# Software Engineering

# YADA

# (Yet Another Diet Assistant)

# Unit-2 Project: *Design Document*

# Team 7


| Roll No | Name |
|---|---|
| *201001123* | *Abhay Pande* |
| *201101173* | *Harsh Vardhan Shukla* |
| *201001086* | *Madhavan Malolan* |
| *201350902* | *Neeraj Mathur* |
| *201001126* | *Saumya Dwivedi* |


## Project Overview:

The aim of the project was to build a Diet Management System , YADA( Yet Another Diet Assistant) , that allows users to :

- Add information about the different Food types consumed by the users.
- Keep track of their food consumption on a daily basis. This allows users to add and delete food items to their daily consumption logs allowing them to keep track of their calories consumed on a daily basis.
- Allow the user to Compute target calories on the basis of their height, weight, age and activity levels. This allows them to manage their diet on a daily basis and perform keep their consumption under check.

## Requirements Overview:

According to the requirements provided, The Diet Management System must allow the user to :

- Maintain a database for the consumption of Food Item. The food items can be Basic or Composite. A basic Food Item has  an identifying string, a set of keywords, and can have

a set of nutritional facts associated with it. Initially only the the calories are required to be kept track of, but the design can be expanded to keep track of other nutritional details like carbohydrates, fat content and so on..

- A Composite food is made up of other Basic or Composite Foods. The user must be able to create new Composite Foods by specifying the name and helping of existing food items. Like the Basic Food Items Composite Food items will also have an identifying string and a set of keywords. The Nutritional Characteristics of a Composite Food Item will be obtained by summing the nutritional details of the basic food items it is made up of.
- The System must store the list of both Basic and Composite Foods in a format that allows them to be read in a standard text editor. Also, the Food Items information must be loaded when the program start and saved when it closes. The user can explicitly save the Food Item details.
- The user interface must allow the user to add new food items, both basic and composite. The System must be easily expanded to allow Basic Food Item information to be downloaded from an external source like a diet database or a restaurant. The interface must also allow users to create Composite foods from existing Food Items by specifying their identifier and number of servings.
- The System maintain daily logs of Food Consumption. The format of the log should be such that it can be read in a text editor. The log must e loaded at the start of the program and saved on exit.
- The user must be able to add and remove Food Items to the consumption logs. The user can add Food Items either by selecting them from the whole list of foods or by narrowing them using keywords. The same food items can be added more than once on a day.
- The user must be able to undo commands to an infinite depth. The undo information is discarded between sessions. Also , the Consumption log for any day can be selected viewed and updated, not just the current date.
- The System must store User information like name, gender, height, weight, age and activity level. The latter three can be changed. By default the previous day's information must be carried forward. Based on this information, the user must be able to calculate the target calorie consumption. This allows the user to keep track of his daily consumption, whether he has exceeded it or not. The System must support multiple target calorie calculation strategies and must allow the user to change between them.

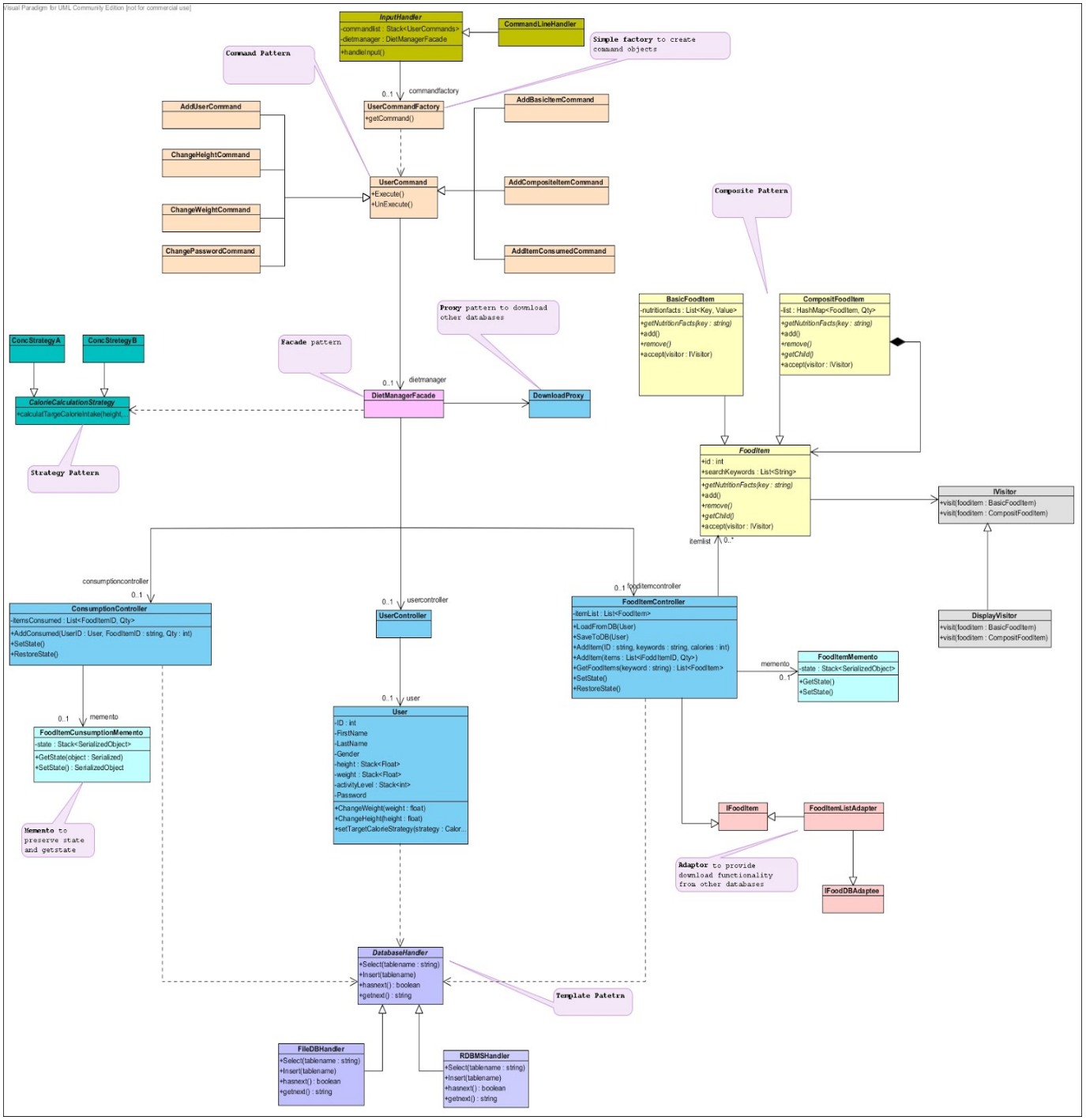
## Assumptions:

