Name: Helena Kwiat

Title: Car Manufacturer

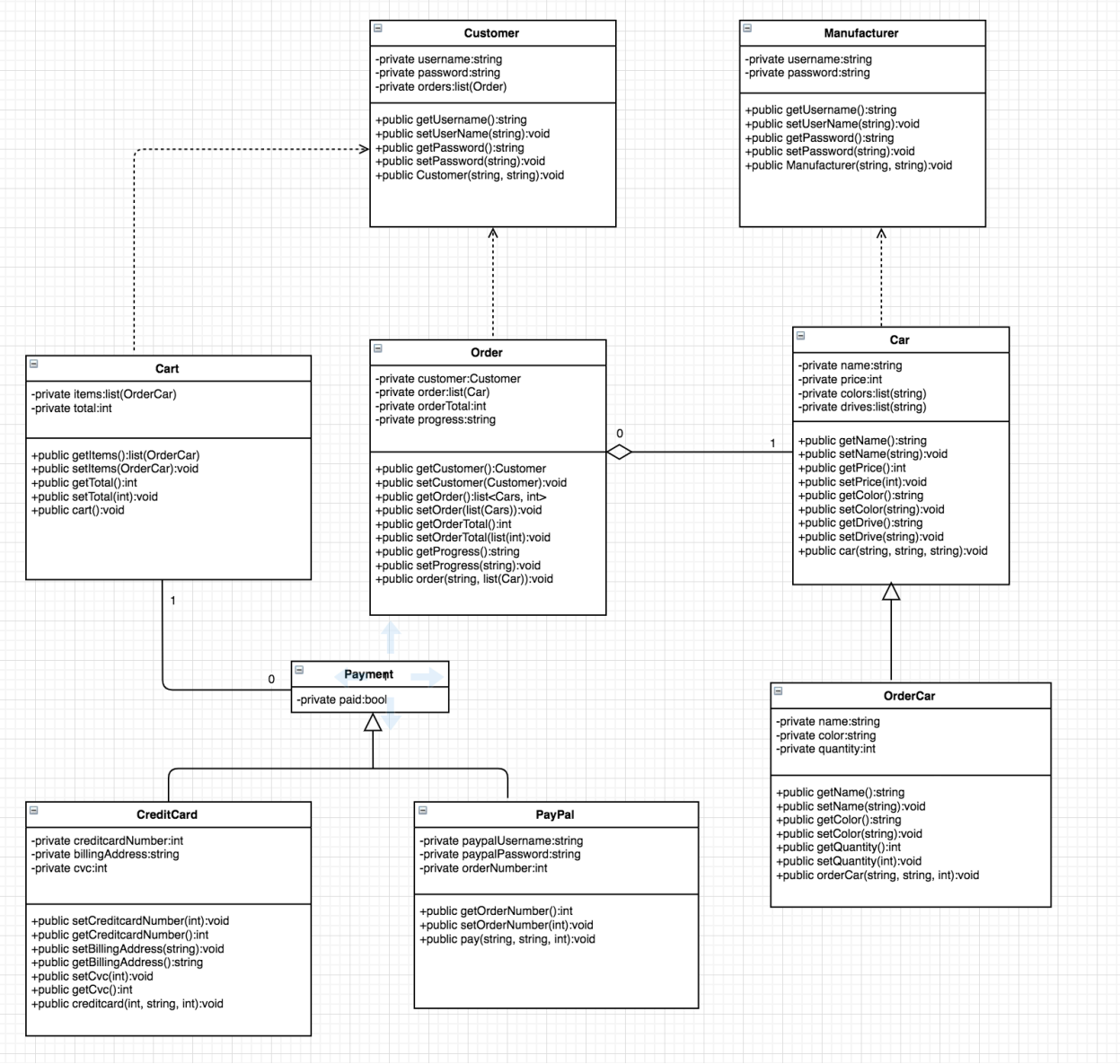
Description: Interface that allows brand name companies to order and customize cars, check progress on orders, and have them delivered.

