DIET MANAGER / INITIAL

Project Design Document

TEAM A

Matthew Atwell <ma3505@rit.edu>
David Camacho <dac6892@rit.edu>
Nathan Hinds <nah7111@rit.edu>
Brendon Strowe <Brendon.Strowe@mail.rit.edu>
Jenna Tillotson <jet7469@rit.edu>

Project Summary

The Diet Manager application aims to provide a way for users to manage the foods they consume on a daily basis and track metrics over time. Users can enter daily logs to report their daily weight, calorie limit, and food intake. From these logs, users can track changes over time. Users can also create a collection of various foods with associated dietary information (i.e. calories, grams of fat, carbohydrates, and protein). Users can then combine these foods into recipes made up of either those basic foods or other smaller recipes.

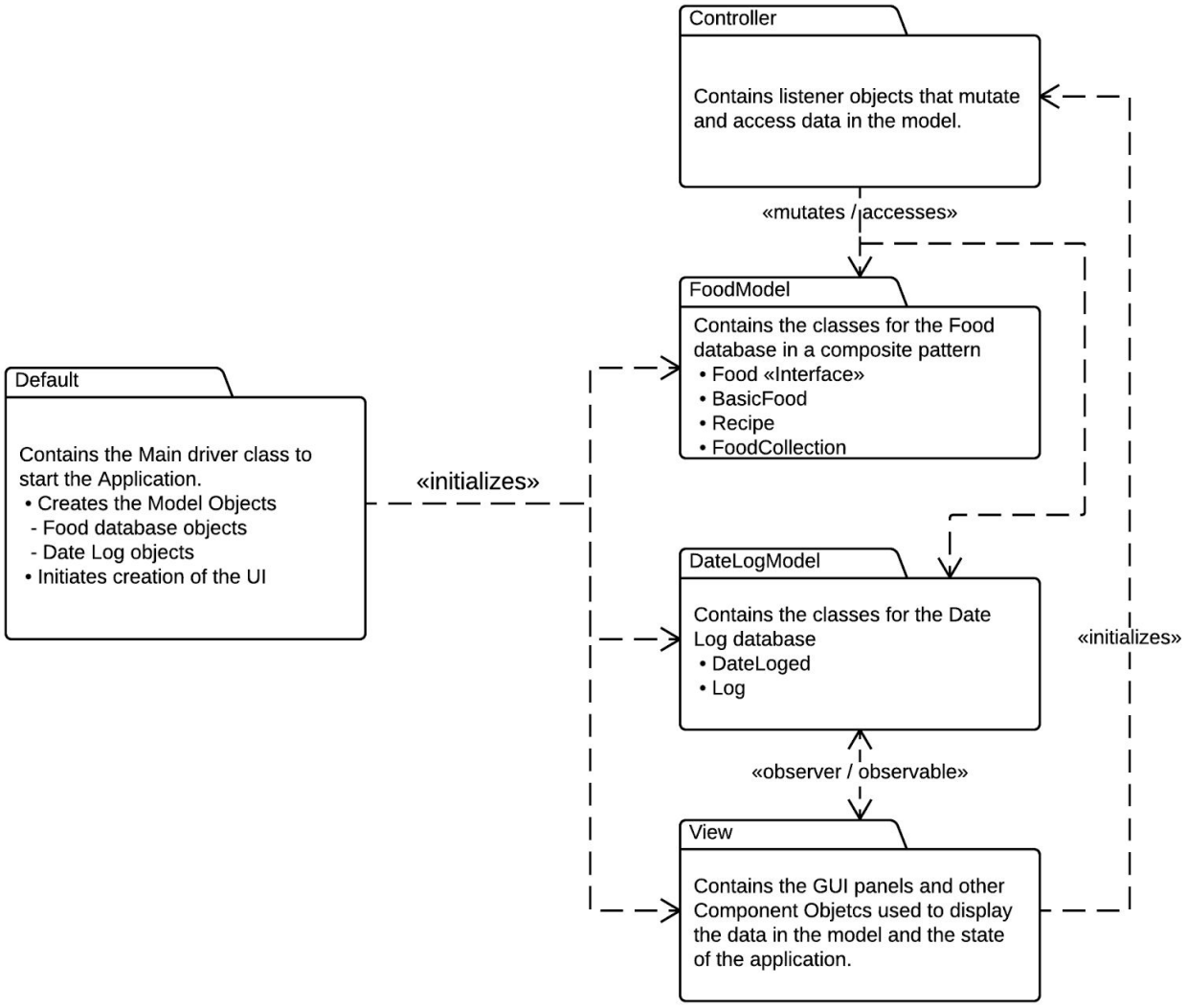
From the date entries and the food collection the user creates, they are able to make daily entries into a log of their weight, their calorie limit, and the servings of foods they consumed on a given day. The application will then be able to generate graphics depicting totals different nutritional areas consumed on that day (i.e. calories, grams of fat, carbohydrates, and protein) as well as changes in weight over time. The application can also determine if the user is over or under their calorie limit goal.