- We are accepting date of birth instead of age and calculating age from date of birth, User is allowed to change weight and activity levels
- Food item once created and used in the logs cannot be removed, however user can disable the item, this constraint is applied to maintain referential integrity of system
- System is assumed to be such that multiple users can log in from the same machine.

## System Overview:

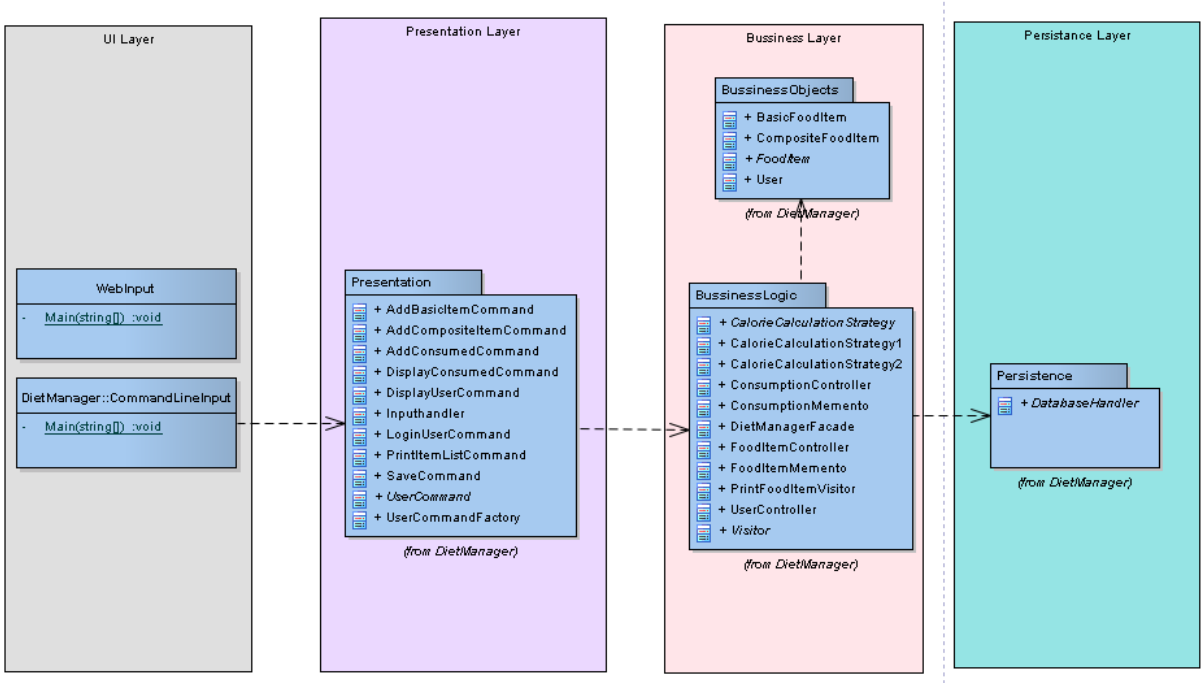
## **Class Diagram**(refer to file design/Unit2 Design.jpg)**:**

The following class diagram gives an overview of the design of the Diet Management System.



**Package/Layer Interaction diagram:**
Given below is the Package Interaction Diagram. The implementation of the Diet Management System is divided into four layers:

- **The UI layer:** This layer handles the interaction of the user with the system. The User is provided with a command line Interface that allows him to enter appropriate commands and interact with the system.
- **The Presentation Layer:** This layer contains the implementation of all the Commands that the Diet Management System provides. It also contains the inputhandler that maintains a stack of all the executed commands and provides the undo facility.
- **The Business Layer:** This Layer contains core system classes including the classes related to Food Items, Food Consumption, Users and Calorie Calculation Strategies. These class objects are used to implement the commands in the presentation layer.
- **The Persistence Layer:** This layer takes care of storing and retrieval of the Food Items, Food Consumption logs and the User Details. It provides the DatabaseHelper which helps in the storage and retrieval of data.

