Creating a UML Class Diagram before writing my code really helped me understand how all the pieces fit together and how my program would be structured. In the past, I had always drawn my diagrams to fit my implementation, but I decided I wouldn't do that this time. It ended up being a major benefit because of the added understanding and involvement I had in the design. Naturally, a few things changed as I wrote my code, but it was structured enough that the actual programming took significantly less time than I thought it would.

While designing this program (see figure on following page), I decided I wanted to have a couple levels of abstraction. I first made a Shape class that all the other classes inherit from (including Point and Line). The reasoning behind this was to make one common move function. On the next level, I had my Point and Line classes along with a TwoDShape class. The TwoDShape class was another abstraction of the remaining classes. This allowed the getArea function to be inherited by the children. After that, I simply had Square as a child of Rectangle, and Circle as a child of Ellipse. At that point, it was really simple to implement all the different classes and their desired functionality.

I never thought I would admit this, but Testing saved me. I wrote my test cases while following along with the functional requirements in the assignment description. A set of my tests failed for my Triangle class and I realized that I had not properly implemented a piece of it. I had to go back and add some more validation before I could get my tests to pass. It was nice to be able to have tests that matched the assignment requirements so that I could have confidence in my implementation.

Overall, I learned how important it is to design and plan out a program's architecture before actually coding it. Designing helps the implementation run smoother. I also learned that testing definitely has a place in software engineering and can prevent oversights and breaking code later on.