Program #2
CS 202 Programming Systems

***** Make sure to read the Background Information first!**
It applies to all programming assignments this term***

****THIS IS NO DESIGN WRITEUP and NO UML DIAGRAM for Program #2 ****

Program #2 – Purpose

In our first programming assignment, we experimented with the notion of OOP and breaking the problem down into multiple classes building a relatively large system. The purpose of that assignment was to get to know how we can create classes that have different relationships. In program #2 our focus shifts to getting experience developing a hierarchy that can be used with dynamic binding – so that the application program can use one line of code to call one of many functions. To get the full benefit of dynamic binding, we will look at a smaller problem with a predefined design.

Program #2 – General Information

- In Portland, there are a variety of new ways that we can become more mobile around town without owning a car. Besides mass transit (like the MAX or the streetcar), we now have the option to get on a scooter for short distances, a bicycle for rent, or check-out a zip car. The companies that manage these transportation options need software to help them manage **when a item is checked-out, where it is located (what intersection of streets), and the amount of money they will make for current trip and for all trips in total. **THIS IS NOT a simulation of vehicles moving. NOTHING moves in this assignment….****

Program #2 – Building a Hierarchy using Dynamic Binding

For your second program, you will be creating a C++ OOP solution to support at least three different types of rental transportation (e.g., scooter, zip car, and one other). Create a base class for general transportation. This would be an abstract base class since we would never rent a general form a transportation – the user would rent a specific form (scooter, etc.).

Then, create three different derived classes for the specific forms of rental transportation you want to support. Remember to push up any common elements to a base class. Implement the following two choices and then ADD ONE OTHER of your own choice. Make your type of rental transportation similar to the other two but yet different!

- For a **scooter**, you would need the intersection where it is closest (e.g., two street names like 4th avenue and Mill Street), the scooter's ID number, the cost per hour, if it is checked out, the battery level, and if it needs repair (e.g., out of service). You may add to this.

- For **zip car**, you would need the intersection where it is closest (e.g., two street names like 4th avenue and Mill Street), the car's id number, the cost per hour, if it is checked out, the license plate, the gas level, and if it needs repair (e.g., out of service). You may add to this.
- For all of these we will also need to know the number of hours taken and the total amount that the company has earned for this form of rental transportation since it was last serviced. For zip cars, we need to be able to find out if the user has filled up the gas tank.

The purpose of this assignment is to use and experience dynamic binding. The requirement for this application is to have at least **three DIFFERENT types of classes**, derived from **a common abstract base class**! To use dynamic binding, there needs to be a self-similar interface among the derived class methods. In this case, for all types of rental transportation, the "state information" of the unit would need to be updated – is the unit checked-out or in need of repair, etc. In the real world, there will be some differences as well, although there shouldn't be too many. **Make sure to find at least one method that is different so that you can experience how to resolve such differences.**

Program #2 – Data Structure Requirements
With program #2, the company that manages these rental units needs to keep track of all of them with their current state information. ONE data structure will be used that can point to all three different kind of transportation forms using upcasting. The data structure will be a Doubly Linked List of base class pointers. Then using upcasting, each node can point to the appropriate type of rental transportation. Implementation of the required data structure(s) requires full support of insert, removal, display, retrieval, and remove-all. This DLL allows a client to build a list for their "fleet" of rental vehicles and then traverse through it to find out how much they have made ($), charge them, or take them back to the shop (removing them from the list). Please support both a head and a tail pointer.

ALL repetitive algorithms should be implemented recursively to prepare for our midterm proficiency demos!

Program #2 – Important C++ Syntax
Remember to support the following constructs as necessary:

1. Every class should have a default constructor
2. Every class that manages dynamic memory must have a destructor
3. Every class that manages dynamic memory must have a copy constructor
4. If a derived class has a copy constructor, then it MUST have an initialization list to cause the base class copy constructor to be invoked
5. Make sure to specify the abstract base class' destructor as virtual

IMPORTANT: ***OOP and dynamic binding are THE PRIMARY GOALS of this assignment!***