Users will be able to save out their information at any time to CSV-formatted files as well as read them in.

Design Overview

Our design from the preliminary design sketch all the way to the final implementation did not change much. In the beginning we started out with the composite pattern using Food, BasicFood and Recipes and putting them each of them in a component, leaf and composite, respectively. From there we were able to use that pattern and incorporate it into our MVC pattern. Since we are using the MVC pattern it automatically gives us a good separation of concerns design where we will have classes that have specific function in the Model, View and Controller pattern. Initially, we had the FoodCollection collect BasicFood and Recipe Objects, but later changed it to collect only Food objects as it was easier to implement. This would lower our coupling in the Model subsystem as FoodCollection will only have to depend on Food to get the objects. Our cohesion for our classes are not as high as our coupling is pretty low based on our design pattern.

Subsystem Structure

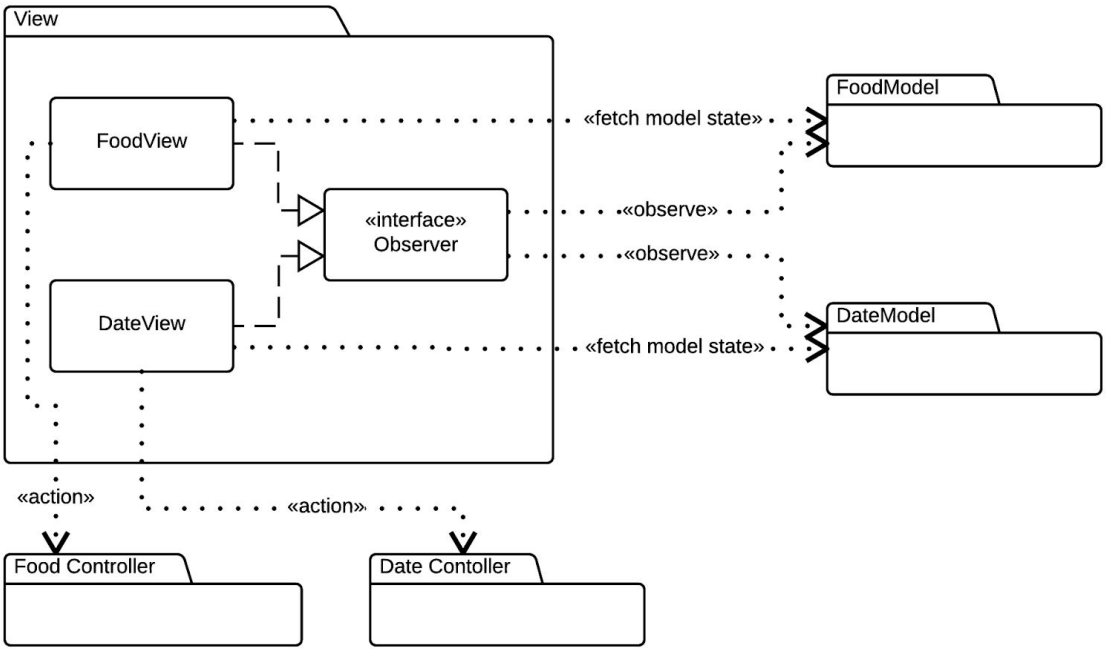


Subsystems

View Subsystem

Class DateView	
Responsibilities	Display and modify information through a GUI about the current LoggedDate Object. Give access to viewing a list of other LoggedDate Objects.
Collaborators (implements)	**java.util.Observer** - to observe the Log of dates
Collaborators (uses)	**model.LoggedDate** - The primary model class **model.Log** - Where other LoggedDate Objects are retrieved from **controller.DateController** - How modifications to the LoggedDates are made **java.util.Observable** - to register Log change notifications

