	
Responsibilities The class is responsible for reading the foods.csv and to add new line into the file after user input	**Collaborators** Logger

Sub-subsystem Composite

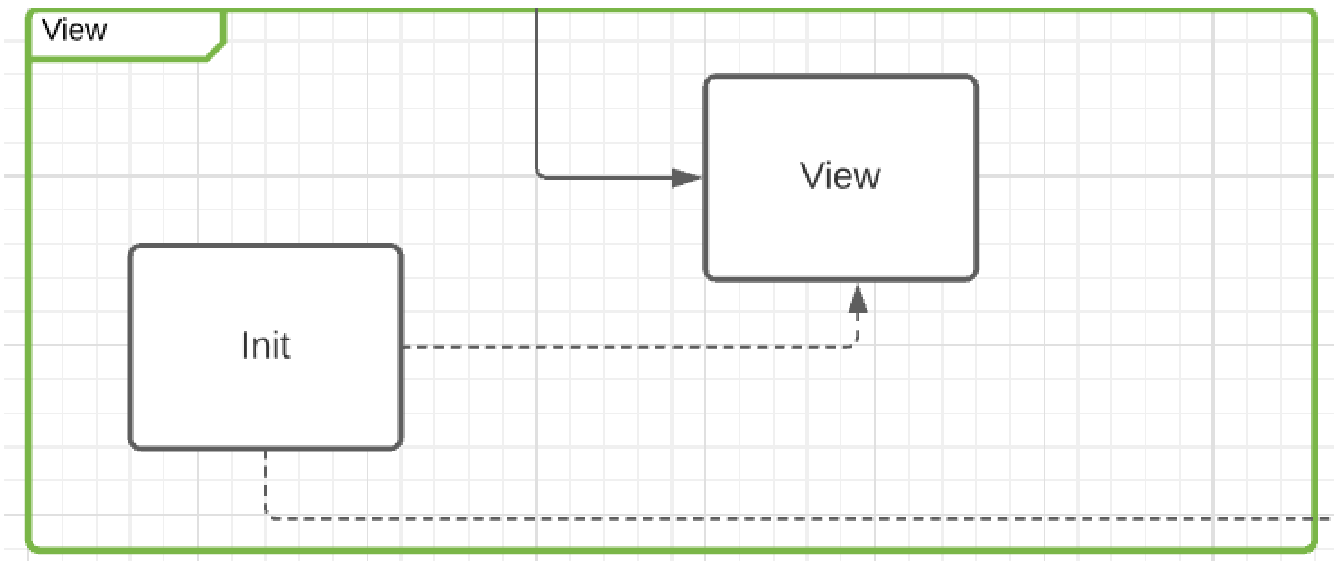


Class: BasicFood	
Responsibilities This class represents a specific food item from the list. Each item that is read from the file will be constructed as a Food model object.	**Collaborators** Food

Class: Recepie	
Responsibilities This class is responsible for holding data and information about the recipes.	**Collaborators** Food

Interface: Food	
Responsibilities This class is responsible for holding data and information about the foods.	**Collaborators** FoodController

