CJ Busca

July 1, 2021

Module 1-5 UML Diagram

A screenshot of a computer

Description automatically generated with medium confidence

Description of the UML Diagram:

The UML diagram depicted above represents a method of how we could implement the driver class, since the only vehicle here is the bicycle, the Bicycle class pulls information from the other two classes to determine what specifics make the bicycle. This class will pull the following attributes of the bicycle such as gears, cost, weight, and color. In this diagram the variables are set up for initialization. The program will prompt the user’s input (get methods). Once the information is pulled the variables will be contained. Once the program calls the driver class, it will pull the class that matches the standards of the program, in this example the Bicycle class uses and gains information for the class.

To explain this briefly, the bicycle class inherits from the two wheeled class, which also inherits from the vehicle class. The attributes contained under bicycle contain accessors and mutators.

**Explanation:**

The project contains of four classes:

* Vehicle
* TwoWheeled
* Bicycle
* Driver

The driver is the main class of this project, while the vehicle is a class without any attributes or methods, TwoWheeled is a class which is inherited by the Vehicle Class which also does not contain any attributes or methods. Finally, Bicycle class is inherited by TwoWheeled Class which has the following attributes and methods.

Atrtributes:

* Gears: Int
* Cost: double
* Weight: double
* Color: String

Methods:

* outputData(): void
* outputData(): Bicycle
* getGears(): Int
* getCost: double
* getWeight(): double
* getColor: String
* setGears(nbr: int): Bicycle
* setCost(amt: double) Bicycle
* setWeight(lbs: double) Bicycle
* setColor(theColor: String): Bicycle

According to Chapter 2, of *Hands on Design Patterns with Java*:

**Inheritance** is a crucial construct of object-oriented programming. It gives the engineer an idea of how an object is modeled through the relationships of certain structures. We can identify a Bicycle as an object with two-wheels which functions as a vehicle.

**Encapsulation** is a way to protect the data in our programs, you should only encapsulate objects as necessary. The way I see encapsulation is similar to the way you declare global/non-global variables in JavaScript or public/private in C++/C#. You only want certain methods to access your variables and typically you would declare a variable as private if you are utilizing it within the confines of the class. However if are attempting to use your variables outside of your class, you would want it to be declared as a public. Declaring a private means you are protecting the data from external methods and functions. Gears, cost, weight, and color attributes are all encapsulated within the Bicycle class.

**Polymorphism** means an object that appears in variant forms. Something from a personal example from game development, I remember working on a game where you are a wizard and you have different spells and potions. One of the spells that we included was a potion of polymorphism, this would change the wizard into a sheep or a beast of the like. In object-oriented programming this means that objects of different types are accessed through a common interface. This is accomplished through subclasses that override superclass methods. This would be extending our classes from the Vehicle Class, to the TwoWheeled Class, to the Bicycle Class.

**Portability** also known as modularity is the leading benefit of OOP. This is the class structure of the program. This makes troubleshooting each class easier, and makes the program more efficient to port to other systems or programs because the object is self-contained. Java itself is a highly portable language, and follows the mantra of write-once-run-anywhere, this is possible due to the Java virtual machine which compiles the program into bytecode.

References:

<https://www.c-sharpcorner.com/UploadFile/3614a6/accessors-and-mutators-in-java/>

Lavieri, E. (2019). *Hands-On Design Patterns with Java: Learn design patterns that enable the building of large-scale software architectures*. Packt Publishing.