Andy Fleischer

Prof. Karla Fant

CS202

19 August 2021

OOP Term Paper

Going into CS202, I had a bit of exposure to object oriented programming from high school computer science classes and personal research. It was not much exposure, and there was a lot that I did not know or understand. Object oriented programming was a looming beast that I was always too afraid to try really learning. Luckily, CS202 forced me to dive into OOP, but at a nice pace at first to really get comfortable with the concepts. Each program built directly off of the concepts learned in the previous programs, which was great for stepping through inheritance, then dynamic binding, then operator overloading. I think this progression was well-paced and structured in a way where each program got easier to write with the newfound techniques (operator overloading was my favorite, it was amazing to see my testing program just melt away). Now, I will be coming out of the class with much more knowledge and confidence in my ability to create functional class-based, object-oriented programs.

The first concepts we learned about in program 1 were the basic rules of object-oriented programming and how inheritance works. I came into the class knowing only the basic overview of inheritance, how a child class that derives from a base class

inherits all of their members and functions. However, learning the soft rules of OOP helped me better understand the purpose and structure of object-oriented programs. I remember going into the first program and realizing *ohhh I can have a parent Event class, and these three classes derive from it since they are all "Events" plus more details.* Speaking of which, that was one of the best tricks to help me understand inheritance, how a derived class IS a base class *plus more*. This structure for the first project took a bit of time for me to grasp, and a lot of testing out code to see how certain edge cases worked, but then on the next programs it felt like the most natural thing to do.

Another thing I didn't really understand was how the public, protected, and private keywords functioned in a hierarchy. However, once I really got how inheritance worked, these keywords made total sense. I understood how you want to access a class through its public member functions that can handle its own private or protected data. Also, each class can manage its own private data, its parent's protected data (and access parent protected functions), and friend private data as well. Getting this concept, and one of the biggest rules of OOP, that all data should be private and have no getters/setters was really key for my understanding. It made me understand how each class should have a purpose, and you shouldn't just throw in a class because you feel like it while all these other classes go and muck with its data and do its job for it.

With program 2, we introduced dynamic binding into the mix for the civilizations game. At first, dynamic binding was so confusing to me how you can have the exact

same function signature but at runtime the program knows exactly which function to call from a base class pointer. But, with a combination of the labs, pre-recorded videos, class time, and looking up and testing out edge cases on my own, I really understood how it worked. The compiler sees the virtual keyword, delays binding until runtime, and then at runtime it will know which derived class to go to to search for the function. This is where it also helped to understand regular function overloading and how a derived class hides its parents functions. Once I got those concepts, virtual destructors made total sense too, and I wasn't just doing it because I was told you have to. If the base class doesn't have a virtual destructor then of course how would you ever reach the derived classes' destructors?

Finally, with program 3, we learned what I think is the coolest and most powerful concept of object-oriented programming: operator overloading. It is a shame that Java does not allow for operator overloading because it really just makes your code much more readable and clean when utilizing classes with overloaded operators. Going into this concept, I was skeptical at its power, seeing as you need to implement so many extra functions all to do the same stuff but with operators instead of functions. However, overloading the insertion and extraction operators for read and display, the equals sign for copying, the addition and subtraction assignment operators for insert and remove, and much more, made the code for handling custom objects just melt away and be so much easier to understand. I remember writing my application code for program 3 and

thinking *wow, this feels as easy as the string class*, because some of the best parts about using the string class are all the overloaded operators, such as == to compare. I think it is amazing that we didn't even get to templates yet (one of the biggest reasons to use operator overloading) and operator overloading was still so powerful.

Overall, I learned so much from CS202 that I know will be useful no matter what I end up doing in the future. Personally, I am not the biggest fan of C++ and unless I need good performance, I would much rather use an "easier" language like Python or Java. However, even if I never pick up C++ again, I feel like the concepts we learned in this class will be so useful later on. Sure, Java doesn't have operator overloading, so that may not be helpful, but the broader object-oriented design skills, like separating code into classes that have specific functions, not directly accessing data from another class, or using polymorphism for more universal code will be useful for anything. Even just having these large programs with 5+ classes that seemed so daunting at first but are typical now helped a lot. Now, I like to really sit back for a bit before coding, and come up with a good plan for implementing such a large design. If I had just gone headfirst into some of these programs, there would be so much headache trying to refactor things a million times to get it to work. They would probably have some non-OOP pieces where I had a bad implementation so I had to use bad code to fix it.

The best part about learning these skills is that I can already see improvements in my coding. On the side of this class, I have been programming my own project in Python

to make a GUI-based, online card game that I like to play with my friends, called Tractor. Because I better learned how to tackle such a large project, I came into it smarter, by making classes like Card, Hand, Deck, Game, and so on. I definitely would not have come nearly as close as I have to actually finishing the game if it weren't for these great concepts that I learned from CS202.