Deliverable #2:

1. **Name:** Quinlin McNatt
2. **Project Tracker:** Fitness Tracker
3. **Project Summary:** The goal of my semester project is to create a fitness tracker app that I can use to track statistics on workouts, health, and nutrition.

---

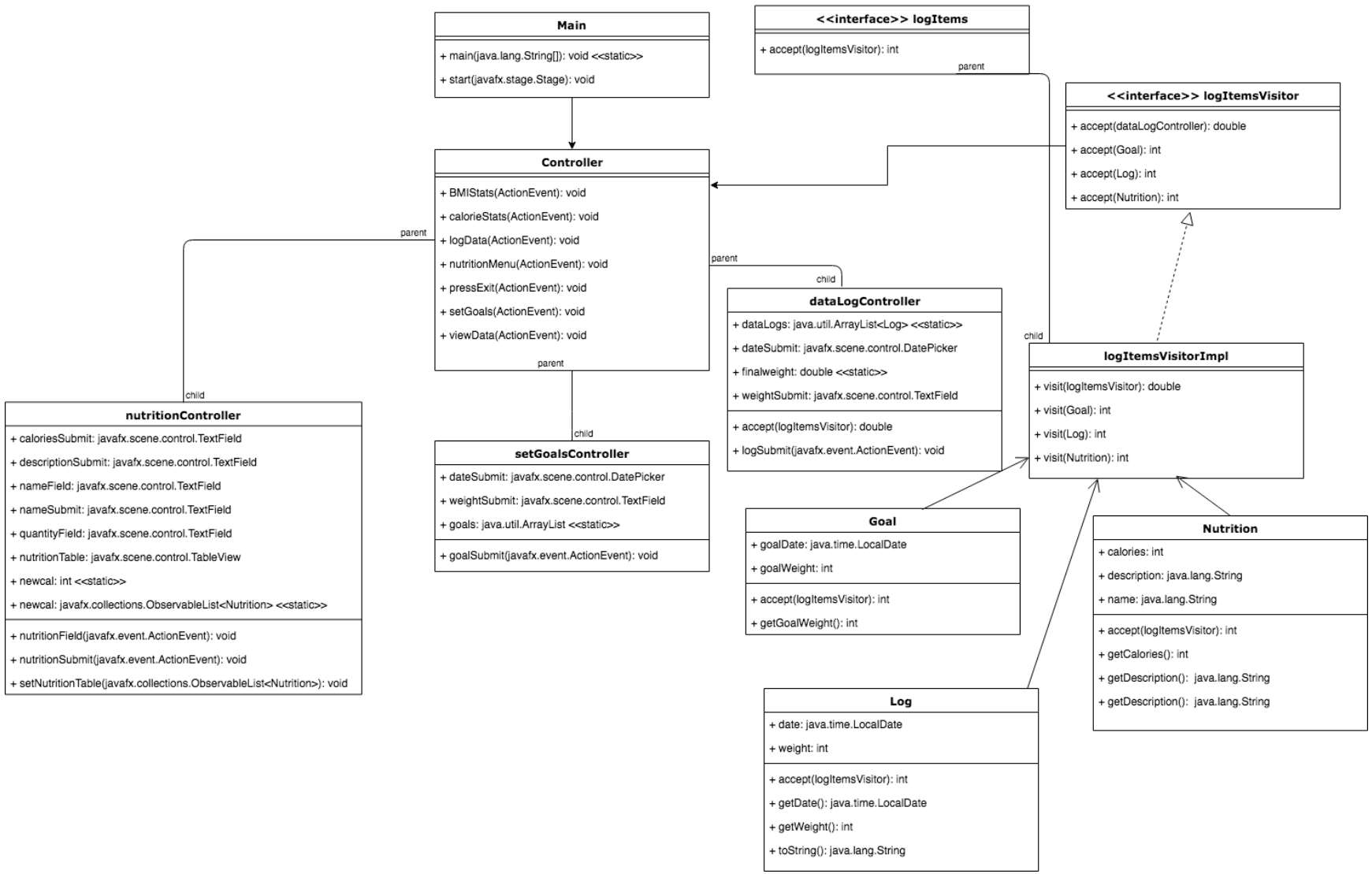
4. **Project Features:**

Functionality ID #	**Requirement ID #**	**User Requirement:**	**Implemented: (TRUE, FALSE)**
1	1A	User can create account	FALSE
2	2A	User can login to account	FALSE
3	3A	User can input information	TRUE
4	4A	User can view logging history	TRUE
5	5A	User can set goals	TRUE
5	5B	User can change goals	TRUE
6	6A	User can add nutrition	TRUE
6	6B	User can change nutrition	FALSE
7	7A	User sees visual analytics on log	TRUE
8	8A	User can search created workouts	FALSE
8	8A	User can search created nutrition	TRUE
9	9A	User can log nutrition	TRUE
9	9B	User can log weight	TRUE
10	10A	User can browse workouts	FALSE
10	10B	User can browse nutririon	TRUE

Basically, the features that were not implemented were all of the workout features, as this would be a carbon copy of the nutrition, I just ran out of time, and a login feature as the project is just for myself, and implementing a login would not be difficult.

