COMP3004 Delivery 3

Team Name: Happy3004

Project Name: The Caf Calorie Tracker

Team Members:

Zhaohao Li	101020144	Ziyang Zhou	101049422
Xiaofeng Luo	101007579	Yejing Li	101056554

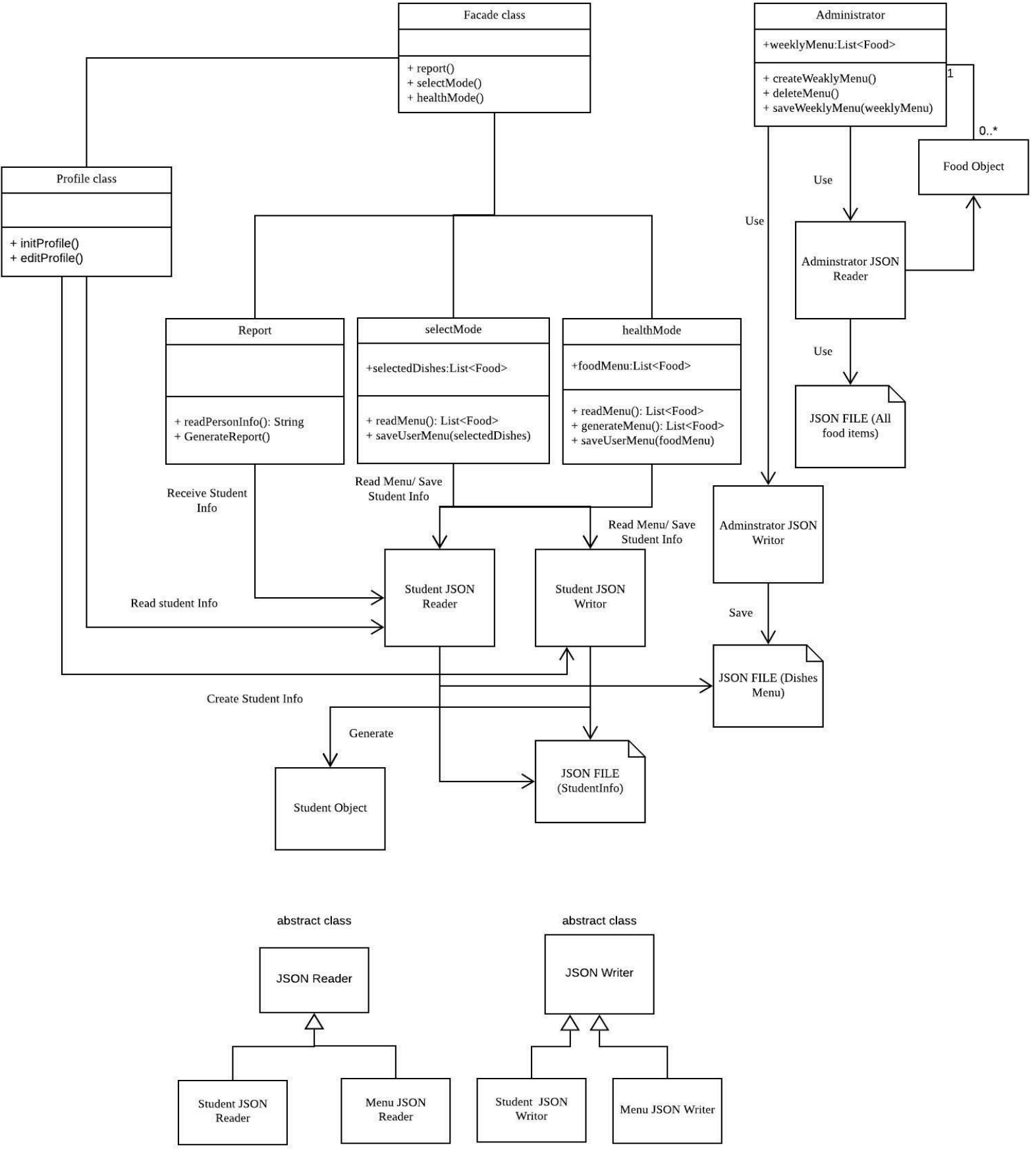
Part 1: Architecture Design

The Caf Calorie Tracker, the most appropriate and vital architecture design is the object-orientation style (OO style), which is based on the compiled language of this project. The Caf Calorie Tracker is launched on the Android platform. Android Applications usually use object-oriented languages Java or Kotlin as the programming language. Hence, choosing object-orientation style architecture is the most suitable way to begin this program. This Java-based architecture determines all the designed functionalities should be supported in Java and all the implementation details will be hidden.

In the object-orientation style, all of the activities in this project must be encapsulated into different classes. Every identified class is needed for this application. All of them are assigned specific responsibilities and collaborate with each other in order to implement the system functions. Figure-1 is the detailed UML diagram for the object-orientation architecture in The Caf Calorie Tracker that presents all the classes implemented in this project. As the figure demonstrates, there are two basic objects that have been initialized and used. One is Food Object that stored all the information about the dishes including name, serving size, calorie, sugar and other nutrition. Another is Student Object that stored all student users' information including name, password, gender, BMI and behaviours. In addition, students' daily records of nutrition intake would also be stored into Report Objects, which enables users to view the summary for their intakes. A JSON files in the local describe user information and food items, which can by two extended JSON Reader (Student Reader and Administrator Reader). Also, these JSON files can be written by JSON Writers. Both of them are the cornerstone of the system, supporting all functionalities including user login, registration, manipulate food menu, etc. There are some use cases where objects will be used:

