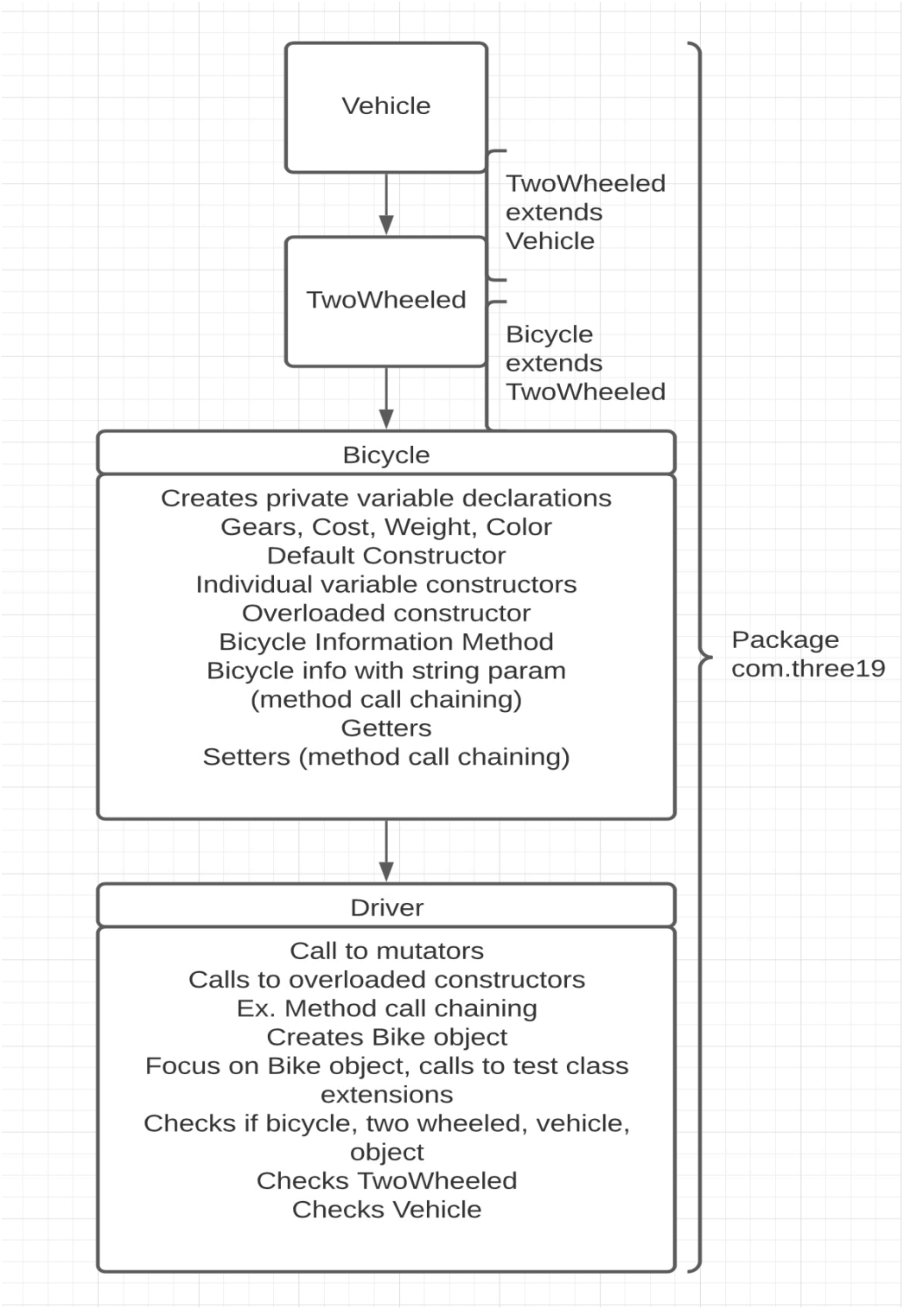
Zachary Meisner
CS 230 Operating Platforms
12/22/21
1-5 UML Diagrams

**Portability**



The portability of this program is expressed within the package itself, making it a self-contained object. This is the first instance of portability I saw when looking at the program, as everything is packaged together as a singular unit to make it easy to include, or dis-include this particular structure. Which leads to the modularity of the combined packages shown in the architecture of the program. In addition to being one of OOP's primary benefits, modularity can be shown and expressed amongst the multitude of classes within the package itself, and the ability to easily check and troubleshoot based off of the simplicity of the class in addition to the initial error checking by the program itself.

**Inheritance**

In this program we can see inheritance expressed within the extension of the class structures. Thinking of this like a family tree is the best example of what exactly is going on within the program, expressing the individuality between each class, while also showing the portability of each individual feature within each class, to the next. In this instance, the flow of data is important, as this full distinguishes the architecture of each class, and allows the modularity of the program to branch out further in the case new features are to be created, or added into the program. For example, if we had wanted to make a game with both Monsters, and Humans; we could inherit from a character class, that share basic stats for the characters, yet individualize the monsters and humans specifically when we initiate those classes themselves.

**Encapsulation**

When it comes to encapsulation, this primarily is expressing the base nature of our variables in the program. We visually see this in the bicycle class when the variables are first initialized as private, and then again when method call chaining is utilized. Method call chaining is important as it is a great benefit of OOP that shows a direct example of modularity within the program through the flow of data from a private variable to data that the user is able to see and interact with when utilizing the program. This helps the program become more secure as well, designating a specific way that data has to be called and utilized in any type of instance, making it more difficult to manipulate the program as the structure of an otherwise fluid program becomes more rigidly defined.

**Polymorphism**

Polymorphism can be expressed in a variety of ways within this program. More specifically, we can see polymorphism through the extension of classes, and within the area in which the driver file is checking the object. We can see that our bicycle object from the output of our program is also two wheeled, a vehicle, and an object. This helps us understand the structure within the modularity, and the ability we have in order to maneuver around within the different types of classes we have to work with. In addition to that, polymorphism helps us understand what exactly it is we are having a problem with, in the case one of our program error checks fails initially within the output helping us backtrack to the potential problem in our program. What I think is most interesting about polymorphism though is the actual state of being our program is taking when referencing real time. As the program instructions are followed and executed in the

computer, there has to be a differential state of being implicitly being expressed amongst the evolutionary state of the program prior to its error checking and output. I think that is insane to think about, and helps when understanding logically what would be the best way to execute a systems architecture in order to find peak efficiency in a program.