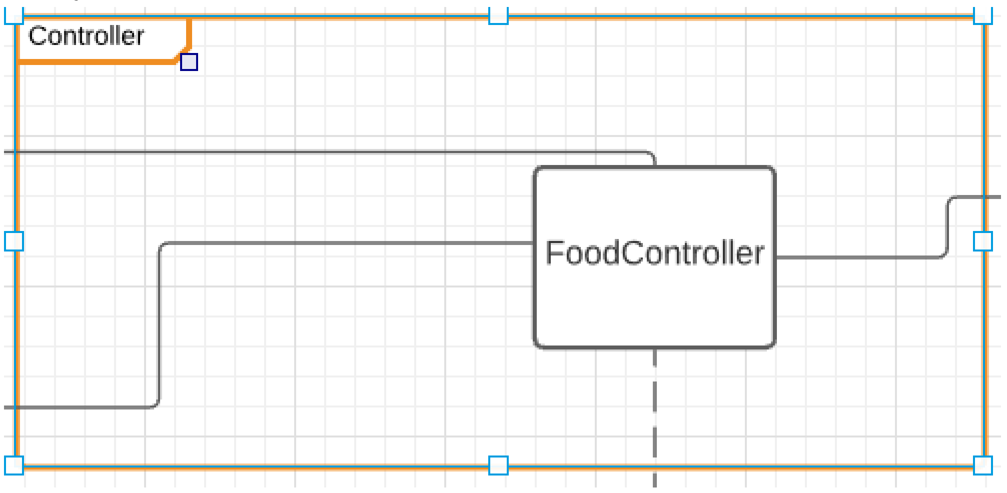
Subsystem View



Class: View	
Responsibilities The class is responsible for creating the user interface	**Collaborators**

Class: Init	
Responsibilities The class is responsible for executing the program	**Collaborators** View Logger