1. One of the functionalities mentioned in the proposal is that the administrator is allowed to create a weekly menu. When a user logs in as an administrator, a list of Food objects is present. The administrator can choose which to be included in the weekly menu. Having confirmed the new weekly menu, a new JSON file is created and published to the students' home page.
2. When users log in as students for the first time, the student object will be asked to fill a personal information form. User information would be parsed by Student JSON Writer. Then for the next time, when a user tries to log in, the application will check if there is a corresponding JSON file for the user. And then the application would attempt log in using the corresponding username and password provided by the user.

In these cases listed above, all of the processes are executed in separate and encapsulated classes, hiding application details from clients. Consequently, there will be no effects on clients even though the implementation is changed. This feature allows our team to maintain the application, to extend our application without affecting any object.


Facade class
+ report()
+ selectMode()
+ healthMode()
Administrator
+weeklyMenu:List<Food>
+ createWeaklyMenu()
+ deleteMenu()
+ saveWeeklyMenu(weeklyMenu)
1
0..*
Food Object
Profile class
+ initProfile()
+ editProfile()
Use
Use
Adminstrator JSON Reader
Use
JSON FILE (All food items)
Report
+ readPersonInfo(): String
+ GenerateReport()
selectMode
+selectedDishes:List<Food>
+ readMenu(): List<Food>
+ saveUserMenu(selectedDishes)
healthMode
+foodMenu:List<Food>
+ readMenu(): List<Food>
+ generateMenu(): List<Food>
+ saveUserMenu(foodMenu)
Receive Student Info
Read Menu/ Save Student Info
Read Menu/ Save Student Info
Adminstrator JSON Writor
Read student Info
Student JSON Reader
Student JSON Writor
Save
JSON FILE (Dishes Menu)
Create Student Info
Generate
Student Object
JSON FILE (StudentInfo)
abstract class
JSON Reader
abstract class
JSON Writer
Student JSON Reader
Menu JSON Reader
Student JSON Writor
Menu JSON Writer

Another important architectural style for The Caf Calorie Tracker is the client and server architecture. Client and server architecture help better clarify the principal and subordinate processing and distinguish the order of function calls.