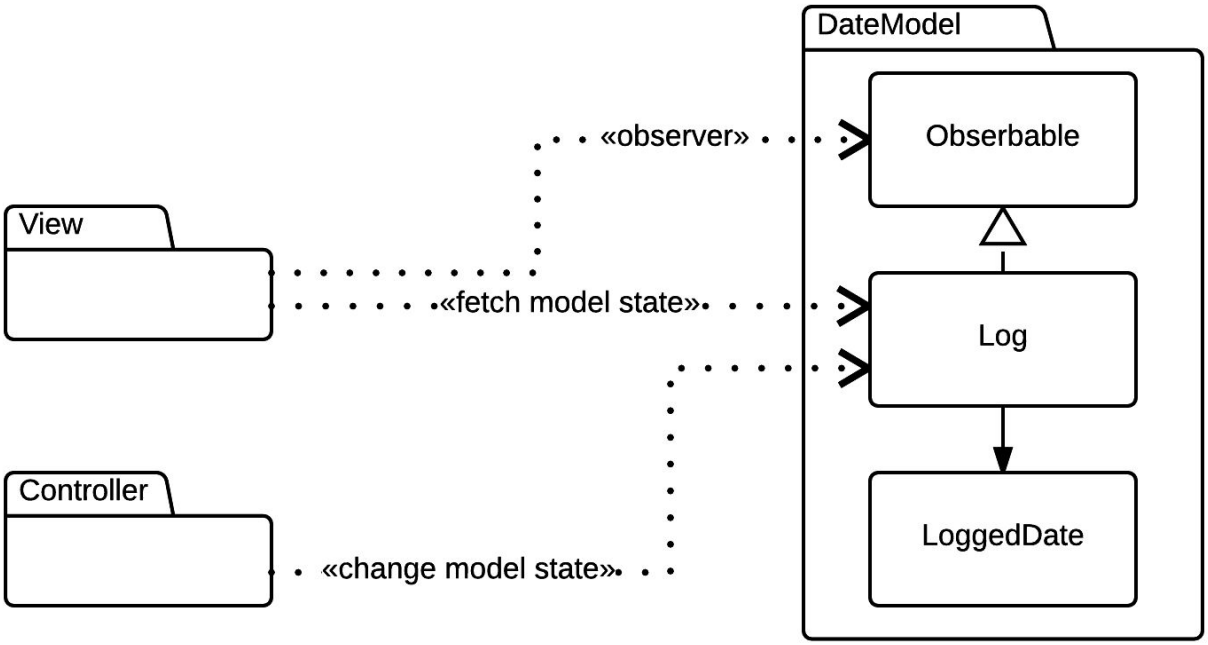
Class FoodView	
Responsibilities	Displays currently stored BasicFoods and Recipes through a GUI. Gives means to modify Food Objects
Collaborators (implements)	**java.util.Observer** - to observe the collection of Food
Collaborators (uses)	**model.FoodCollection** - The primary model class for accessing Food Objects **controller.FoodController** - The primary model class for mutiating Food Objects **java.util.Observable** - to register Food colelction change notifications



DateModel Subsystem

Class Log	
Responsibilities	Contains a list of all LoggedDate objects that have been created.
Collaborators (inherits)	**java.util.Observable** - so Log changes can be observed by others.
Collaborators (uses)	**model.LoggedDate** - LoggedDate objects that are stored in the Log **java.util.Observer** - for notifications of Log changes to views.

