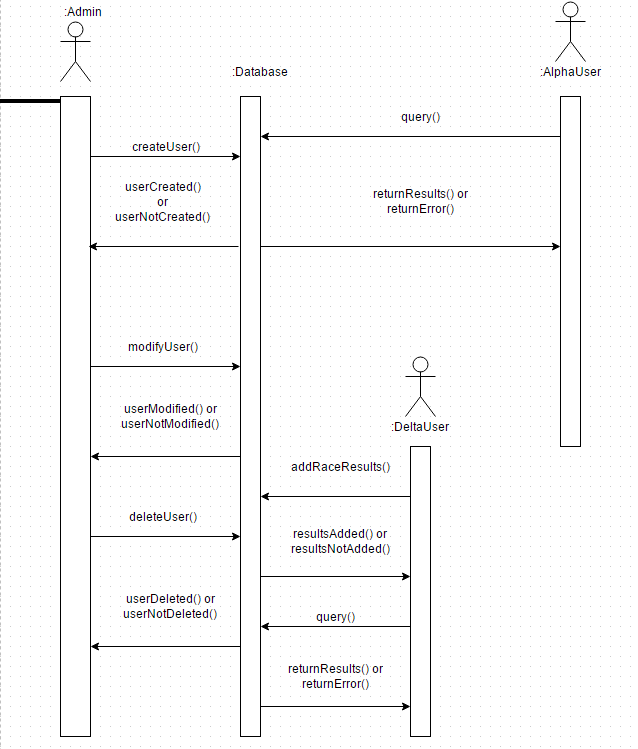
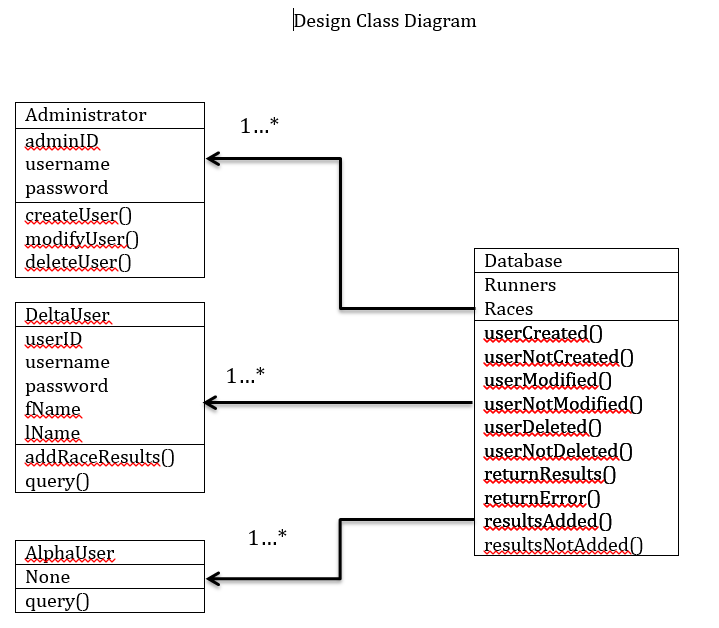
Design Document

Interaction Diagram





For the Interaction Sequence Diagram we have four main actors each corresponding to their own class or object. The Admin, and Delta User are all Creator classes. The admin can create users, Alpha or Delta, as well as modify user information. This means that the Admin must have the ability to create instances of both Delta and Alpha User classes. Delta Users are also Creators because they have the ability to create instances of the RaceResults class and add that information to the database. The Alpha User has very limited access and can only really query the database so it’s designation in terms of GRASP is not clear. It cannot create instances of classes and it must go through the Database object in order to obtain information.

The Database object is our Information Expert in this case. A Database object can provide any information to the other classes that they may need such as race results and user information.

Our system overall has relatively low coupling and high cohesion as it is not an especially complex system. Our Admin has only two responsibilities and that is to manage both Alpha and Delta Users. That means it manages only two classes. Our Database doesn’t not manage any classes, it is only there to provide access to the information available to it. Our Alpha User has no responsibilities either, it is only allowed to query the database so it can ignore any other classes. The Delta User has the ability to create race results giving it a single class to manage.

Our Controller is the overall system. The UI that is currently implemented acts as a controller allowing any users to interact with the system. When a user enters a query or attempts to login, the system sends the info to the class it need to go to and then that class sends the information requested back through the system.

As for polymorphism, our current system has none implemented. Each and every class is independent of one another and none of them are extensions of any other classes.

Our class also has no instance of Pure Fabrication represented. The Delta user and Admin send data to the Database object through the system which the Database object takes care of saving itself. This may heighten coupling and reduce cohesion but since the system is rather simple already this doesn’t create much of a problem. At least not at this point in development.

Lastly, our system has no need for Indirection or Protected Variation.

The Design Class Diagram doesn’t add much to the discussion that hasn’t already been stated in the preceding paragraphs. It shows the relationship between the Database and the three other actors. The Database keeps track of all information related to the two users as well as race results.

The one database has a one-to-many relationship with each object. The database can have many admins. The database can have many alpha users. The database can have many delta users. The relationships here are very simple.

Design Patterns

As for GOF design patterns in our pattern, a Runner could be seen as an observer. A Runner signs into the system and is able to observe changes to race results as well as view and modify their own racer information.

Our organizer, or admin, is essentially a factory. He creates races and race results as well as reviews racer information before approving it and allowing it to enter the database.

The database could be seen as a Singleton. There is a single instance of it and everything in it can be modified by several actors in the code. This information needs to be the same no matter who is accessing it when.