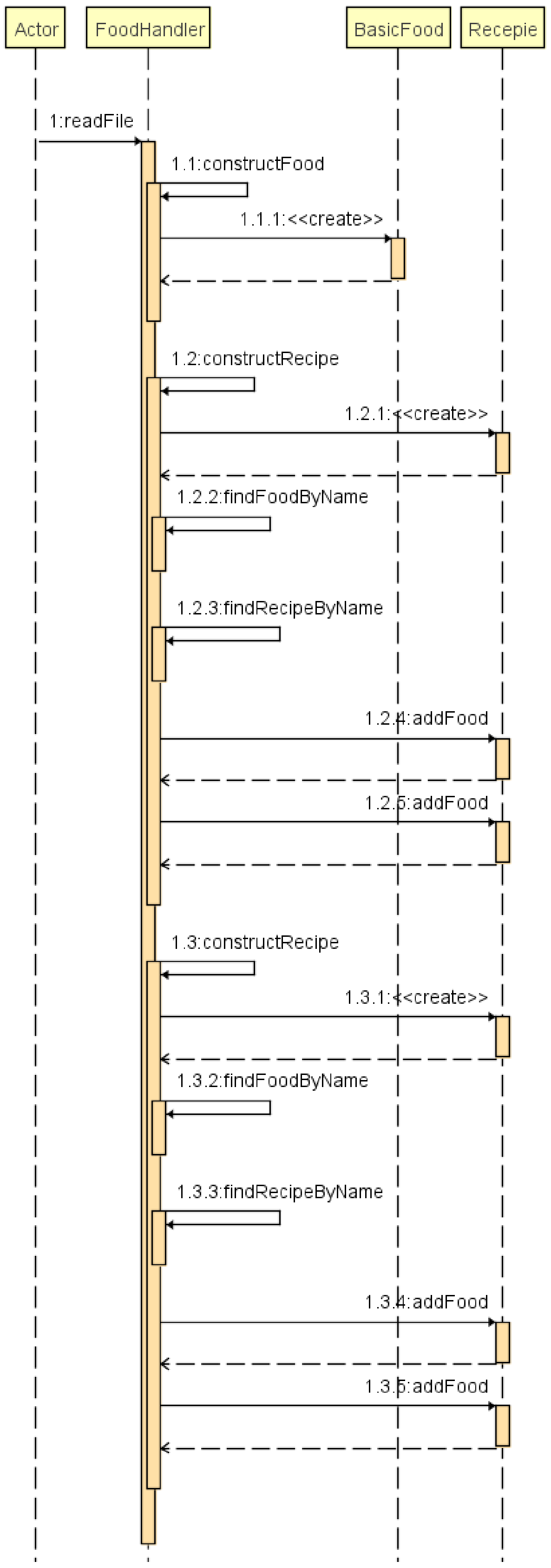
Subsystem Controller



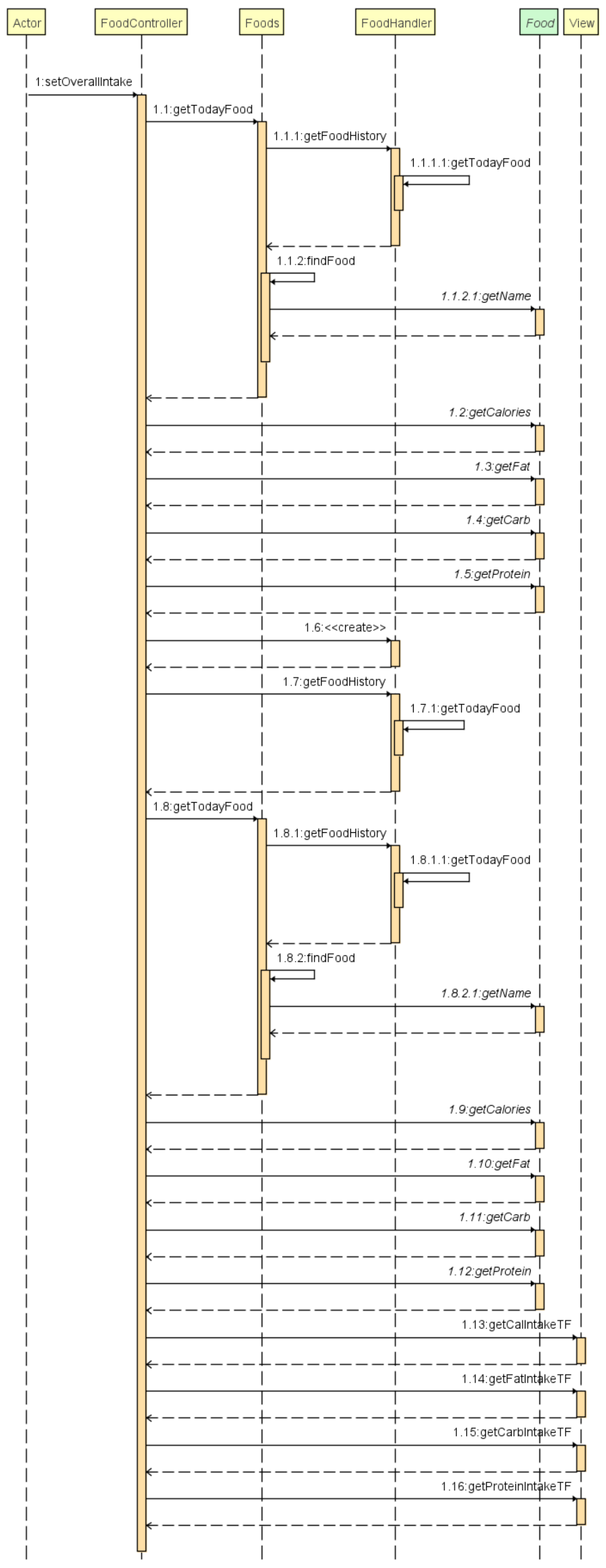
Class: FoodController	
Responsibilities The class is responsible for sending data from model to the view	**Collaborators** All classes

