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 – General Background Story
In our first programming assignment, we developed a mass-transit program for streetcar monitoring, where we had cars on different types of lines, moving from stop to stop, and examined the process from the controller perspective. We experimented with the notion of OOP and breaking the problem down into multiple classes.

Now, we are going to examine mass-transit from a **rider's perspective**. **This is NOT an extension of the first program!** As a rider, there are a number of options when using mass-transit: streetcar, bus, MAX, tram, or even Uber. When a rider makes their decision about mass-transit, they may want to use information available to determine if this is the correct form of transit to use. They will want to know if one of the MAX lines is delayed or disrupted (e.g., is a bus being used instead due to ice on the lines)? They will want to know how busy the system is (on average) at different times of the day (e.g., at rush hour, the MAX has standing room only). They will also want to know about their previous experiences on that form of transit. Were they enjoyable? Would they want to ride again?

Program #2 – Building a Hierarchy using Dynamic Binding
For your second program, you will be creating a C++ OOP solution to support people who ride mass-transit, providing them with the information needed to make informed decisions. As part of this, the application will keep their "rider history" which will allow the rider to rate their experience.

THIS IS NOT A SCHEDULING PROGRAM. WE DON'T CARE WHERE THE BUSES, TRAMS, etc. are located or when/where they will arrive. ****** READ CAREFULLY!

The purpose of this assignment is to use and experience dynamic binding. The requirement for this application is to have at least **Three of the DIFFERENT forms of transportation supported**, 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 transit that you support, the user would like to find out the status (is it delayed?), find out how much it will cost to ride, find out how full they are running given the time of

day, and get information on what the rider thought last time they used this type of transit, and so on. In the real world, there will be some differences as well, although there shouldn't be too many. For example, for most forms of transportation $2.50 is all you need to pay and you can go from one form of transportation to another for a given period of time. But, Uber has a different fee structure (which you are allowed to simplify). Use this set of required functionality to get started but also think about what other functions make sense. **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

Every program this term will experiment with combinations of data structures. Implementation of the data structures requires full support of insert, removal, display, retrieval, and remove-all. Efficiency must be part of your data structure design.

You will need to implement data structures for:

1. **Rider History** – A Linear Linked List where each node is an array of M riding experiences, where the most recent riding experiences can be accessed without traversal.

2. **Popularity** (the average rider volume for time periods) – An array of transit pointers (to the abstract base class) and head pointers. Each array element will have a head pointer to a LLL for the popularity at different time intervals. (A LLL is chosen so that different transit types can support "time" uniquely – for example, MAX might give hour-by-hour volume whereas the Tram might give more general rush_hour, mid-day, and weekend volume of ridership).

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!***