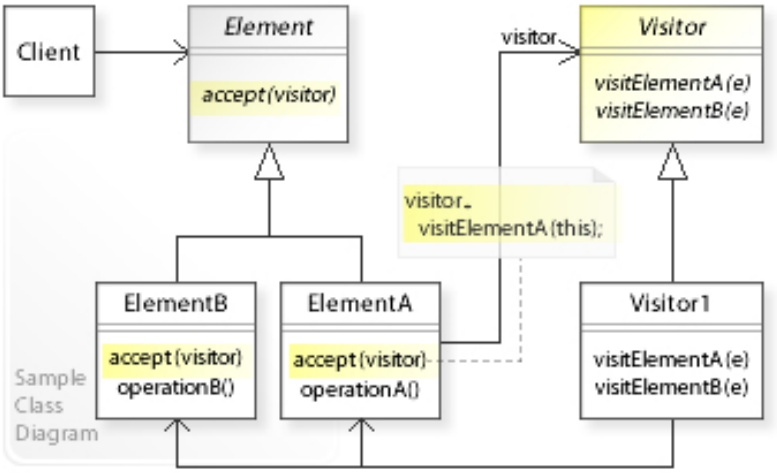
Note: For part 5, I rotated the diagram since it's too large to fit on the pdf in the right way.

Class Diagram:


Main
+ main(java.lang.String[]): void <<static>>
+ start(javafx.stage.Stage): void
<<interface>> logItems
+ accept(logItemsVisitor): int
parent
<<interface>> logItemsVisitor
+ accept(dataLogController): double
+ accept(Goal): int
+ accept(Log): int
+ accept(Nutrition): int
Controller
+ BMIStats(ActionEvent): void
+ calorieStats(ActionEvent): void
+ logData(ActionEvent): void
+ nutritionMenu(ActionEvent): void
+ pressExit(ActionEvent): void
+ setGoals(ActionEvent): void
+ viewData(ActionEvent): void
parent
parent
child
dataLogController
+ dataLogs: java.util.ArrayList<Log> <<static>>
+ dateSubmit: javafx.scene.control.DatePicker
+ finalweight: double <<static>>
+ weightSubmit: javafx.scene.control.TextField
+ accept(logItemsVisitor): double
+ logSubmit(javafx.event.ActionEvent): void
child
logItemsVisitorImpl
+ visit(logItemsVisitor): double
+ visit(Goal): int
+ visit(Log): int
+ visit(Nutrition): int
parent
child
nutritionController
+ caloriesSubmit: javafx.scene.control.TextField
+ descriptionSubmit: javafx.scene.control.TextField
+ nameField: javafx.scene.control.TextField
+ nameSubmit: javafx.scene.control.TextField
+ quantityField: javafx.scene.control.TextField
+ nutritionTable: javafx.scene.control.TableView
+ newcal: int <<static>>
+ newcal: javafx.collections.ObservableList<Nutrition> <<static>>
+ nutritionField(javafx.event.ActionEvent): void
+ nutritionSubmit(javafx.event.ActionEvent): void
+ setNutritionTable(javafx.collections.ObservableList<Nutrition>): void
child
setGoalsController
+ dateSubmit: javafx.scene.control.DatePicker
+ weightSubmit: javafx.scene.control.TextField
+ goals: java.util.ArrayList <<static>>
+ goalSubmit(javafx.event.ActionEvent): void
Goal
+ goalDate: java.time.LocalDate
+ goalWeight: int
+ accept(logItemsVisitor): int
+ getGoalWeight(): int
Nutrition
+ calories: int
+ description: java.lang.String
+ name: java.lang.String
+ accept(logItemsVisitor): int
+ getCalories(): int
+ getDescription(): java.lang.String
+ getDescription(): java.lang.String
Log
+ date: java.time.LocalDate
+ weight: int
+ accept(logItemsVisitor): int
+ getDate(): java.time.LocalDate
+ getWeight(): int
+ toString(): java.lang.String

Part 6:

For my design pattern, I implemented visitor. As you can see I have a visitor interface, then an interface class that lists all of the functions of each class I "visit". The reason I wanted to use visitor, was it provided me to be able to write methods for classes without writing in the classes themselves, therefore if I wanted to add analytical features with my controllers in the future, I wouldn't have to change the base classes, only add methods to my visitor class to produce more analytics. Here is the class diagram for visitor:



The classes that implement my design pattern are:

- logItemsVisitor
- logItemsImpl
- logItems
- Log
- Nutrition

  The Log and Nutrition classes have accept() methods which allow the visitor to access instances of the class. For every analytical method I wrote out a visitor method to do the calculations.

Another design pattern I half-implemented was observer, as for my nutrition table, I needed a way to update the table every time a new instance of nutrition was added. I used an observable list and imported the java.observer class inorder to update the list.

Part 7:

Overall, the main thing I learned and wish I had done better is planning. I had very little comprehension of object oriented design. At the beginning of the project, I had no idea I would even need controllers in order to code a UI. I have learned a ton about design patterns, and java, a language I had never touched. I think the best way to tackle this project would be to pick a design pattern, and shape your project around it, as I was not thinking about this before, I had to re-format my project in order to implement one. Overall the project was super insightful and I learned a lot about Object Oriented Analysis and Design.