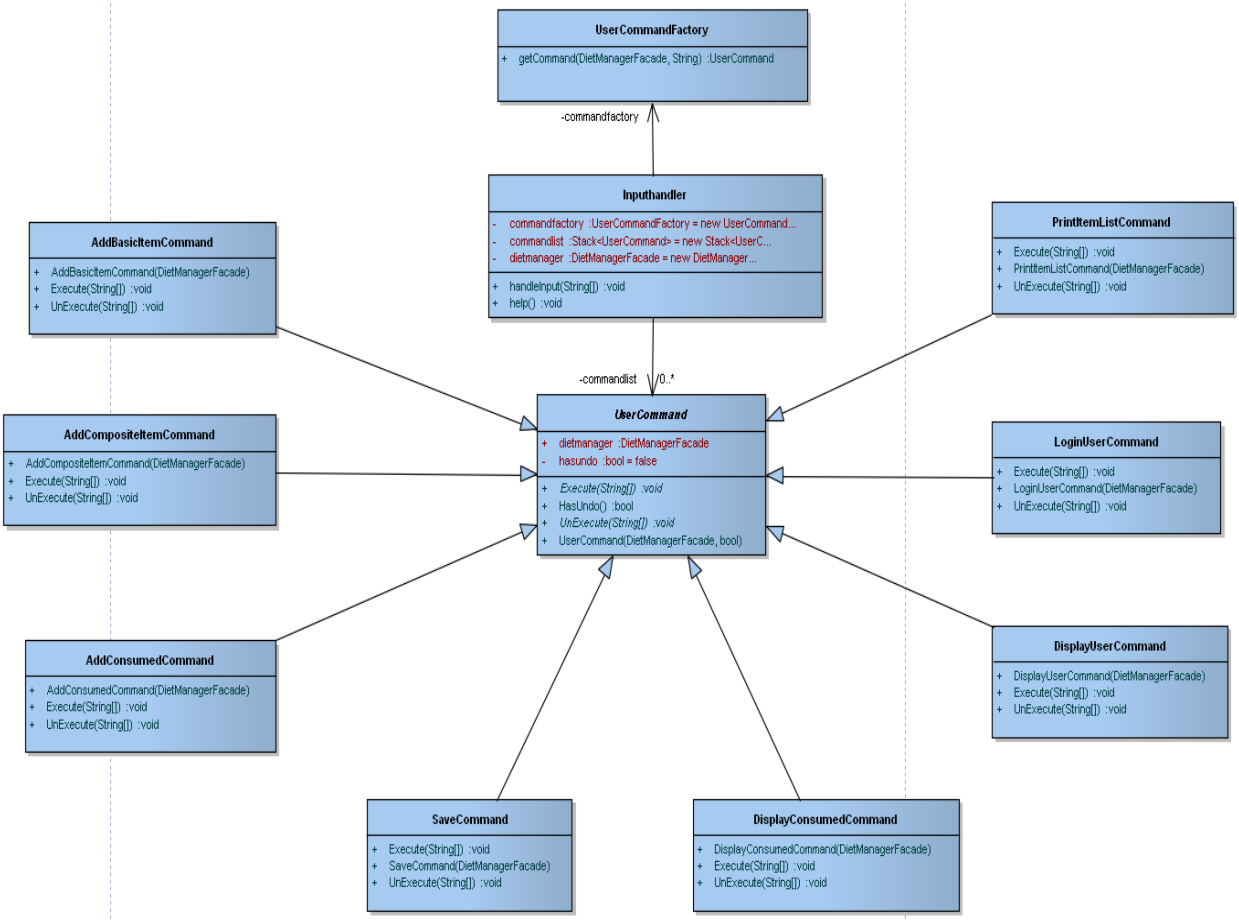



As the core functionality of the implementation of the Diet Management System lies in the Presentation layer and Business Logic layer, let us look at it in some more detail.

**Presentation Layer:**
The presentation layer provides the implementation of the commands that the Diet Management System provides. A look into this layer gives us details of all the functionality that the System provides. The presentation layer includes the implementation of the **Command Pattern** in order to implement the different commands that the user can run. The use of command pattern allows

us to implement the **undo functionality easily**. And a UserCommand **factory** class is used by input handler to create different command objects. Moreover, addition of a new command becomes relatively simple as integrating it into the system is simple.

- The User can add a new food item using the AddItem Command . AddItem <name> <search keywords> <calorie>
- The User can create a Composite food Item using the AddCompositeItem command. AddCompositeItem <name> <search keywords> <identifier,qty> <identifier,qty> ...
- The user can change the date to any specified date. This will load the Consumption logs for that particular date if they are present. SetDate <date Format='DD/MM/YYYY'>
- The user can view a list of all food items preset using the PrintItems Command. By providing an optional list of search keywords, only those items that have these keywords are shown, PrintItems <SearchKeywords:optional >, seaerch keywords is optional if not supplied then all enabled items will be listed
- The user can add a consumption record to the logs by using the AddConsumed Command. This consumption log is added to the log of the selected date value. By specifying a negative value in quantity the said item can be deleted or decrease servings. AddConsumed <identifier> <Quantity>
- The PrintConsumed command can be used to vie the Consumption for a selected date.
- The PrintUserInfo command is used to proint the details of the current user of the system
- The Commands ChangeWeight and ChangeActivityLevel are used to change the weight and ActivityLevel of the Current User.
- The Undo command is used to undo the last command. Only AddItem, AddCompositeItem and AddConsumed Commands can be undone.
- The Save Command is used to explicitly save the FoodItems, User information and FoodConsumption details of the system.
- The SetCalcStrategy is used to set the target calorie calculation strategy. Currently two strategies are used. One that only takes your activity level into consideration while the other that also calculates your BMR.SetCalcStrategy <1=Normal, 2=BMI>
- The Exit command is used to exit the application.
- The Disable Command is used to Disable FoodItems from the FoodItems list. It allows us to disable items so that they would not be shown as part of the Food Items list.