Sequence Diagrams

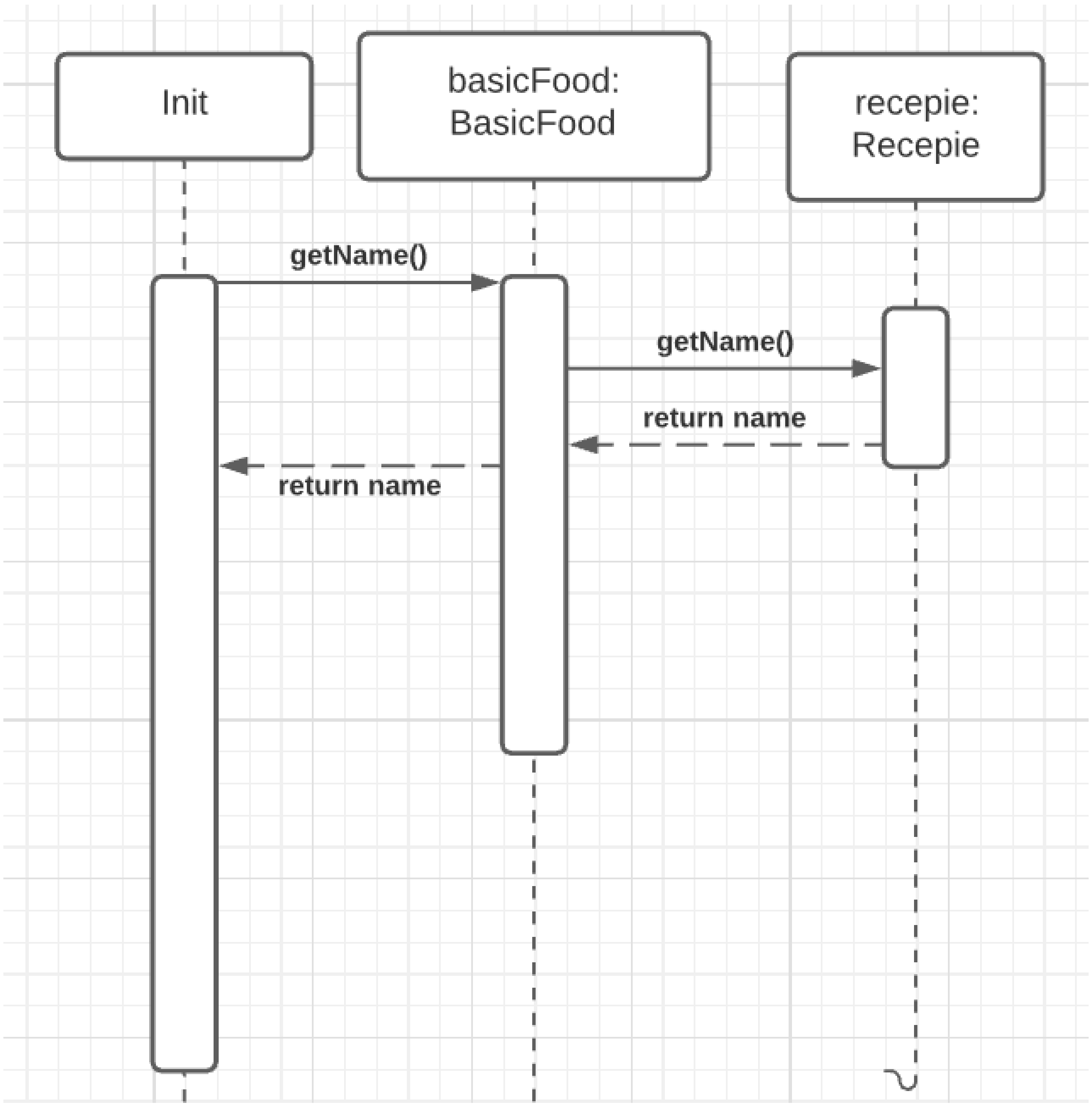
Sequence Diagram 1 (reading from food database).