Class LoggedDate	
Responsibilities	Handles a set weight and calorie limit for a given day. Tracks the list of Food a user has consumed on a given day.
Collaborators (uses)	**model.Food** - Used to reference Food Objects stored



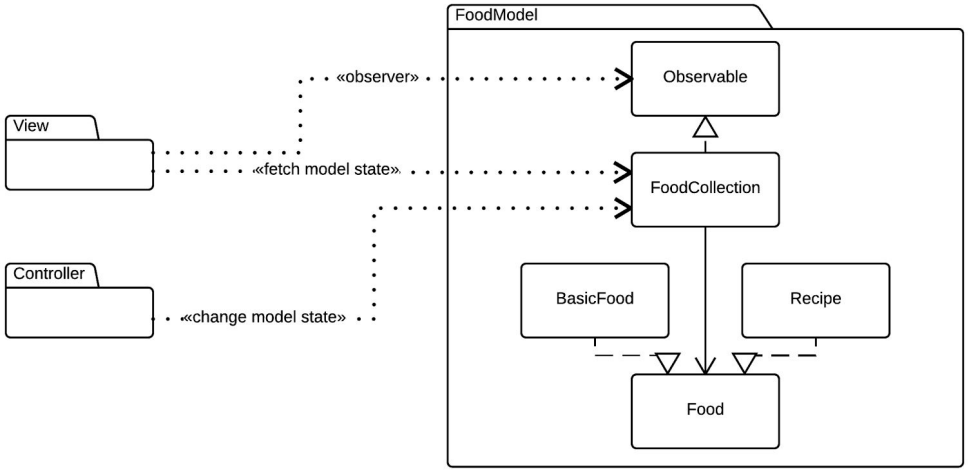
FoodModel Subsystem

Class Food	
Responsibilities	Methods and properties which will be used by all Food Objects, both BasicFood and Recipe Objects.

Class BasicFood	
Responsibilities	Defines Food Objects that the user can use in recipes as well as consume.
Collaborators (inherits)	**FoodModel.Food** - All BasicFood objects are Food objects

Class Recipe	
Responsibilities	A collection of Food Objects that can be made up of BasicFood as well as other Recipes.
Collaborators (inherits)	**FoodModel.Food** - All BasicFood objects are Food objects
Collaborators (uses)	**FoodModel.Recipe** - Recipes can be made up of BasicFood objects

Class FoodCollection	
Responsibilities	Lists a user's BasicFoods and Recipes that they create.
Collaborators (inherits)	**java.util.Observable** - so FoodCollection changes can be observed by others.
Collaborators (uses)	**FoodModel.Food** - FoodCollection contains a collection of Food objects. **java.util.Observer** - for notifications of FoodCollection changes to views.

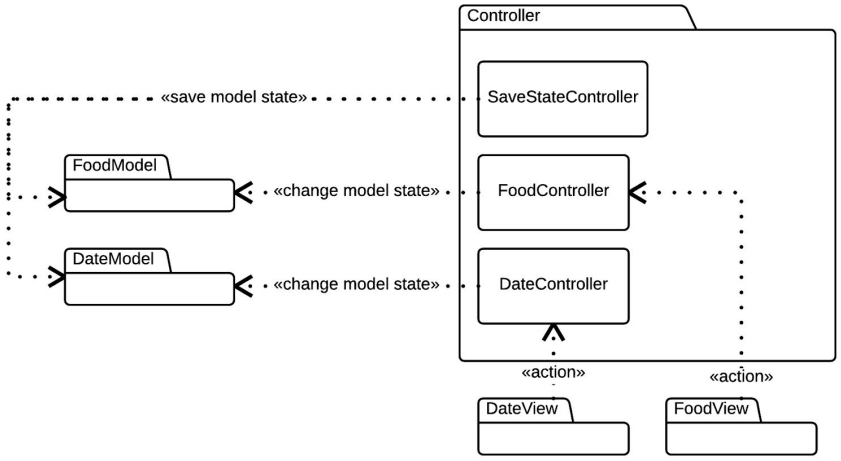


Controller Subsystem

Class DateController	
Responsibilities	Gets the LoggedDate Objects from the log CSV. Passes that information to the DateView. Populates the Log of LoggedDate objects Controls the creation of new LoggedDate objects Controls the modification of LoggedDate objects
Collaborators (uses)	**DateModel.Log** - place for storing LoggedDate objects while the application runs **DateModel.LoggedDate** - Objects to be manipulated and read **View.DateView** - Where the read LoggedDate objects are shown

Class FoodController	
Responsibilities	Allow creation of new BasicFood or Recipe objects. Gets the BasicFood Objects and Recipe objects that are in the food collection class and passes them to the FoodView.
Collaborators (uses)	**FoodModel.FoodCollection** - place for storing Food objects while the application runs **FooModel.Food** - Objects to be manipulated and read **view.FoodView** - Where the read Food objects are shown