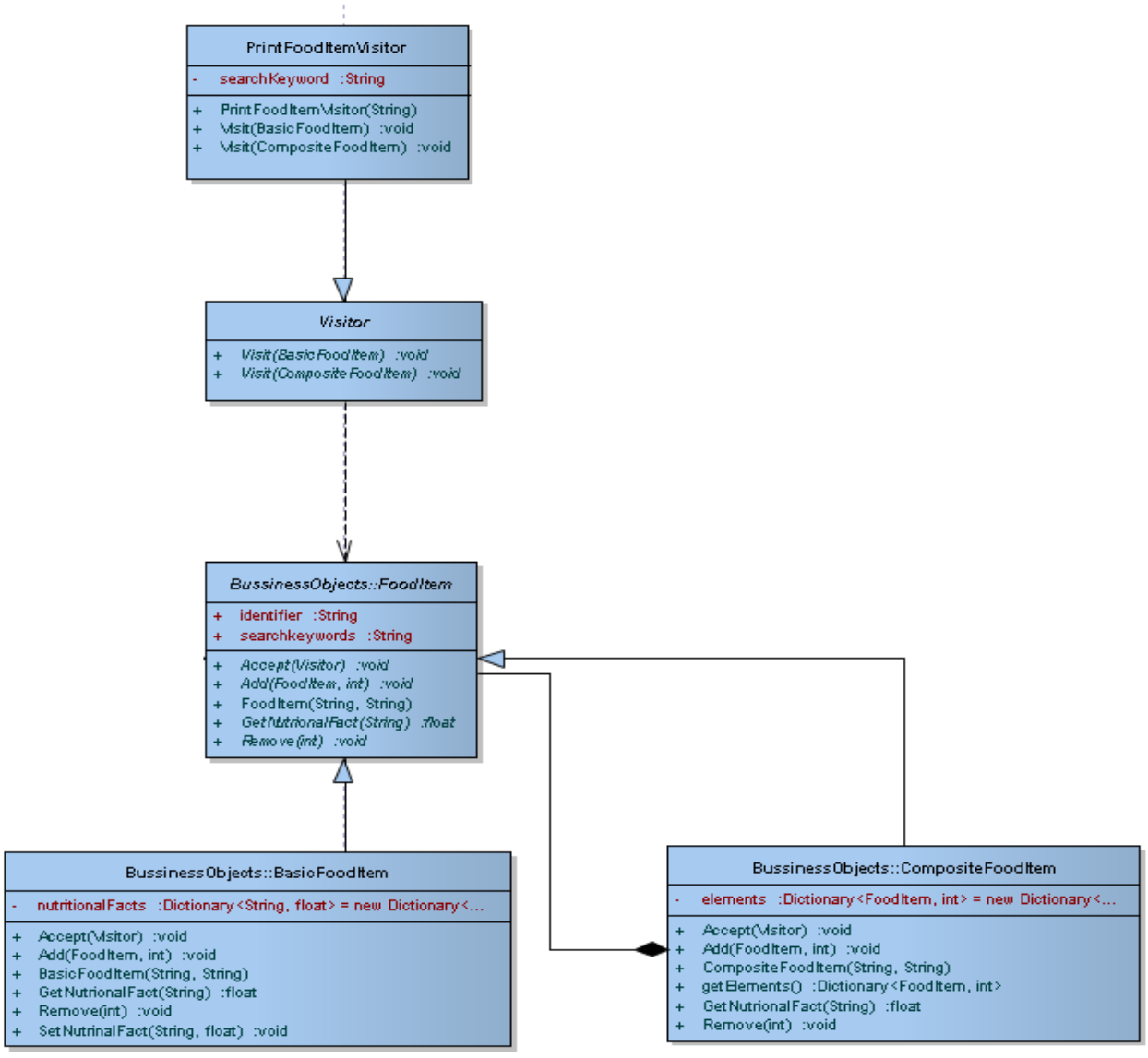


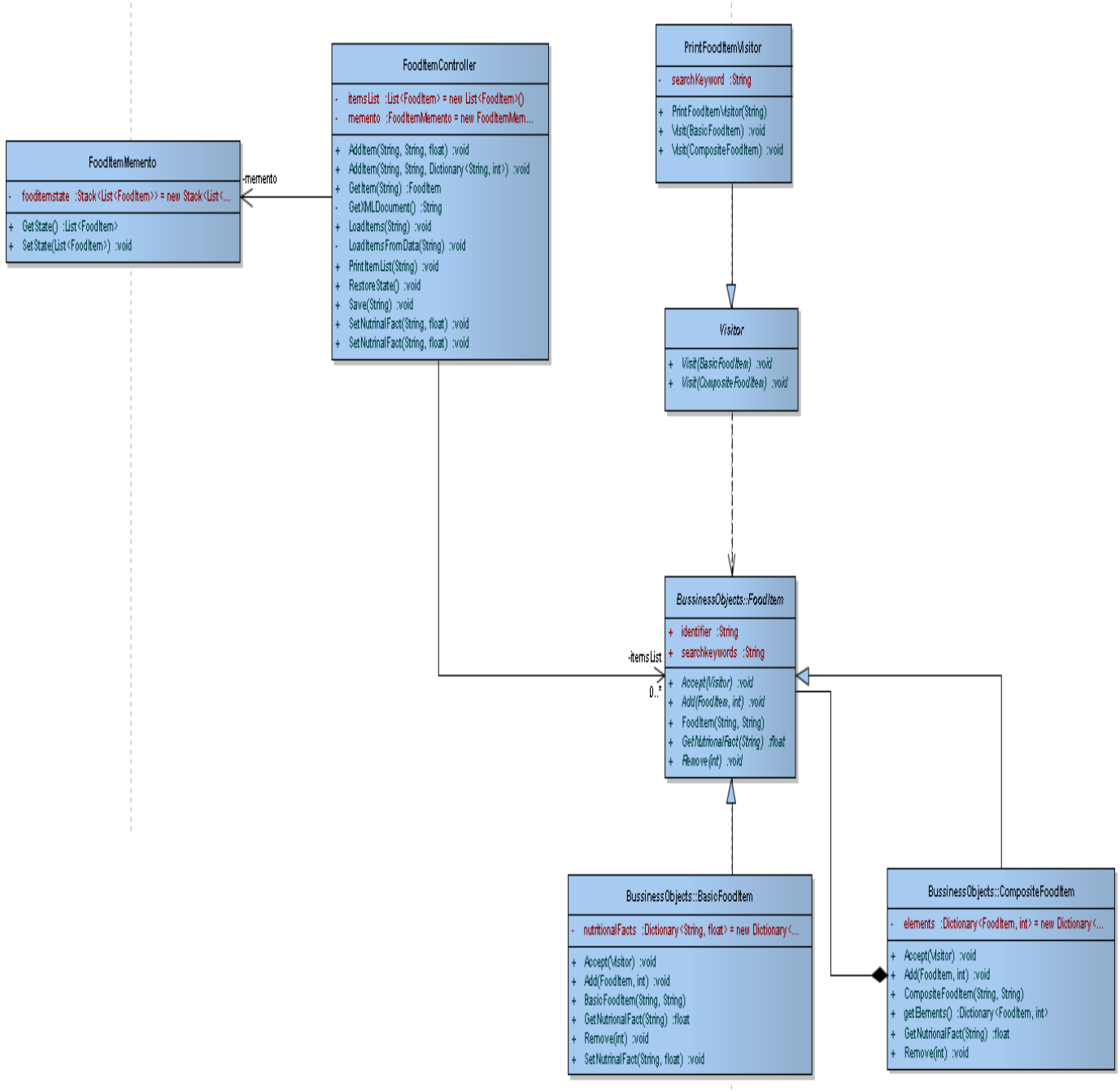

**Business Layer:**The business layer contains the implementation of the classes that help us implement the bussiness logic of the Diet Management System. These include the classes used to implement the FoodItems and the Consumtion logs along with the user details and the calculation strategies for the target calories consumption.