The client-server architecture style is the most commonly used architecture style in this program. It is not only used in talking to google authentication servers but also behaved in the communication between layers. There are three basic layers in the architecture which are called user layer, mode layer and data layer, and they are clearly organized providing services to their upper layers. However, in this program, the layered system is not very strict that the upper layer can access any lower layers. There are several cases where the program uses client-server architecture:

1. The Calorie Tracker supports users to login with their google account and if they do, the application needs to communicate with the Google authentication server. The credential will be checked and finally, a corresponding account will be created and stored in the local.

2. Users who login in as an administrator are able to generate the weekly menu. When the administrator enters the menu edit interface, all available dishes in the Caf will be read by a menu JSON reader and displayed in the interface. After the administrator selects dishes and creates a weekly menu, the new weekly menu will be stored in a Food list and written into a JSON file. Students can view the weekly menu instead of the original menu database.

3. Student users who keep using this app for at least seven days are able to view the weekly report. Each time when students use health mode or self choosing mode, the calories for all selected dishes/confirmed health menu will be recorded locally into a JSON file. Therefore, when students are viewing the report interface, record files will be read by the student JSON reader.

In these cases listed above, the first case that using Google API to process authentication uses a remote server. Similarly, the local log in also uses a client-server architecture. While logging in as a local user, the data layer provides services and data to another two upper layers, which is also considered as a server in design. All users (including layers) need to do is ask for a server for data or functions. To see our application as a whole, by doing this determines student inputs as client-side behaviours, while the application acting as a server to transmit user requests. Therefore, the client-server architecture is necessary and powerful in The Caf Calorie Tracker. Figure-2 is a detailed component diagram for the client-server architecture in The Caf Calorie Tracker.