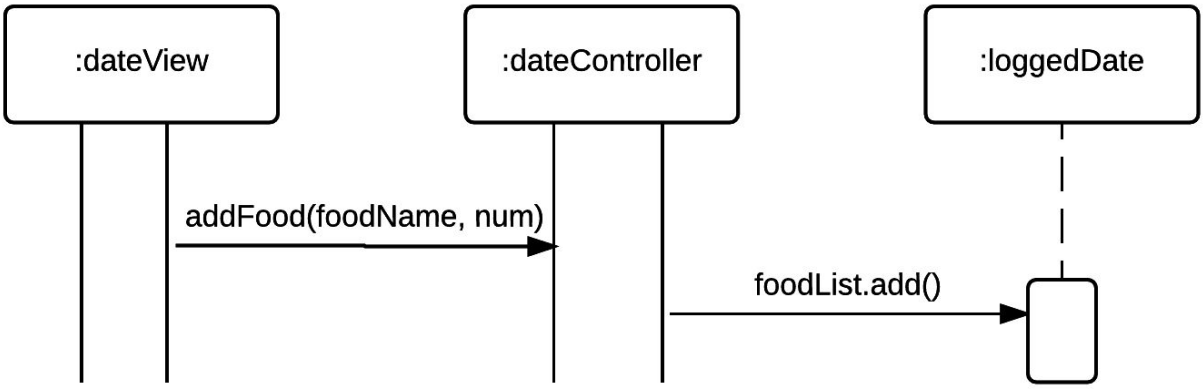
Class SaveStateController	
Responsibilities	Gets the information from the Log and FoodCollection lists and saves them out to CSV files upon invocation from any UI View.
Collaborators (uses)	**DateModel.Log** - where all of the LoggedDate data is gathered from to save. **FoodModel.FoodCollection** - where all of the Food data is gathered from.



