Challenges and Lessons Learned

Challenges:

1. **Understanding Design Patterns**: I've worked with various Design Patterns before so making a plan for Development wasn't too hard, but understanding more of them better before getting too deep in Development would have helped and made the initial version of the System take much less time. You can't predict everything that will make a project better sometimes, but once I researched and understood more how Controller Design Patterns worked it made the Development much easier to do but took time in converting everything to utilize it.
2. **Design Document Creation**: The Domain models and Use Cases documents weren't hard to make, but the Design Document was a nightmare due to its massive scale. This took a very long time to build and then continue to make changes to after Development progressed.
3. **Low Coupling**: This concept was introduced a little late in the class for when development was already happening, but once it was it made since why this is important to keep certain logic restricted to certain areas to make refactoring much easier later on.
4. **Perfectionism**: I like to have things in full working order before submitting, but this does not fit with Agile development cycles and I need to be better with having a working version that does most things being good enough sometimes. I had to let go of my need to get everything working exactly as it should and accept what does work for what it is and simply state that some things are not completed yet and that was okay. My grade wasn't affected for some of the logic that wasn't implemented in my first version, so I need to let go a little more when it comes to submitting an initial version of a project.
5. **Creating a Working Application**: This was my first project on this scale that I've worked on that was a complete application on its own and that was a big challenge to understand how to do. I've done many assignments before that have done this on a smaller scale, but nothing like this, and it was a huge learning experience as well as a great Challenge to figure out how to get certain functionality working in this setting, such as having a login screen for different users function correctly. There was a lot of requirements for this project that were things I hadn't worked on too much before, most previous assignments have been applications that just do something stand-alone like a Calculator or something to display information to a user that they can interact with. This project, however, required a lot more thinking, planning and new development challenges that I hadn't experienced before that I'm grateful I have under my belt now. I learned a ton about Refactoring also and information I didn't know was available in Eclipse, such as the Call Hierarchy, that made fixing bugs a lot easier.

Lessons Learned:

1. **Start Writing Testing Classes Early**: There were many bugs I discovered through building Junit tests that were very useful in getting the program to work properly in ways I wouldn't have thought, and saving this for later in my Development cycle was a mistake. I should have followed the practice of "Write Code, Test, Write Code, Test" to have avoided this.
2. **Address Non-Development and Non-Design Requirements**: This relates more to the creation of the video, integrating our code in Github, anything that wasn't directly related to the creation of the Gym System and more a procedural requirement of the Project. For whatever reasons I encountered problems in these two specific examples that wasted time and I wish I had done some tests of them earlier. For instance, my first commit of the entire Gym System was when I had it almost done for P3 and Github just did not want to sync with Eclipse and I have no idea why and unfortunately now don't even know what I did to fix this. It was a headache that caused stress and made the project harder for no reason other than me not trying it out sooner and getting everything set up earlier for pushing code. Additionally, the computer I decided to record my demo on I had never done before and I should have prepared more in advance.
3. **Design Document Creation**: As noted above, this took a lot of time to build and I still don't fully know an easy way to do so. This Document took more time than any other one that I made and I'm going to guess that with practice this will be faster to build in the future, but for this assignment I spent many, many hours getting it done and not sure how to make the process faster right now.
4. **Consider User Experience**: I am fine with my Command Line Interface, but it was the stroke of genius while I was developing where I figured out an easy way using maps to display options to the user for making selections versus having to Search for an Entity or Person by name. The original Use Case documents stressed the Client searching for people by name, but this is not ideal UX and also made bugs much more likely to happen unless some really thorough logic was written to prevent this. Instead, keeping the interaction between various Domain types through the numeric input was a Godsend and once I found an easy way to do this it was by far the best way to allow the User to make selections and changes in the View. I had to spend a lot of time going back and putting this in all the methods that I originally had written the other way, with the original idea being "this isn't ideal, but I'll fix it later" being my downfall. This was my fix and if I would have done it from the start by thinking about how I was proposing the user interact with the System it would have made development a lot easier.
5. **Refactoring is your Friend**: Due to the constant changes that came in this development, refactoring was a requirement and saved much time when it came to fixing bugs. I had little experience before working with so many files and classes in an assignment in Eclipse and I learned a lot about how to simply set up a project better next time, in addition to using the refactoring functionality inside it to make changes safely and easily in the future too.