3. Features that were implemented:

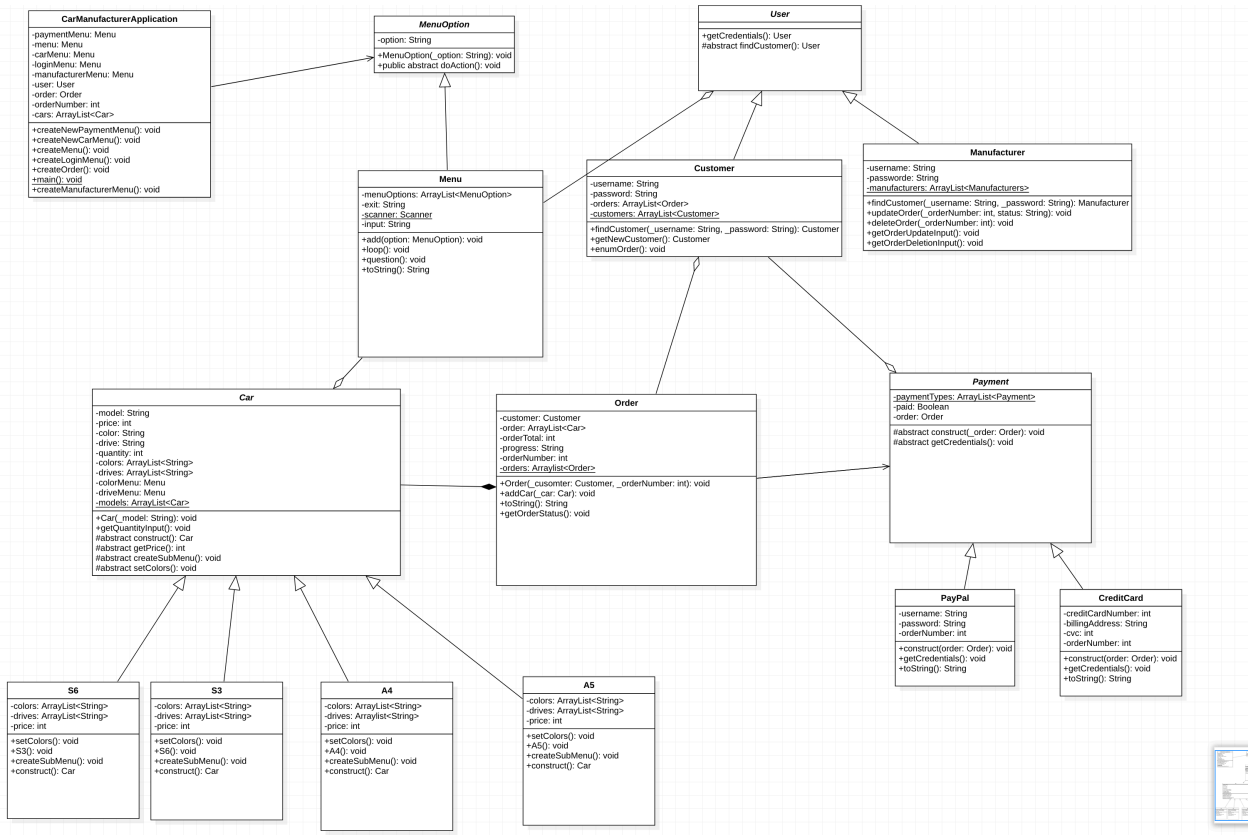
System Functionality #	ID#	User Requirement
1	1a	Be able to log in
3	3a	Be able to add cars to cart
4	4a	Be able to change car color
4	4b	Be able to change car drive
5	5a	Be able to checkout and see total
5	5b	Be able to pay
6	6a	Be able to view in progress orders and see how far along they are