Sequence Diagrams

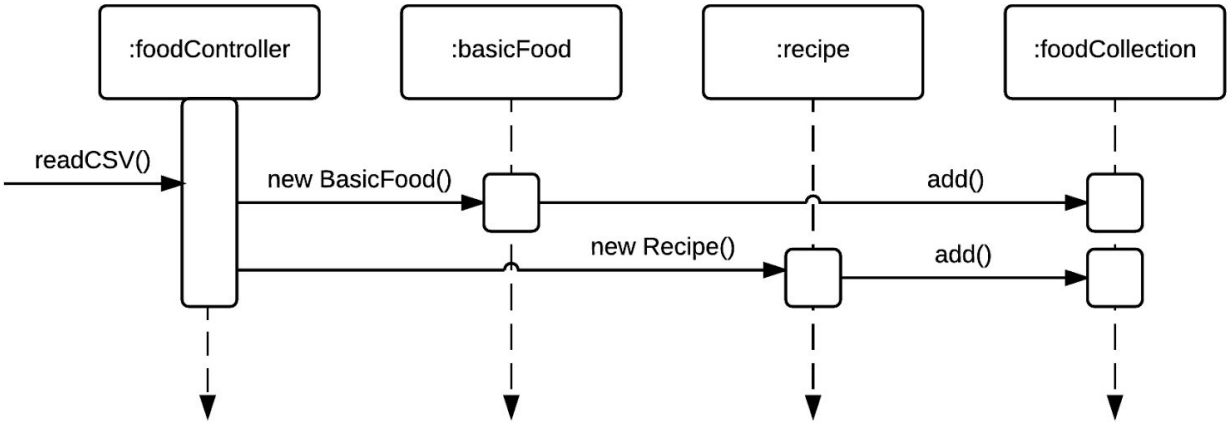
Sequence 1 - Adding a Food Entry

User adds a new food entry on the current date



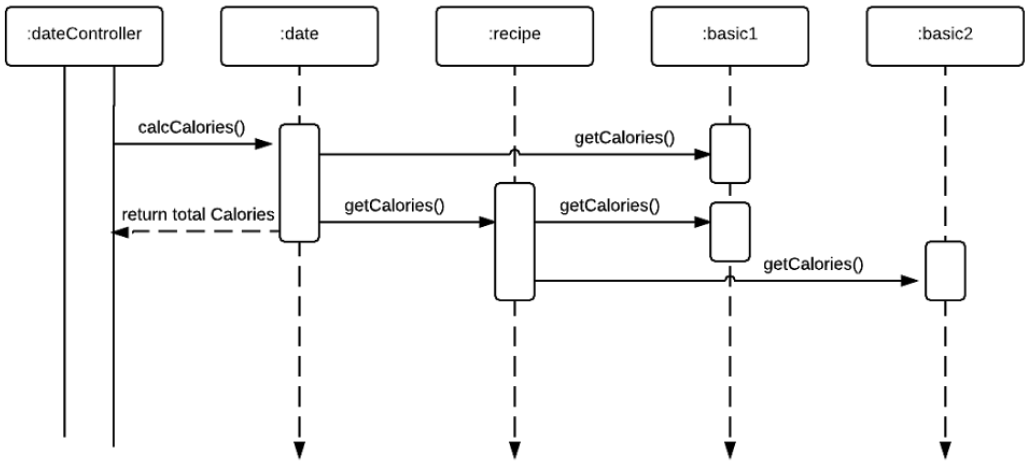
Sequence 2 - Creating BasicFood or Recipe Objects

User creates a new food or recipe object to be entered into the database


:foodController
:basicFood
:recipe
:foodCollection
readCSV()
new BasicFood()
add()
new Recipe()
add()

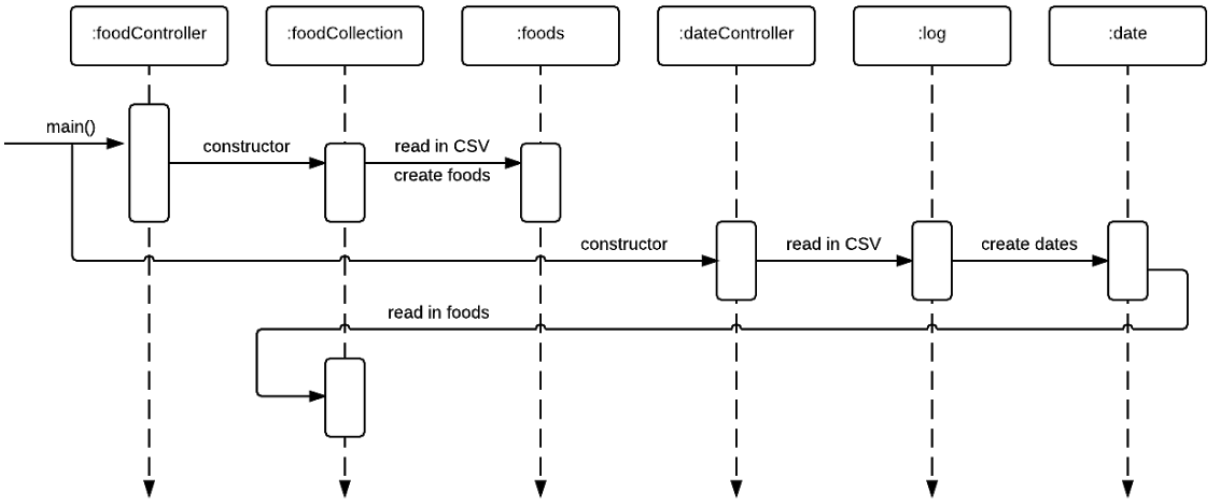
Sequence 3 - Calculating Calories by Date

Calculates the total calories consumed from the food entries for a given date



Sequence 4 - App Initialization

Loads in the current iteration of the data file and loads in the past date entries from the log



Pattern Usage

Pattern #1 MVC

The application is organized using the model view controller pattern to recognize commands to the controllers, manipulate the Log and FoodCollection as the models, and reflect changes via the views.

MVC Pattern	
Model	Log dateComponent Food BasicFood Recipe FoodCollection
View	DateView FoodView
Controller	DateController FoodController SaveStateController

Pattern #2 Composite

The FoodModel subsystem is organized using the composite pattern. Food is the interface in which the other classes inherit from. BasicFood objects are the leaves as they cannot contain any other objects. Recipes can (and will) consist of either BasicFood objects or other Recipes, which in turn are also made up of other BaiscFood or recipe objects and soforth.

Composite Pattern	
Composite	Recipe
Component	Food
Leaf	BasicFood