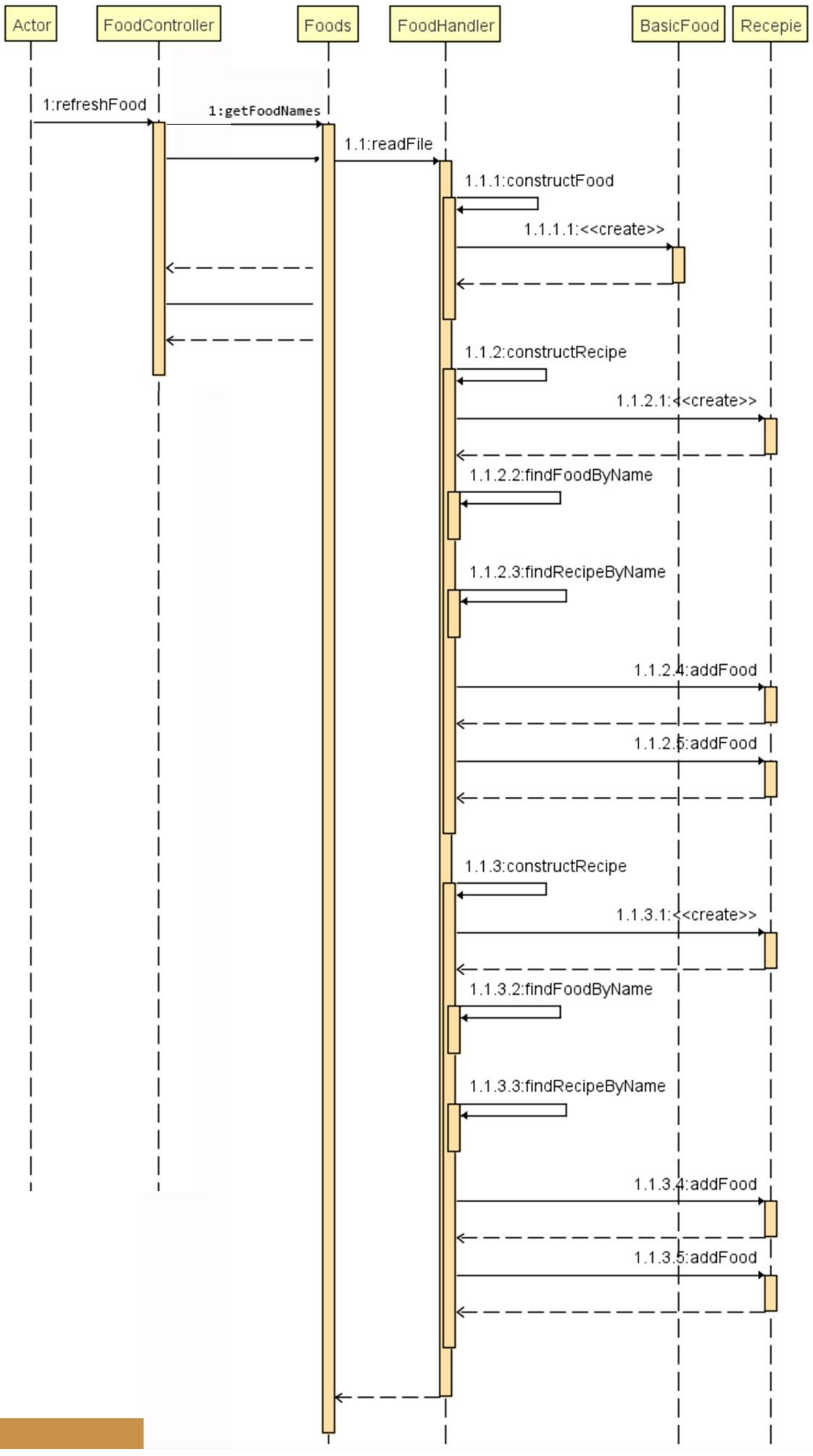
Sequence Diagram 2 (compute total number of calories for current date)



Sequence Diagram 3 (Composite - getName() method)