4. I was able to implement all of the initial features from my User requirements.

5. Original Class Diagram:

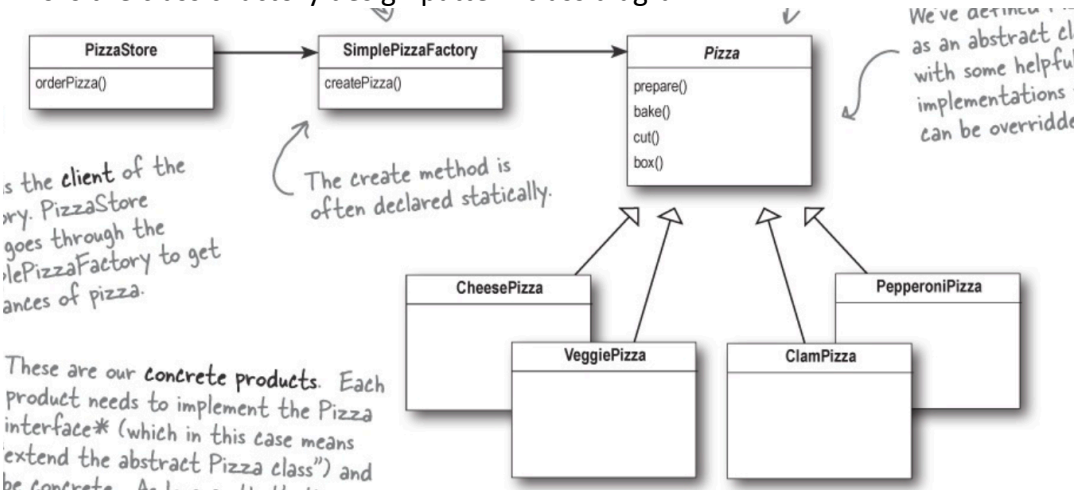


Final Class Diagram:



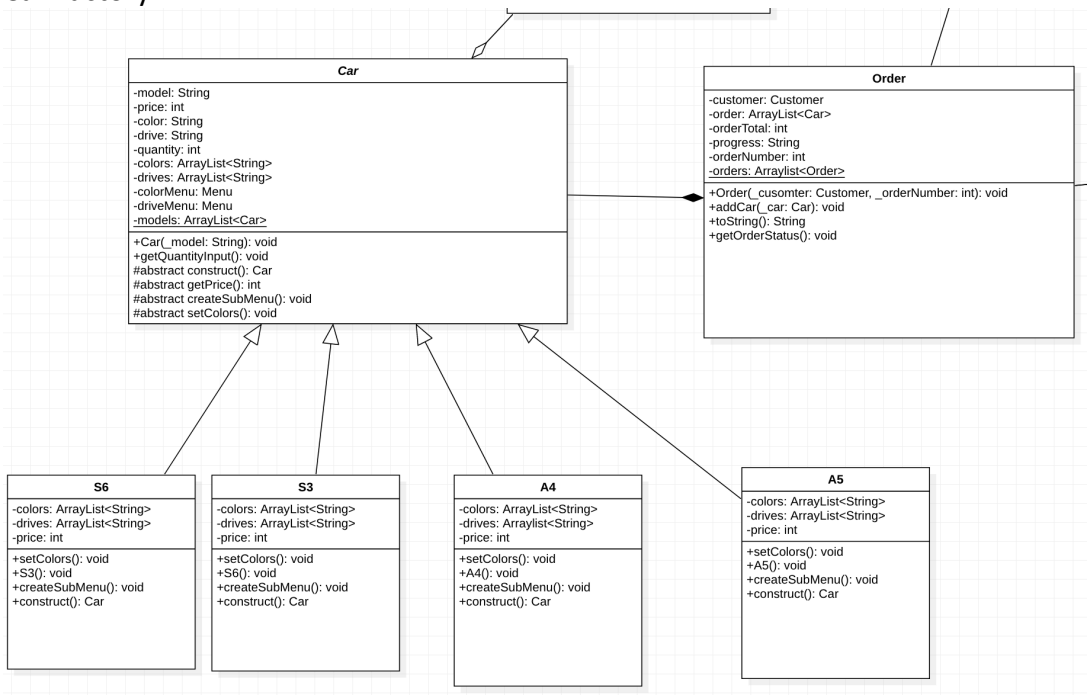
My final class diagram changed a lot from my initial one. I was unsure about what design pattern I wanted to implement when I designed it, so that had to do with many of the changes and added classes in the Car factory and the user factory. I also needed to add a few unforeseen classes to account for the menu functionalities. I also was able to take out and combine a few of the classes like OrderCar and cart as they were easily added into Order without much work. Overall my final vision is much different than what I had initially designed.

6. Design Pattern: I chose the factory pattern for my project and I have three factories. This is the classic factory design pattern class diagram:

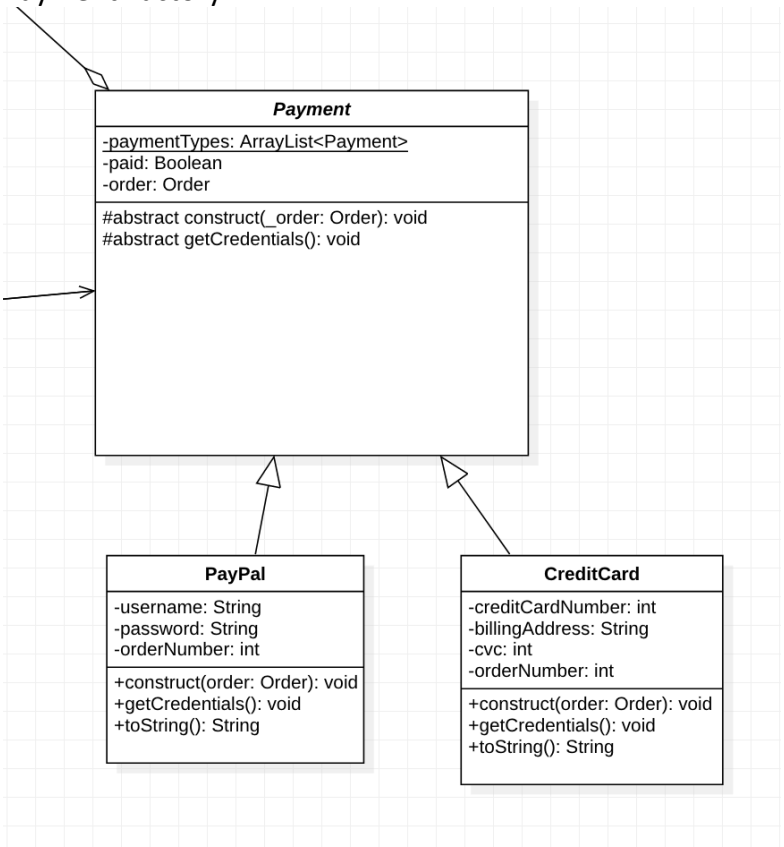


My three factories:

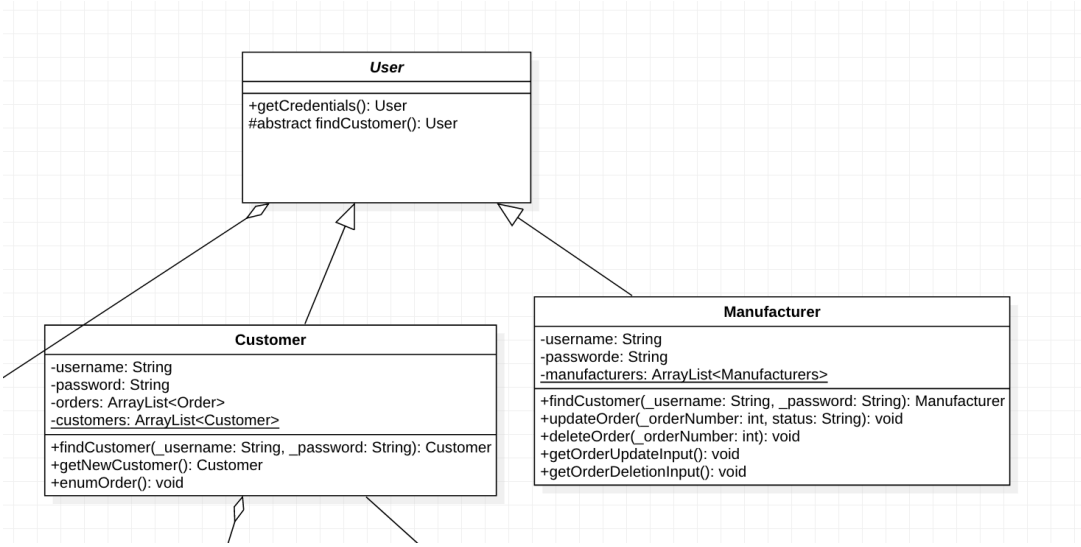
Car Factory



Payment Factory



User Factory



I implemented the design pattern with abstract overarching classes (User, Car, Payment) with a few protected methods in each. And all of their respective subclasses extended their parents. I chose the factory method because I chose to basically implement a Car factory, using the factory method allowed for me to not write duplicate code because many of the methods for the different categories overlapped, and it also prevented me from having to write many messy if-statements in order to accommodate for which car, payment, or user was chosen.

7. I have learned through this process that while it may require more thinking and planning before writing the code, using patterns and OOP principles really makes coding easier, less messy, and more readable. I was very skeptical at first, but after having implemented this system I will always remember these principals and try to write cleaner more encapsulated code.
In retrospect I probably would have done the initial design(class diagram) for this code after having thoroughly researched the design pattern I wanted to use and I think my initial class diagram would have come out a lot more similar to my final diagram and I think I would have run into less issues and refactoring down the road.