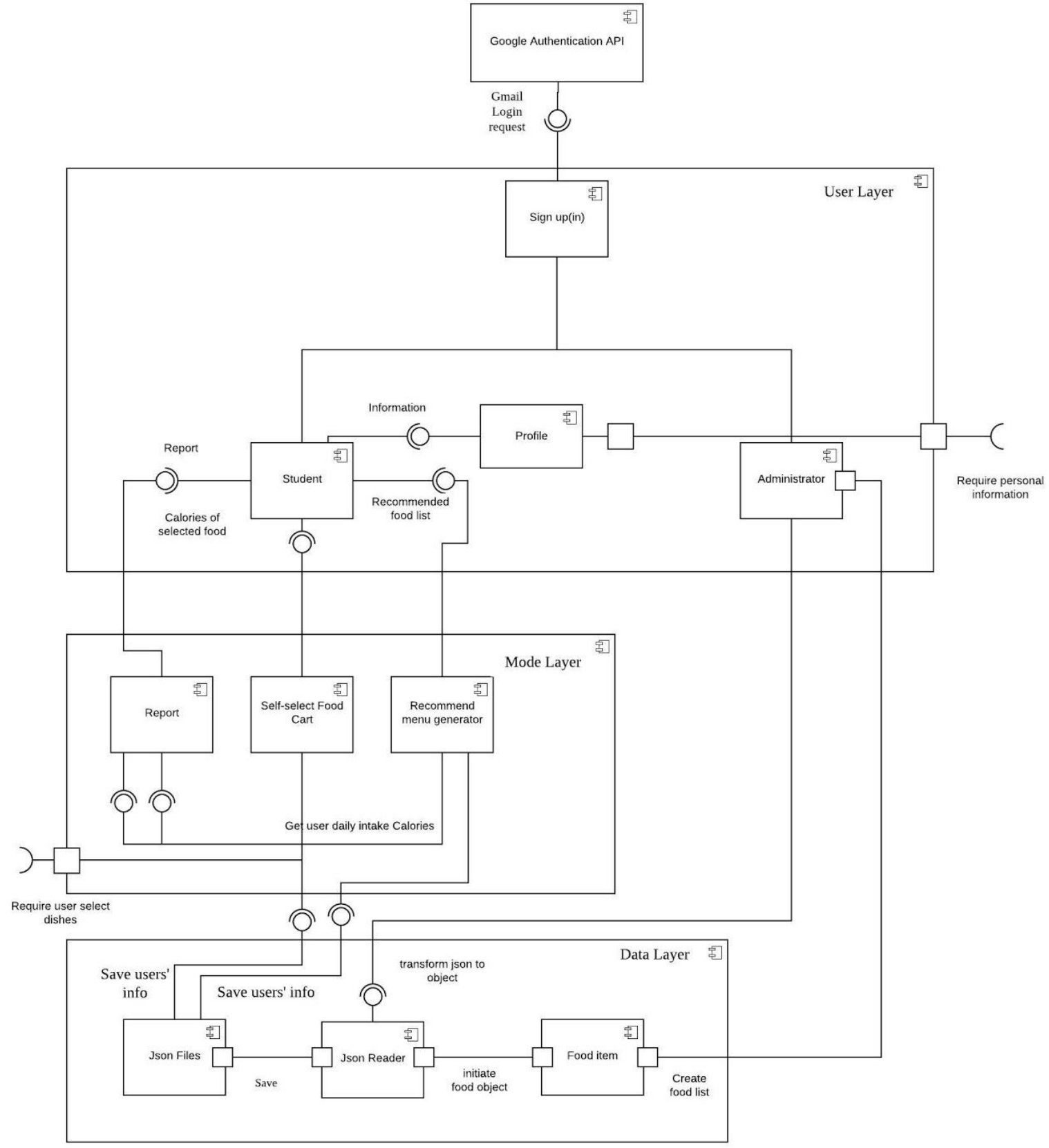


Figure-2: Component diagram for the client-server architecture

Part 2: System Design

Design Pattern 1

Facade

In The Caf Calorie Tracker, the key features that users will use are under student account. These functions are report generator in weekly report mode, calorie calculator in self-select mode and the menu generator in a health mode. All of them are encapsulated in separate classes and have their own interfaces that are able to interact or be displayed to users. In order to better manage and easier operate the subsystem, a facade pattern is an ideal choice for this design, which will provide a higher-level interface for masking these functions. Figure-3 is the basic UML diagram design for the Facade pattern.