- The Classes FoodItem, BasicFoodItem and CompositeFoodItem together represent the composite that makes the food items. The **Composite Pattern** was used because it allows us to treat both Basic and Composite food items in the same manner. This allows us ease in implementation and usage.
- The classes ConsumptionController and ConsumptionMemento keep track of the state of the consumption logs. They allow us to add to and remove from the Consumption logs, load from and save to memory the Consumption details. The ConsumptionMemento remember the state of the Consumtion logs in order to help implement the undo Functionality.
- The Visitor class implements a visitor over the FoodItem class. Currently, the **visitor** is used only to print the FoodItem details. However, the Visitor can be easily extended to perform other tasks on the FoodItems thus allowing extensiblity.

- The FoodItem Controller allows the addition of and removal of new food Items. It manages the loading and Storage of Food items on starting and exiting the applications respecttively. Together with the FoodConsumption controller  and the UserController class which manages the User specific control information that allows the weight and activity level of the user to be changed, the FoodItem controller is used by the
- Dietmanager **Facade** classs to provide an external and less complicated interface to the Presentation Layer and also hides the implementation details thus reduces coupling .
- The **Strategy pattern** is used to implement the target calorie consumption strategies. Use of strategy pattern helps decouple the system from the consumption algorithm, thus reducing coupling. This also allows for extensibility as new strategies can be introduced without causing any ripples in the system.

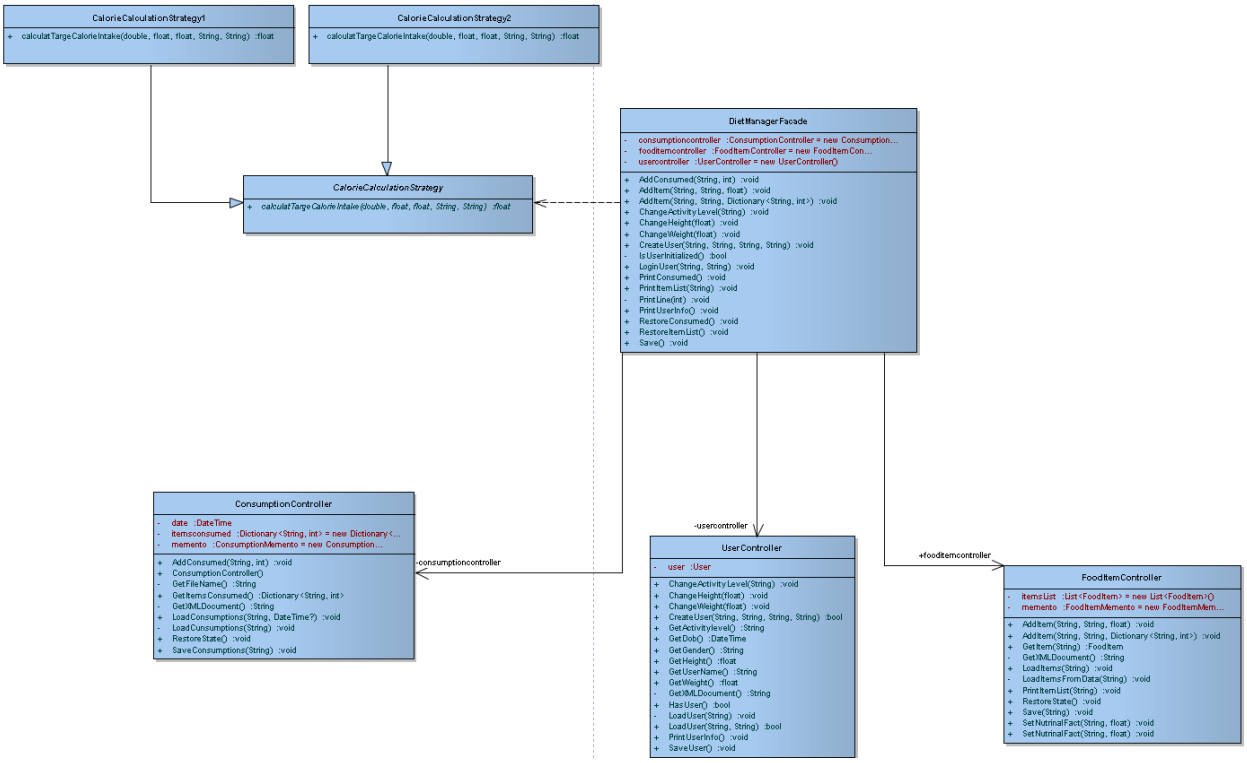
**The FoodItem Composite along with the Visitor**





