This project focused on developing a Java-based car investment system to manage manufacturers, car models, and sales reporting. The system included four core classes: car_model for individual vehicles, manufacturer for managing collections of models, reporting for analysing sales data, and reportingIO for user interaction through a console-based menu. While the overall task was manageable, it definitely stretched my understanding of Java, especially as someone still getting familiar with object-oriented programming.

One of the more time-consuming aspects was handling user input in reportingIO.java. I had to make sure things like weight, sales price, and number sold were properly validated. It wasn't hard to understand but figuring out how to keep asking for input when users entered blanks or invalid numbers took longer than expected. Eventually, I created helper methods to handle this, which made the code cleaner and easier to manage. Without them, the input checking would have been messy and repetitive.

Working with arrays for storing manufacturers and car models also felt a bit clunky. Since the array size was fixed, I had to check whether the array was full and keep track of how many manufacturers had been added. It wasn't difficult, just more steps than expected to manage simple storage.

The Javadoc caught me off guard. Writing the comments themselves made sense, but when I ran the Javadoc tool, it kept throwing warnings about missing comments. It turned out that even if you document a method, you have to be really careful about including every @param and @return tag, or Javadoc complains. It wasn't difficult to fix, but it was frustrating trying to figure out why things weren't working when the comments looked fine at first glance. This part made it clear how much Java emphasizes structure and consistency.

Testing the system was straightforward but took time. I manually entered manufacturers, added different car models, and ran reports to make sure everything worked. Invalid inputs, like negative prices or weights outside the valid range, were handled correctly after some adjustments. I also had to make sure blank inputs didn't cause crashes, which led to improving the input validation in reportingIO.java.

One thing I didn't expect was how much attention small details required. For example, when filtering car models by price, I initially forgot to handle cases where no cars met the threshold, which caused an empty array to return without any indication to the user. Fixing it was simple, but it showed how easy it is to overlook edge cases.

It also showed me how easy it is to overlook small details when generating documentation. While the project wasn't overly complex, it definitely reinforced a requirement for attention to detail that my python experience didn't, especially when it comes to error handling and documentation.