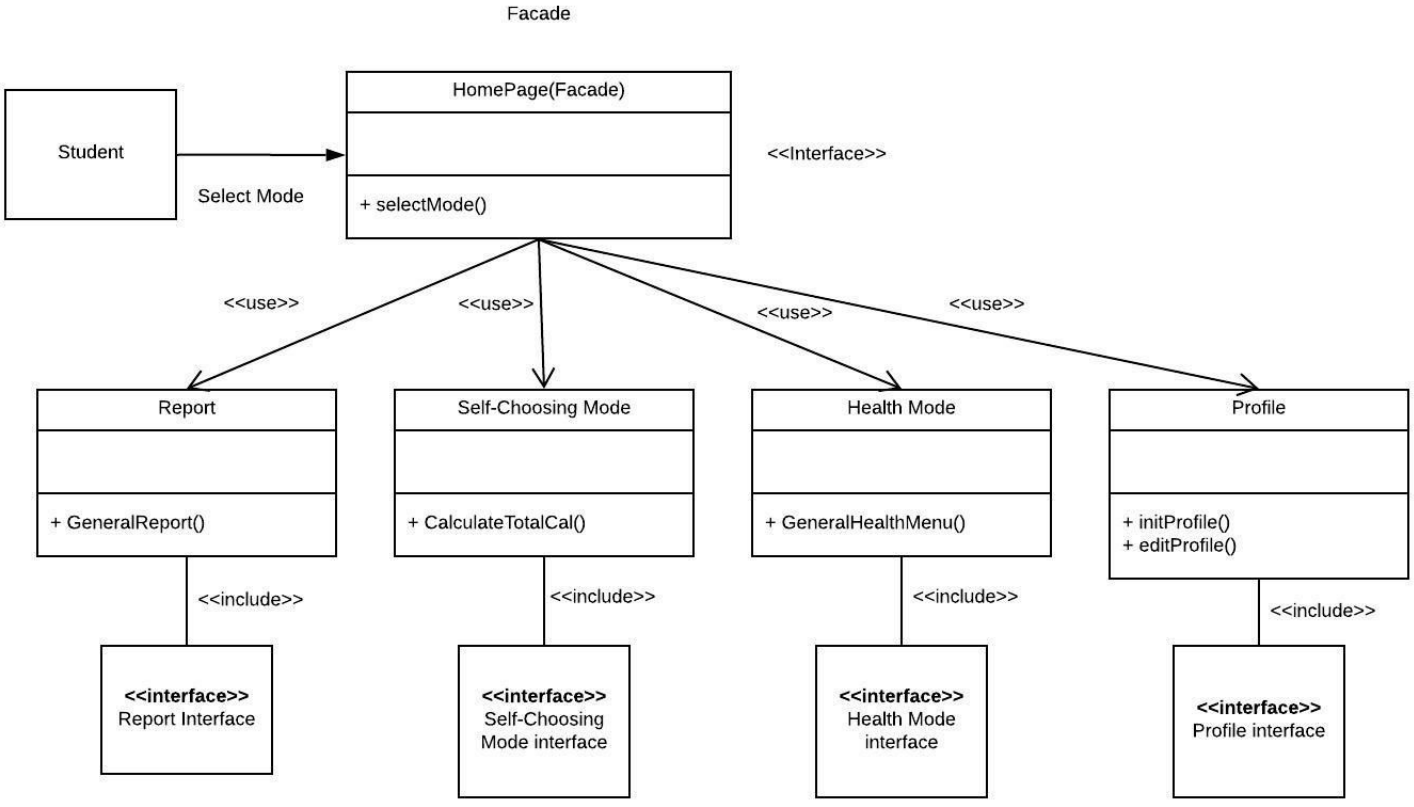


Figure-3 The basic UML diagram for the Facade pattern.

As Figure-3 shown, The Home Activity class plays the role of a Facade, hiding the class including Report, Self-Choosing Mode, Health Mode, Profile classes. The façade pattern provides access only to the four public features offered by the subsystem and hides all other details according to the principle of encapsulation. The Home Activity class is designed to present the user with a unified interface that allows them to switch to the desired functionalities after they log in as student users, which can reduce a set of interfaces in the subsystem and better present capabilities in student accounts. Moreover, it solves the problem of oversized

communication between subsystems and dependencies. The Facade pattern effectively shields users from subsystem components. In addition, Facade pattern weakens coupling between the client (upper-layer) and the subsystem by reducing the amount and complexity of dependencies.

Design Pattern 2

Factory Method Pattern

Another important design pattern implemented in this project is the factory pattern. In this application, there is a lot of information that required to be read from JSON file database. JSON file became the carrier of data. And obviously there is a demand of JSON file readers to parse the data into Food objects and Student objects in object-oriented architecture style. Hence, we decide to implement the factory method design to “produce” Food objects and Student Objects. In Factory Method Design, we will implement a demo called FactoryPatternDemo to send request to JsonReaderFactory. JsonReaderFactory plays the role of creator in the Factory design pattern which will create various types of JsonReader as product.

The product created by JsonReaderFactory can be described as two major types: MenuJsonReader and StudentJsonReader. We can call them product 1 and product 2 in order to make it easier to remember. MenuJsonReader is designed for user to load the weekly food menu from local Json database. However StudentJsonReader is used to get their personal information like BMI, weekly food-taking data and use these data to generate students' personal weekly reports. Due to their different designs, in order to easier to implement and reuse, a factory method pattern is used. Figure-5 shows the UML diagram of a factory method pattern for JSON readers.