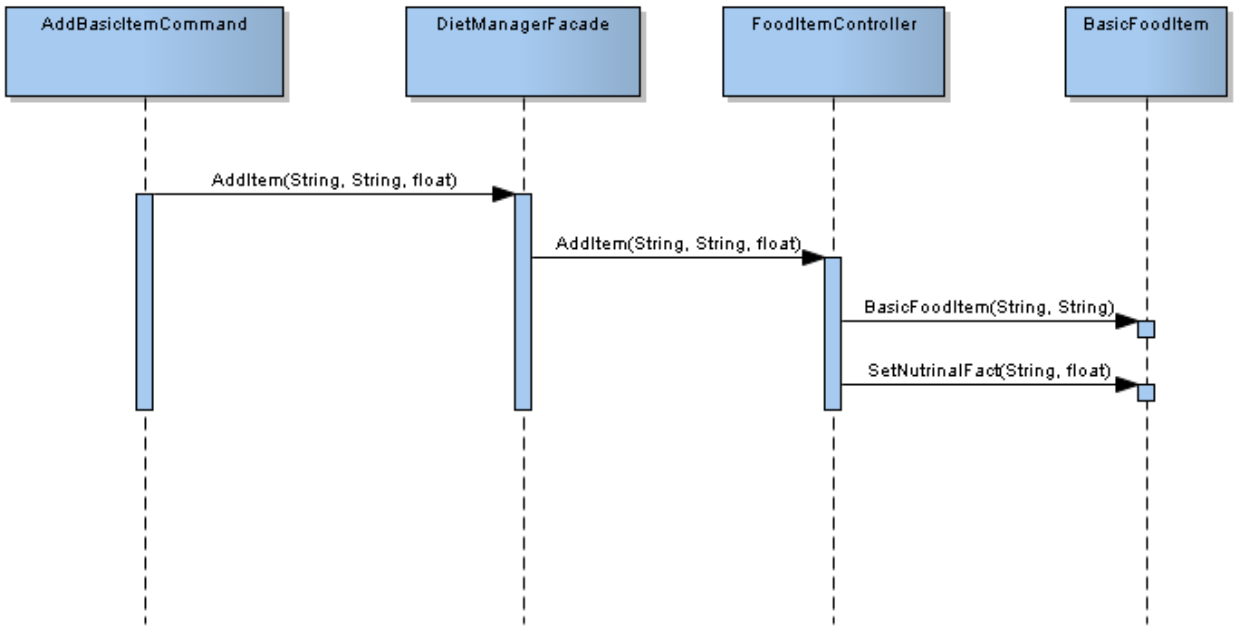

**The FoodItem Controller and Memento object**

**Diet Manger Facade Interaction:**



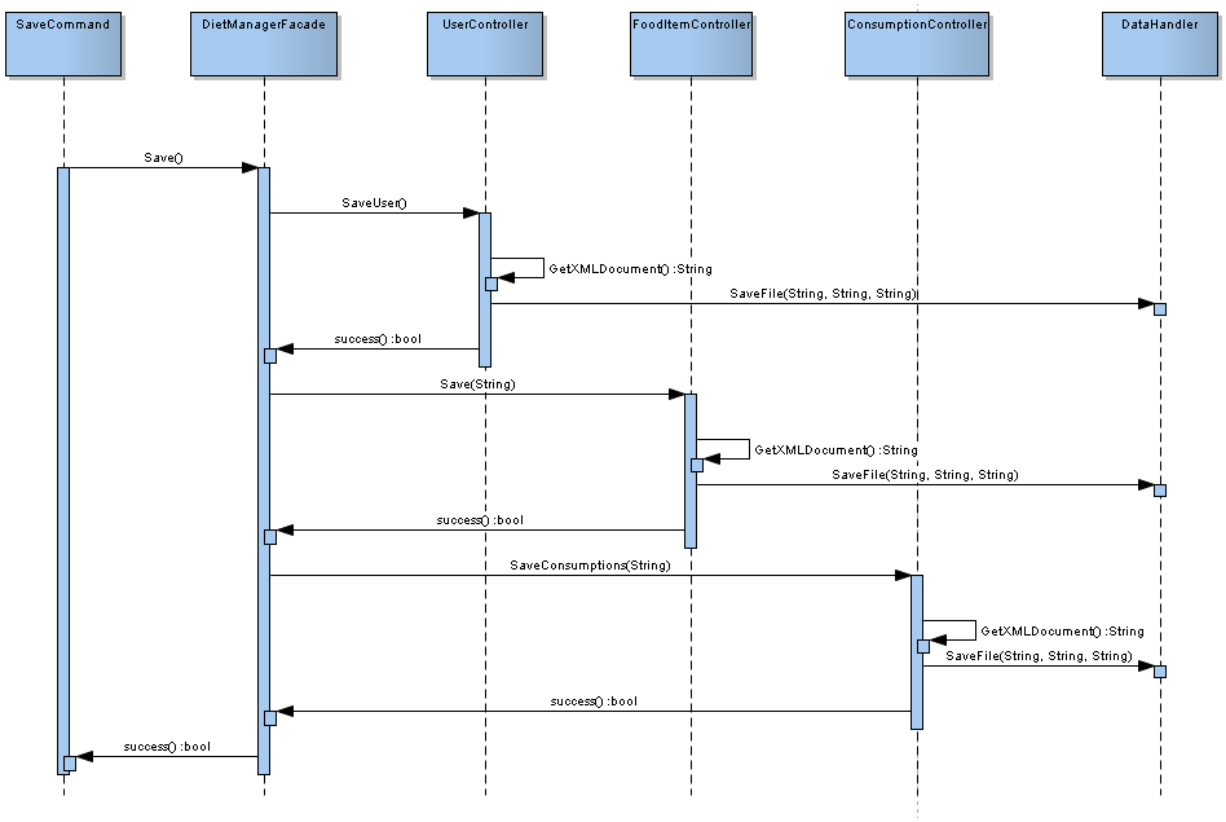

In order to better understand the working of the system, let us see how the following commands work.

- Adding a Basic Food Item
    - When the user gives the Command to add a new basic food Item, the AddBasicCommand object calls the AddItem method of the DietmanagerFacade
    - The DietmanagerFacade in turn calls the FoodItemController's AddItem method to add the basic food item.
    - The FoodItemController creates a new BasicFoodItem and sets the nutritional values as provided by the user.



