Design Sketch

Ryan Wren

Vincent Cheng

Efmajackson

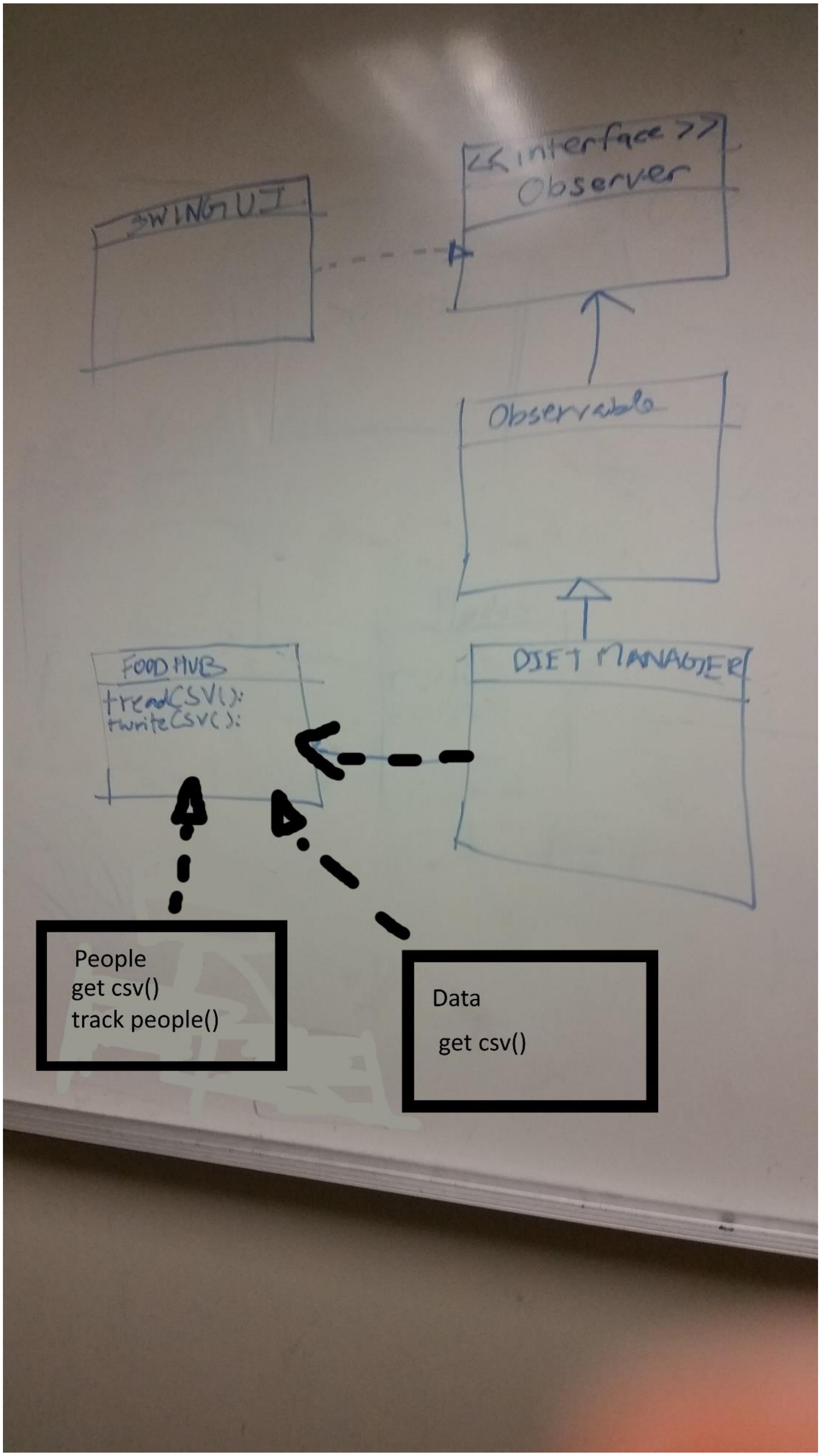
Vignesh Kumar

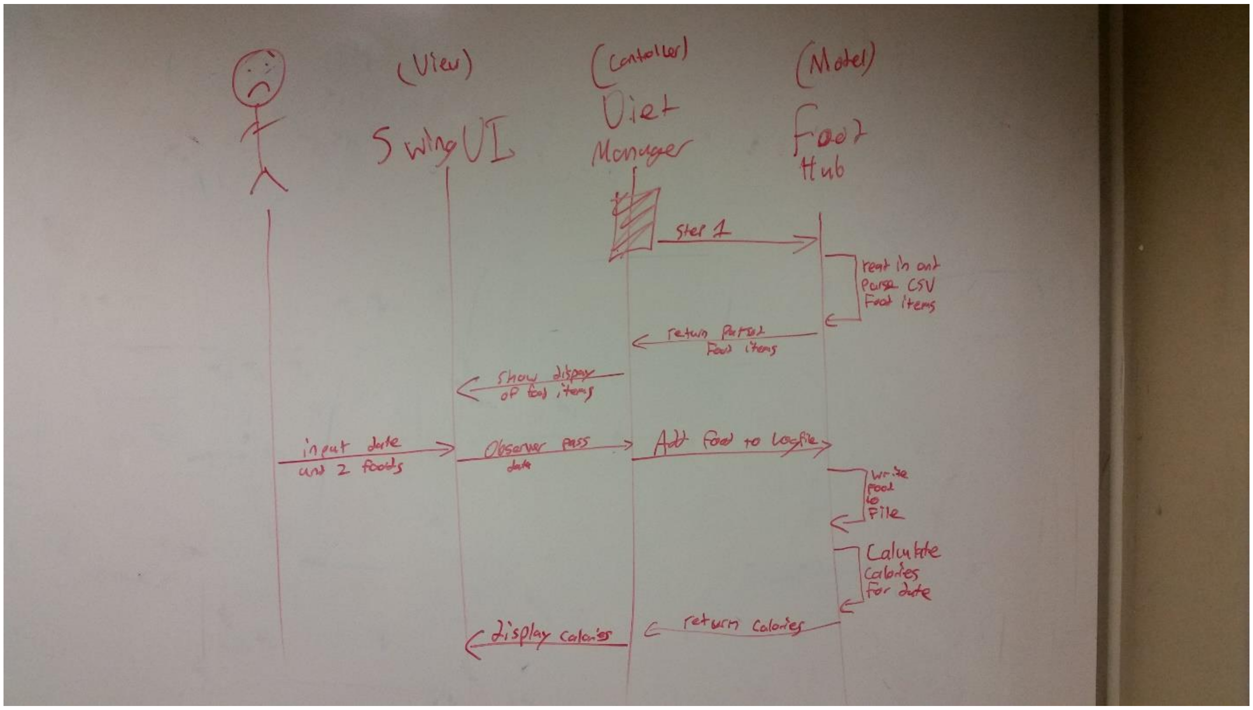
Team E

4. 1. The three main application classes other than the UI would be the FoodHub and the DietManager class. The FoodHub acts as the Model and the DietManager acts as the controller class. We are using Swing UI as the view. The FoodHub class will have the methods that can read from the CSV file and can also write to the CSV file. The DietManager class will have object that invokes the method in the FoodHub class and passes its value to the Swing UI (View) to display it for the user. The people class acts as a composite with data class acting as the leaf and FoodHub as the component. The objects of DietManager class will know what's happening with the UI and the FoodHub class as it acts as a middleman between these two classes.

4. 2. The reason for organizing the system in this way is to implement MVC to make the code work efficiently. This way of organizing our project is very helpful in increasing the overall cohesion of each class and also weaken the coupling. This also ensures we implement clean code and don't violate the DRY principle. As we move forward with this project, it becomes easier to maintain the code because of how the whole application is structured. Error identification and fixing is easier with this structure.

The only disadvantage that we could think of is that we might find a design pattern which might make it easier and more efficient for managing complex items and keeping track of the diet as we proceed further with this project.


<<interface>>
Observer
SWINGUI
Observable
FOOD HUB
+readCSV();
+writeCSV();
DIET MANAGER
People
get csv()
track people()
Data
get csv()


(View)
Swing UI
(Controller)
Diet Manager
(Model)
Food Hub
Step 1
read in and Parse CSV Food items
return Parsed Food items
show dispay of food items
input date and 2 foods
Observer pass data
Add food to logfile
write food to File
Calulate Calories for date
return Calories
display calories