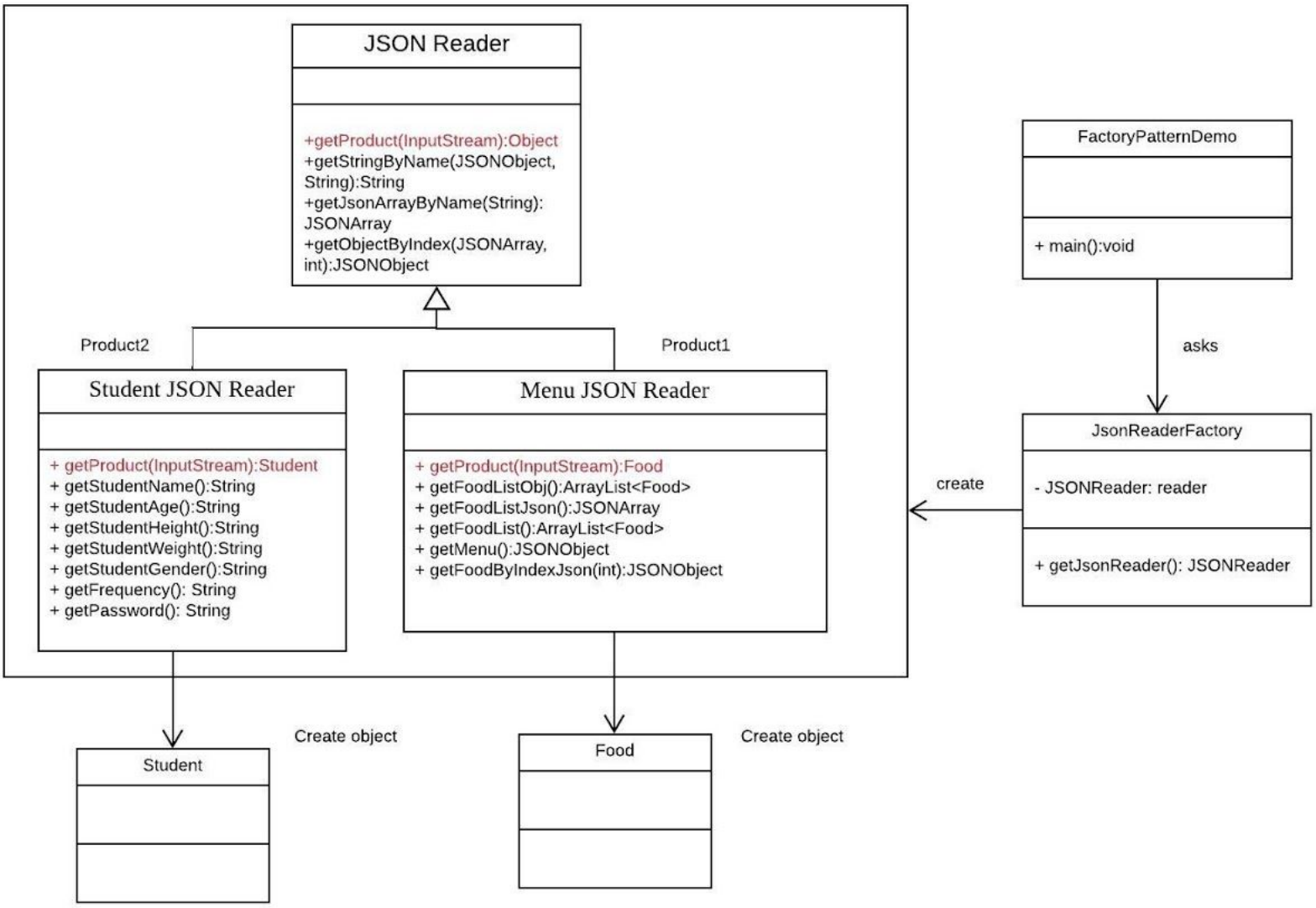


Figure-4 The UML diagram of a factory method pattern

As Figure-4 shows, the Student JSON reader and Menu JSON reader are extended from the same abstract classes, JSON reader. The base JSON Reader play the role of unifying the behaviours of their children classes. In our factory design pattern, the JSON reader is the abstract reader class. Two products are the subclasses of JsonReader. Both MenuJsonReader and StudentJsonReader inherited getProduct function from JsonReader. StudentJsonReader and MenuJsonReader are two independent product and the object created by them are obviously different from each other.