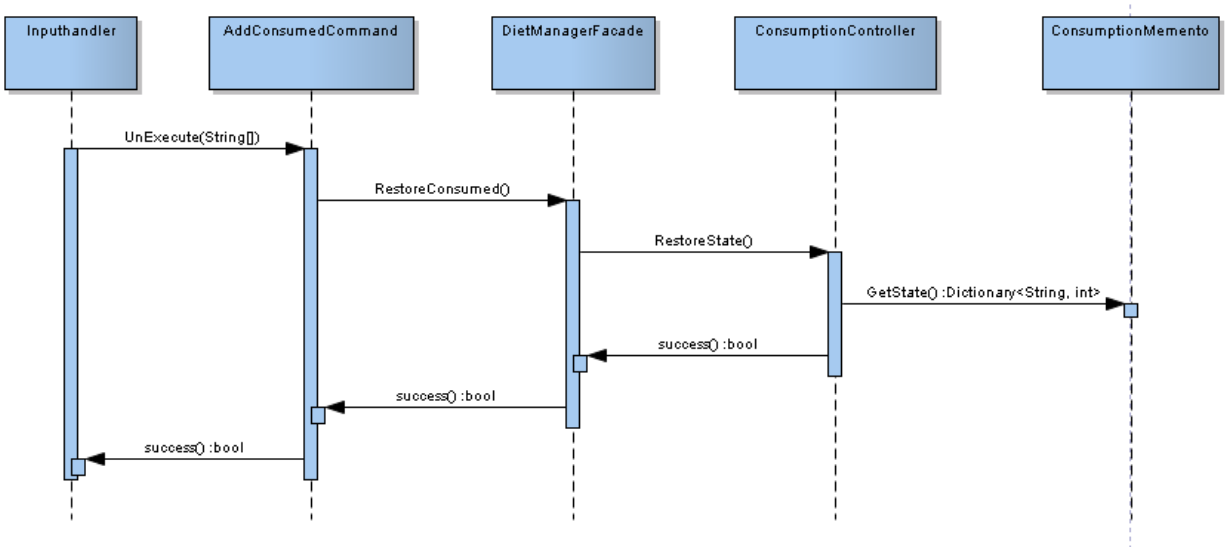

*Adding a basic food item*

- Running the save command
    - The SaveCommand object invokes the save command of the DietmanagerFacade object.
    - The DietmanagerFacade uses the UserController's SaveUser() Method. This in turn get the xml file corresponding to the current user and inturn calls the SaveFile method of the DataHandler object. A successful return status is returned to the DietmanagerFacade.
    - The DietManagerFacade also calls the save() method of the FoodItemController object. This method in turn obtains the filename corresponding to the fooditems for the current user and calls the Savefile method of the DataHelper. The DietmanagerFacade receives a status of the saving process.
    - The DietmanagerFacade calls the save method of the FoodConsumption object which in turn finds the cile in which the details are to be saved and passed it to the DataHelper's SaveFile method. The DietmanagerFacade receives a status of the save .




*Sequence diagram for saving*

- Running the Undo command to undo consumption
    - The inputhandler pops the AddConsumptionCommand object at the top of its tack of commands and calls the unexecute() method.
    - In the unexecute() method the RestoreConsumed of the DietManager Facade is called.
    - The RestoreConsumed function in turn calls the RestoreState() method of the Consumtion controller. The Consumption controller gets the state from the consumption memento and restores the state. The success is notified to the DietManagerFacade




*Undo of the AddConsumption command*

## Class Table With Responsibilities

| **Package** | **Class** | **Responsibility** |
|---|---|---|
| **Persistence** | **DatabaseHandler** | This abstract class which provides template methods to accessing persistent objects from file/DB |
| | **FileDBHandler** | concrete implementation of DatabaseHandler class for XML files |
| **BusinessObjects** | | |
| | **FoodItem** | Abstract food item which has identifier, search keywords properties and Add,Remove,Accept visitor and GetNutritional facts we are using Key,Value pair for nutritional facts so it is easy to add new types of nutritional facts |

| | | |
|---|---|---|
| | **BasicFoodItem** | Responsible for basic food item element nutritional facts are attached to it |
| | **CompositeFoodItem** | Composite item is made up to basic item, the nutritional facts of a composed item is calculated from it basic items |
| | **User** | contains user properties , username, gender, DOB, height, weight, activity level |
| **BussinessLogic** | | |
| | **UserController** | This class has a object of BusinessObject.User it controls behaviour of user object. Save user details to XML file on system exit or explicitly on sav command |
| | **FoodItemController** | Class provides methods to add,change and disable food items, it contains all the list of items present in system. Save it to XML file on system exit or explicitly on Save command |
| | **FoodItemMemento** | Saves food item changes which are used to restore the state of food items for undo command |
| | **ConsumptionController** | It contains the list of items consumed by user by List<key,value>, it exposes methods to add consumed items, change consumed items, restore consumed item state |
| | **ConsumptionMemento** | Saves consumed item changes which are used to restore the consumptions for undo command, it preserves stack of changes made to the consumption list |

| | | |
|---|---|---|
| | **Visitor** | Abstract food item visitor |
| | **PrintVisitor** | Concrete print visitor used to display food item details |
| | **DietManagerFacade** | Single class which encapsulates all controllers and exposed to presentation layer |
| | **CalorieCalculationStrategy** | Abstract class for calculating calorie intake |
| | **CalorieCalculationStrategy1** | Concrete implementation of Normal based calorie intake calculation |
| | **CalorieCalculationStrategyBMI** | Concrete implementation of BMI based calorie intake calculation |
| **Presentation** | | |
| | **Inputhandler** | This class handles different UI inputs and converts UI input to an appropriate command cobject |
| | **UserCommand** | Abstract user command containing Execute and UnExecute methods |
| | **UserCommandFactory** | Given an identifying string, this creates the corresponding Command object and returns it to the user. |
| | **AddBasicItemCommand** | A class whose object helps add a new basic food item in the execution of its execute method |
| | **AddCompositeItemCommand** | A class whose object helps add a new composite food item in the execution of its execute method. |
| | **AddConsumedCommand** | A class whose object helps add a new consumption entry for th |

| | | |
|---|---|---|
| | | selected day in the execution o its execute method. |
| | **ChangeActivityLevelComma nd** | A class whose object helps change the activity level for the current user and selected date in the execution of its execute method. |
| | **ChangeWeightCommand** | A class whose object helps change the weight for the current user and selected date in the execution of its execute method. |
| | **CreateUserCommand** | A class whose object helps create new user for the system |
| | **DisplayConsumedCommand** | A class whose object helps display the consumption logs fo the current user and selected date in the execution of its execute method. |
| | **DisplayUserCommand** | A class whose object helps display the details of the current user on the selected date in the execution of its execute method. |
| | **LoginUserCommand** | A class whose object helps set the current user and session related information. Provides protection by enabling login. |
| | **NullCommand** | Checks whether the entered command is a valid command or simply a null entry. |
| | **PrintItemListCommand** | This command object prints the list of food items for the given user in the implementation of it execute() method. |
| | **SaveCommand** | This command object allows us to save the Food Items and the |

| | | |
|---|---|---|
| | | Consumption log as well as the User details explicitly on the execution of its execute() method. |
| | **SetCalorieCalcStrategy** | This command object is used to set the calculation strategy being used to calculate the target consumption. |
| | **SetDateCommand** | This class object helps set the date for which the consumption log and user information are considered. |