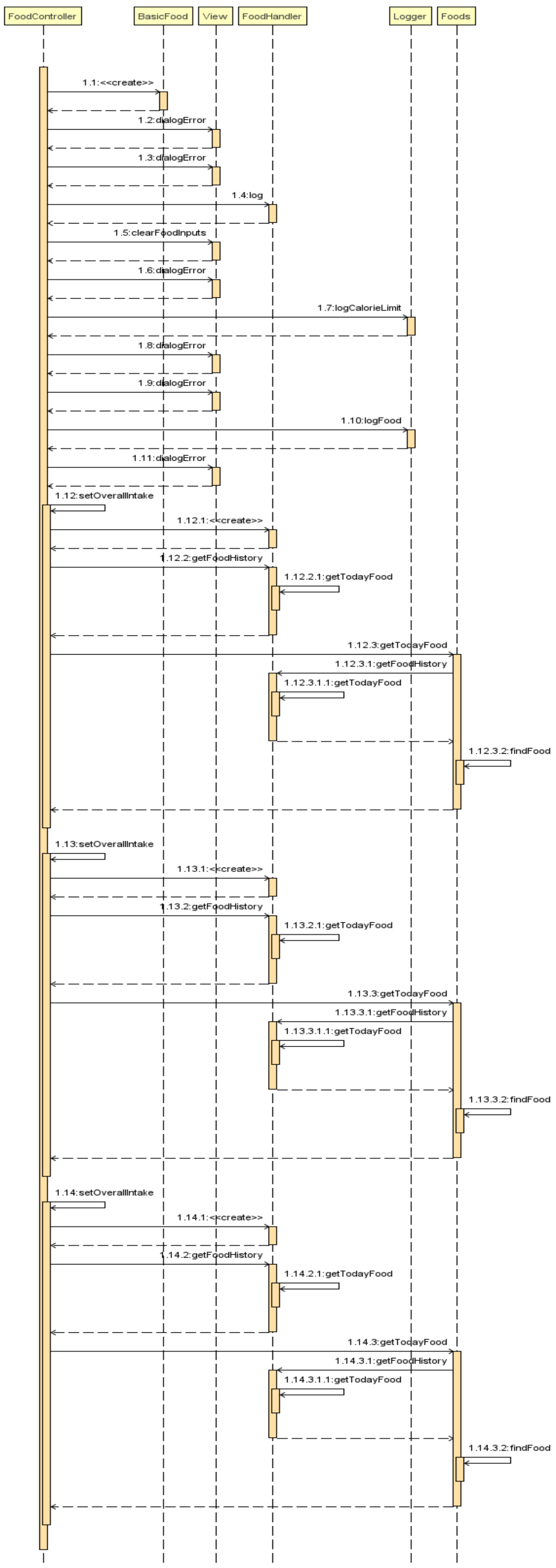


Sequence Diagram 4 (loading data for 1 basic food and 1 recipe)



Sequence Diagram 5 (Add servings to log entry for the current date)

We are adding 1 serving of the Pizza Slice & 2 servings of the PB+J to log entry for the current date.


FoodController
BasicFood
View
FoodHandler
Logger
Foods
1.1:<<create>>
1.2:dialogError
1.3:dialogError
1.4:log
1.5:clearFoodInputs
1.6:dialogError
1.7:logCalorieLimit
1.8:dialogError
1.9:dialogError
1.10:logFood
1.11:dialogError
1.12:setOverallIntake
1.12.1:<<create>>
1.12.2:getFoodHistory
1.12.2.1:getTodayFood
1.12.3:getTodayFood
1.12.3.1:getFoodHistory
1.12.3.1.1:getTodayFood
1.12.3.2:findFood
1.13:setOverallIntake
1.13.1:<<create>>
1.13.2:getFoodHistory
1.13.2.1:getTodayFood
1.13.3:getTodayFood
1.13.3.1:getFoodHistory
1.13.3.1.1:getTodayFood
1.13.3.2:findFood
1.14:setOverallIntake
1.14.1:<<create>>
1.14.2:getFoodHistory
1.14.2.1:getTodayFood
1.14.3:getTodayFood
1.14.3.1:getFoodHistory
1.14.3.1.1:getTodayFood
1.14.3.2:findFood