Modeling an Electric Vehicle using Java

Introduction

Many young adults take advantage of today’s modern vehicles, a tool they use a lot but understand very little about. This has led to a general sense of apathy and lack of confidence in one’s own ability to upkeep proper vehicle maintenance. This effect is exaggerated with more technologically advanced vehicles like electric cars. The Electric Vehicle Modeling System (EVMS) was created as a solution to this rising problem. By recreating an electric vehicle in Java each part and step can be separately modeled and explained one at a time. The explanation is done by creating a user interface that displays lines of text detailing what is happening in the vehicle one step at a time. This creates a simple visual method for understanding the intricacies of electric vehicles in an online form that is more appealing to young adults.

Implementation

A screenshot of a cell phone

Description generated with high confidence

When the user first runs the program, they are directed to choose the vehicle they wish to model. This is done by first selecting the manufacture than the specific model. Once the vehicle has been selected they are shown the main screen where the top half displays messages about what is happening, and the bottom half contains buttons that allow the user to control the system. They are prompted with a Start button, which after being pressed gives the user two options. They can either select the Next Step button to step through the program one line at a time, or they can select Play All and have the whole program run at once.

A screenshot of a social media post

Description generated with very high confidence

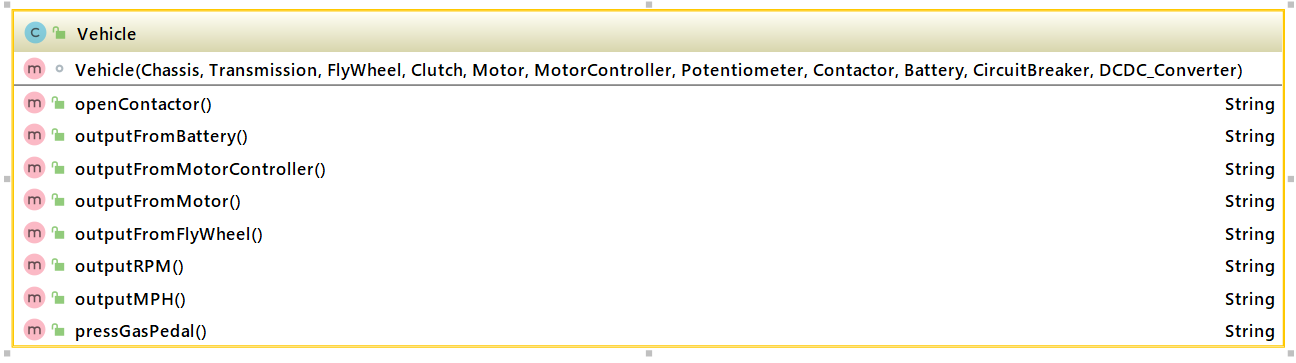
This functionality allows the user to learn at their own pace, where they can take it one process at a time or see a large summary of everything at once.

