Controller: We used Controller to carryout user interactions with the home screen of the app. For example, if the user choses to open the add calories screen, then the dashboard controller will handle the user interaction (mouse click) and give a call to the add calorie controller which will handle all user input and add it to the database.

Creator: Throughout the project we had to use many temporary arrays to retrieve data from the database. We made these arrays using the creator grasp and the Abstract Factory design pattern. This really refined our code and made the creation of new Arrays easier to implement.

High Cohesion: Each screen that shows up in our app has a unique controller class which only deals with the methods and on-screen objects for that screen. This implementation uses high cohesion which makes each function specific to its rule and makes it not have more responsibility than it needs.

Indirection: To separate the database statements and the Java FX code we created a class to indirectly connect the two. This allows for cleaner code and reusability. The DatabaseGateway class inserts, selects, removes objects from the database that is inputted by the user. This means that the controller classes and the database code could be reused if needed by another project.

Information Expert: To ensure that each system action is delegated to a class we used Information Expert. This allowed us to create a class for each controller, which delegated each screens responsibility to a controller, so no class is doing two classes worth of work.

Low Coupling: We integrated Low coupling by ensuring that each User has no awareness to a Goal or Food. All the User has is a calorie count which is linked to food and goals using the database as a middleman.

Polymorphism: Polymorphism is used in the Female and Male users. The equations for calculating Body Mass Index is different for males and females. So, the Male User and FemaleUser is inherited from the User super class.

Pure Fabrication: To ensure Low Coupling and High Cohesion we had to create a controller and a Database gateway. These two made it so each class is specific to its role and so classes will not know about each other if they don’t need to.

Do Not Talk To Strangers: Users have no interaction with other users, and Goals and Calories have no interaction with each other. This ensures that no class is interacting with an unrelated class. The only classes that have a relation with each other are the ones that need to.