To figure out how does JsonReaderFactory works, we need to look deeper into the code part in below. The prototype code part shows that JsonReaderFactory class uses a parameter called JsonType to define the product type. After the JsonType is decided, the static function of JsonReaderFactory class will return the product with specified type.

```
class JSONReaderFactory {
   public static JSONReader JSONReaderFactoryMethod(InputStream is) {
      JSONReader product = null;

      int JSONType = determineJSONType(is);
      switch (JSONType) {
         case JSONReaderFactory.Menu:
            product = new FoodReader(is);
         case JSONReaderFactory.Student:
            product = new StudentReader(is);
         //...
      }
      return product;
   }
}
```

Assignment of tasks for each team member:

Xiaofeng Luo:

In conclusion, in the past few weeks, the work I have done is to draw overview diagrams, write documents in every delivery and figure out how to create an adapter that connects activities and XML files. For D1, I have work on the proposal with all of my teammates. For D2, I have been work on adapter classes and component diagram and for this D3, I have taken the responsibility to all the diagrams and I have work with Zhaohao Li with the document part. For the following D4, I will keep working on the doc and try to work with Yejing Li and Ziyang Zhou with the rest of the coding.

Yejing Li:

To conclude, my work is mainly about writing fundamental Classes, helper Classes, building UI framework, and populated the menu database into JSON files. So far, I have finished build up our application layers according to the component diagram layouts, which is designed by Xiaofeng Luo. In the user login and register subsystem, I created Login Activity and Create User activity Classes, which accepts user input and store them into the local repository, using JSON format. After the user has logged in, the Home page would list the personal information, and three modes available to choose. This is accomplished by building a Home Activity and add four fragments(Personal Information, and three modes) into the activity. For this part, my work is more about rendering user information and dealing with the data interactions between the user and the application. The algorithmic and mathematical part of code for Health Mode, Self-selected and Weekly report is done by Ziyang Zhou.

Moreover, I have connected the application architecture with XML layouts, rendering our application, ensuring the user experience while using our app. Last but not least, our menu database was populated by using the REST API from The Caf website. Also, I have build student readers and writer classes to generate student databases in the format of JSON files.

Zhaohao Li:

My contribution to our application can be described as the documentation task. At the beginning of the project, I worked on the project proposal with all team members, writing some ideas and make sure it is achievable and of the essence. After that, before the deliverable 2, I worked on building the database, making sure our food information match The Caf website. Then I was responsible for visualizing available dishes in the Caf by the implementation of Menu JSON Reader. Also, working on the image of each food and upload functionality. During the deliverable 3, I have been working with Xiaofeng Luo to make the documentation. I make the text parts, talking with Yejing Li and Ziyang Zhou to comprehend the stage we are in, in order to correspond to the code part and diagram parts as well. For deliverable 4, I will make the documentation again and make the demo video and post-production.

Ziyang Zhou:

Generally, my work can be described as two main parts: designing the main project structure and the algorithmic and mathematical part of code for Health Mode and Weekly report. At the very beginning of our product, I build the structure chart of the whole project and made several announcements of the individual functionality of each mode. For the code part, my part is building the health mode and weekly report. Health mode is one of three major modes of our project and weekly project is also one of them. Health mode will serve the users with an automatically generated menu with the calorie limit set by the students and weekly report will produce a report about their health situation with the data of their weekly calorie-taking.