Solving the problem

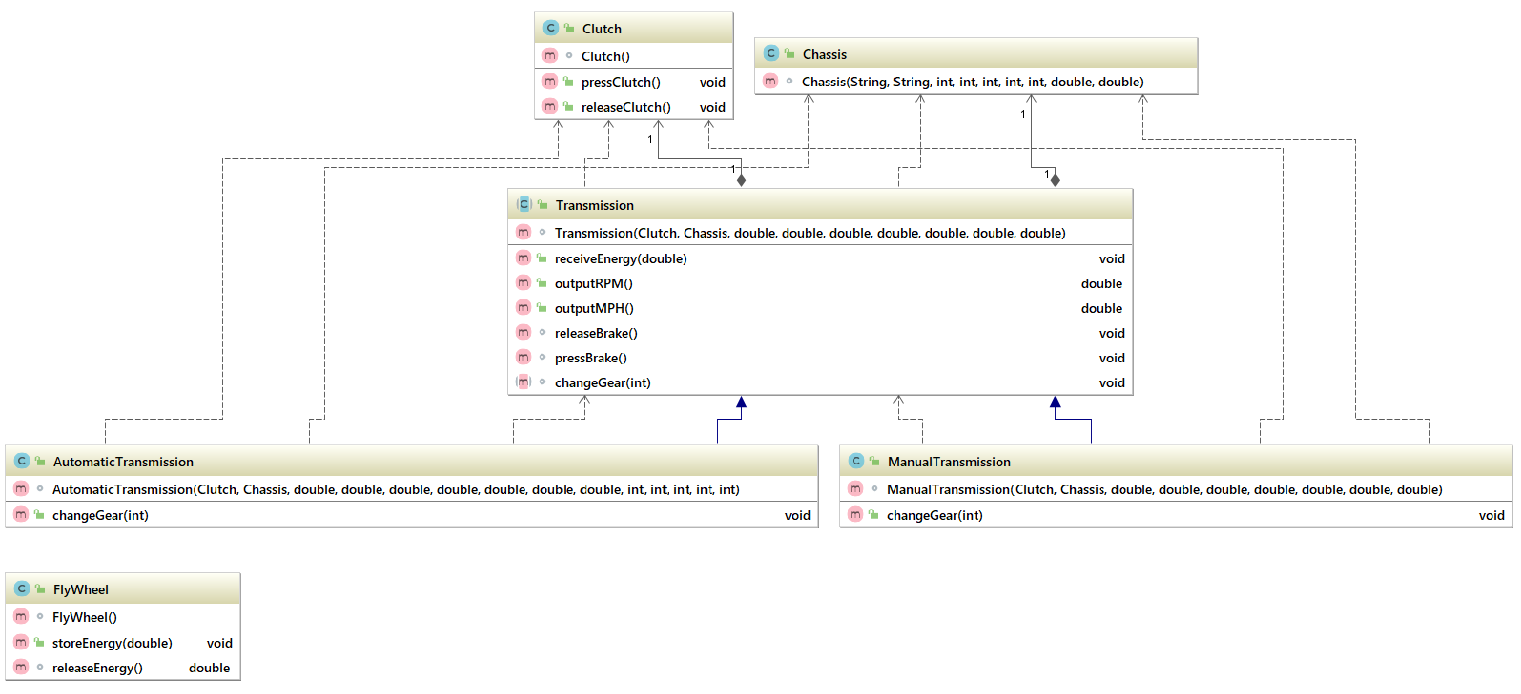
To develop a system complex enough to accurately model an electric vehicle in three weeks, a detailed time schedule would be needed. This was accomplished using a building cycle that broke down into five steps and were repeated each week for three cycles of developing different parts. First research was done on the parts being built that week, emphasis was given on what input was needed and what output was created. Then a UML diagram was generated showing how the different parts would interact with each other, and to the rest of the program. Next a meeting with the professor occurred to discuss what methods to use, what patterns to implement, and the best techniques to develop the code. Afterwards the Java code was written using the IntelliJ IDEA, while focusing on grammar semantics to make the code more readable. For example, using uppercase letters for class names, naming member variables with a lower-case m, and naming parameters with a lower-case p. Lastly the code was tested and debugged if any errors arose, this was done using system logging and a try/catch system.

How the program works

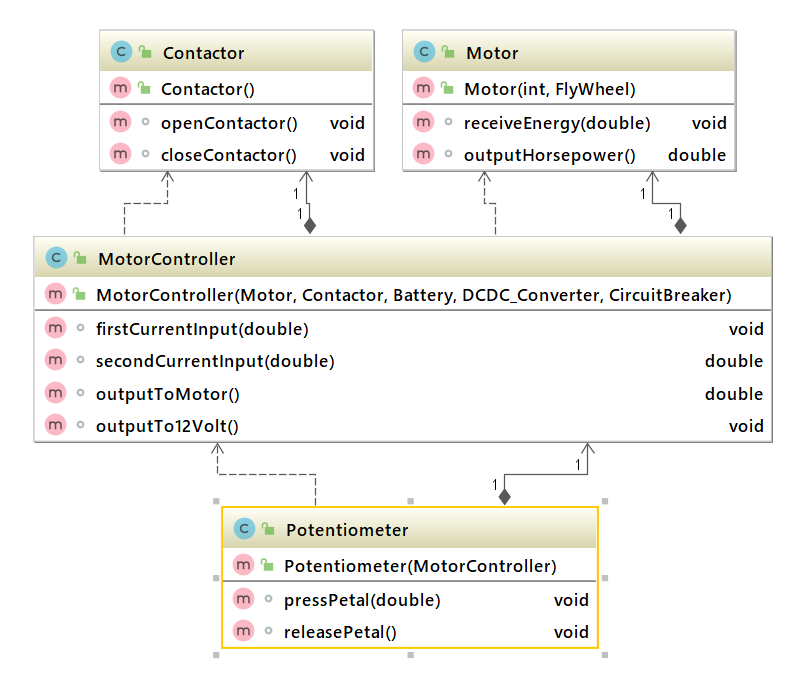
When designing this program, the goal was to create a single Vehicle class that could be used to instantiate any type of vehicle based on the parameters given to it. A large focus of this project was to implement design patterns to make the code more efficient and to remove repeat code. This is why composition was used so uniformly throughout the project, all of the classes represent a generic part and require parameters passed to it in construction to be instantiated differently. For example, if you want a Ford Explorer 2002 Motor you just create a Motor and give it the parameters that a Ford Explorer Motor would use. This was done to keep the structure of the Vehicle class generic for all possible vehicle parts. Keeping the structure generic also reduces the amount of repeat code in classes which is why abstract classes were used over interfaces. This can be seen in the Transmission class which was made abstract so that the subclasses can implement the changeGear() method differently, without the need for overwriting code.