## Code Metrics

Given below are the details of the performance of the code under the different metrics.

| **Metrics Name** | **Value** |
|---|---|
| Code Size | 34816 |
| Types | 38 |
| Abstracts | 4 |
| EfferentCouplings | 21 |
| AfferentCouplings | 19 |

| **Name** | **Size** | **Weighted Cyclomatic Complexity** | **Fields** | **Methods** |
|---|---|---|---|---|
| AddBasicItemCommand | 94 | 3 | 0 | 3 |
| AddCompositeItemCommand | 185 | 4 | 0 | 3 |
| AddConsumedCommand | 83 | 3 | 0 | 3 |
| BasicFoodItem | 91 | 6 | 1 | 6 |

| | | | | |
|---|---|---|---|---|
| CalorieCalculationStrategy | 7 | -1 | 0 | 2 |
| CalorieCalculationStrategy1 | 288 | 7 | 0 | 2 |
| CalorieCalculationStrategy2 | 18 | 2 | 0 | 2 |
| ChangeActivityLevelCommand | 68 | 3 | 0 | 3 |
| ChangeHeightCommand | 74 | 3 | 0 | 3 |
| ChangeWeightCommand | 74 | 3 | 0 | 3 |
| CommandLineInput | 242 | 5 | 0 | 2 |
| CompositeFoodItem | 161 | 7 | 1 | 6 |
| ConsumptionController | 710 | 21 | 3 | 9 |
| ConsumptionMemento | 209 | 6 | 1 | 3 |
| CreateUserCommand | 72 | 3 | 0 | 3 |
| DataHandler | 174 | 4 | 1 | 3 |
| DietManagerFacade | 1136 | 32 | 3 | 17 |
| DisplayConsumedCommand | 60 | 3 | 0 | 3 |
| DisplayUserCommand | 60 | 3 | 0 | 3 |
| FoodItem | 24 | -1 | 2 | 5 |
| FoodItemController | 1663 | 28 | 2 | 10 |
| FoodItemMemento | 136 | 4 | 1 | 3 |
| Inputhandler | 312 | 8 | 3 | 3 |
| LoadDataCommand | 16 | 3 | 0 | 3 |
| LoginUserCommand | 66 | 3 | 0 | 3 |
| NullCommand | 38 | 3 | 0 | 3 |
| PrintFoodItemVisitor | 299 | 7 | 1 | 3 |

| | | | | |
|---|---|---|---|---|
| PrintItemListCommand | 78 | 4 | 0 | 3 |
| SaveCommand | 28 | 3 | 0 | 3 |
| User | 673 | 33 | 7 | 21 |
| UserCommand | 43 | -1 | 2 | 4 |
| UserCommandFactory | 412 | 6 | 0 | 2 |
| UserController | 1888 | 29 | 1 | 17 |
| Visitor | 7 | -1 | 0 | 3 |

## Default system values:

### Default food item list provided:

| Identifier | Keywords | Calories | Type | Enabled |
|---|---|---|---|---|
| sugar | sugar | 200 | Basic | True |
| flour | flour | 2000 | Basic | True |
| milk | milk | 10 | Basic | True |
| bread | bread | 24000 | Composite | True |
| apple | apple | 28 | Basic | True |
| banana | banana | 61 | Basic | True |
| orange | orange | 21 | Basic | True |
| sweetpotato | sweetpotato | 100 | Basic | True |
| butter | butter | 102 | Basic | True |
| carrots | carrots | 52 | Basic | True |
| cofee | cofee | 2 | Basic | True |
| egg | egg | 102 | Basic | True |

| | | | | |
|---|---|---|---|---|
| rice | rice | 205 | Basic | True |
| dates | dates | 502 | Basic | True |
| lemon | lemon | 17 | Basic | True |
| papaya | papaya | 117 | Basic | True |

**Default consumed values:**
<<<--- Items Consumed on 18/03/2014 , Strategy=Normal --->>>

| **Item** | **Servings** |
|---|---|
| sugar | 5 |
| bread | 40 |
| milk | 10 |

TOTAL CALORIE(S) CONSUMED: 961100
TARGET CALORIE(S) INTAKE: -956049

**Default user info:**
---
<<< User Information >>
Username :team7
Gender :male
Date of Birth :27/05/1983
Weight(KG) :73
Height(FEET) :80
Activity Level :High

## Conclusion(Positive and Negative points):

1. The implementation satisfies all the functional requirements of the DietManagement system as specified in the requirements.
2. An advantage of the system is that it has a layered architecture. Thu changes in one layer affect changes only in the adjacent layer. This helps in reducing coupling and increases maintainability.
3. Moreover, the logic is so implemented that it can be extended to a distributed environment, i.e multiple users can open up multiple sessions in different machines.

Disadvantages

1. The system design incorporates a large number of GOF design patterns , namely Composite, Strategy, Command, Memento, Facade and Visitor. Because of the large number of design patterns used it might become difficult to refactor code or extend it in the future, making it difficult to manage. The difficulty arises because design patterns do enforce some semantics in the design which need to be followed. A naive attempt to refactor and maintain the code might make the system complicated and unmanageable.
2. One drawback is the use of files as a persistence mechanism as it involves unnecessary complications in correctly reading and writing to consumption and food logs into them. It is in an attempt to reduce this complication that we have chosen to create a new consumption log file for the each day for each user. While this simplifies and naturally avoids concurrency issues, it leads to an exponential increase in the number of files.