Design Patterns:

Singleton: To interact with the Database in our app, an instance of the DatabaseGateway class must be created. Since DatabaseGateway has the potential to modify the data in the database, we are only allowing one instance of it at any time, using singleton. This lowers the risk of unnecessary interaction between the Database and the rest of the application. If a method requires an instance of the DatabaseGateway the class will return the same instance over and over so that there is never more than one instance.

Abstract Factory: When interacting with the database the creation of temporary ArrayLists was necessary. When selecting Food, Calories, or Goals, an ArrayList would have to be created to hold the large amounts of data. In order to support reusability and code arbitration we used a ListFactory to produce a List whenever a controller needs one.

Command: To promote high cohesion we made a controller class to handle each screen that is viewable by the user. Since each of these controllers have to execute a command to display, the command pattern was used. This allows us to use the controller.execute method on each controller and assume that it will successfully open the page.

Observer: When updating a user’s information many different fields must be filled. If a field becomes changed, then the database must be notified in order for the data to be saved. This is implemented as an observer pattern. The observer pattern allows us to notify all observers anytime a field is changed, which in return will update the database.

State: We wanted to reward the user with a ribbon if their FitnessScore is good. A user can get a Gold ribbon for a great score, a Silver ribbon for a okay score, and a Bronze ribbon for a bad score. In order to recognize the level at which the user is currently in we used the State pattern. This allows the system to know how which ribbon the user deserves at all times.

Prototype: The user has the ability to add, remove, and undo goals. When undoing a goal, a copy of the goal must be created so that the goal can be added back into the database. This happens through the prototype pattern. The goal gets cloned using the cloneable interface and gets sent to the database using an instance of the DatabaseGateway.

Memento: When the user deletes a goal, the goal must be saved somewhere so that an undo will add the goal back to the goal list. These deleted goals are saved using the Memento pattern. A deleted goal will be saved into an arrayList created by the Factory, and then held until the user requests the goal to be undoed.