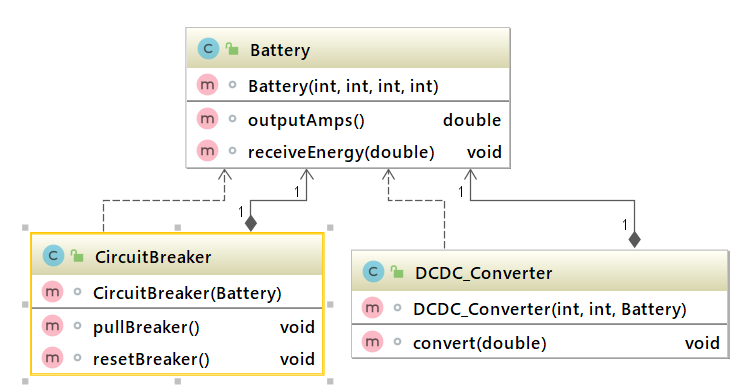
This also allows for the template design pattern to be used, where the Vehicle class has many methods that use the parts to calculate important values. That way the values these methods produce are different based on the parts used to build the car (i.e its instantiation), but assures that all vehicles are capable of these specific behaviors. Another design pattern that could have been used was to have the instantiation taken care of through inheritance. So, if a part for a specific model of car was needed, a new subclass of that part would have to be created and made differently. For example, if you wanted a Ford Explorer 2002 Motor you would have to make a new class called Ford Explorer 2002 that extends the Motor class. This pattern was not used because it would lead to class explosion in the long term when more and more cars would be added to the program.

How the program was implemented 

The first module is the vehicles base, essentially the parts that motor and electric vehicle’s share. The classes that make up this module are the Chassis which holds information about the vehicle’s structure, and the Clutch which connects the engine to the wheels. The Clutch class accomplishes this by holding a boolean value that can be changed using the setter methods pressClutch() and releaseClutch(). Also included in this module are the classes FlyWheel which stores energy from the motor, as well as the Transmission which takes that energy and transfers it to the wheels in the form of rotations per minute. The Transmission class is an abstract class with two subclasses AutomaticTransmission and ManualTransmission which both instantiate the changeGear() method differently. The Automatic version changes the gear ratio based on how fast the vehicle is already moving, while the Manual version has to set by the user by calling the method.



The second module contains the classes that revolve around the motor and sending output to it. The Motor class has to receive energy in the form of amps from the MotorController and convert them to Horsepower for the Transmission. This is done through an equation based on the voltage of the electric Motor which is a parameter given during its construction. The MotorController class has to receive input from two sources at the same time (Battery and Potentiometer) and divert those amps to two different places (Motor and 12Volt Battery). The Potentiometer is the equivalent to the gas pedal in a car and is used to massively increase amp output to the Motor, this is accomplished using an equation based on pressure. While lastly the contactor is in charge of connecting the MotorController to the Battery, this is done using a boolean similar to the Clutch.



The final module focused on the Battery and other electric components that are in charge of supply the vehicle with electricity. The Battery class is in charge of outputting a consistent number of amps to a single destination. There are two instantiations of the class used in a Vehicle the main Battery which is in charge of supplying the MotorController and a secondary 12 Volt Battery which powers the vehicles lights, radio, etc. The DCDC\_Converter class is needed for the 12 Volt battery as it takes amps from the Motor Controller (which can be any Voltage) and converts them to amps at 12 Voltage which can be accepted into the 12 Volt Battery. Lastly the CircuitBreaker acts as a safety measure to keep the Motor from being overloaded with too much electricity. This functionality is accomplished using a boolean similar to the Clutch and Contactor classes.

Conclusion

Future work on this project would include expanding the number of cars available to the user for testing, as there is only the Ford Explorer 2002 available. Also, a visual element that shows the physical movements of the parts in the vehicle would greatly assist in learning.

In a world dominated with information without practical application, there is a demand to help educate young adults in real life skills. To do this the EVMS was created to as an interactive system for learning the intricacies of how electric vehicles work. By working on this project, I have learned coding semantics for more readable code, gained familiarity with IntelliJ, increased my domain knowledge on electric vehicles, and learned how to implement design patterns in large scale programming projects.