Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Rideshare Project APCS**

*Due date:*

**Objective**

Imagine a long, single, straight 31-mile length of road with ‘stations’ located at each mile (so 32 stations in total). A certain number of cars, driven by an unnamed driver, each with a random starting station and destination station are traveling along the road at the same rate. Each car can hold 3 passengers.

A certain number of passengers are also populated at each station. Every passenger has a destination station they are trying to get to. If a traveling car is heading in the correct direction and has space available, a passenger travels with the car either until the car or passenger reaches its destination.

If there are 50 passengers, what percentage of them make it to their destination if there 20 cars? What percentage of passengers make it if there are 40 cars?

**Program Specifications**

Create a series of files to simulate the scenario as well as a runner file that makes use of your other files. It is up to you to determine the specifics of how your simulation is handled.

Submit a single zip file that includes a txt file with a short writeup of what your findings show as to which scenario produces a higher average revenue per car.

**Notes**

* This is one of the most complex projects we will work on in this class! Before starting, it’s helpful to sketch out what classes you want to design and what fields/methods they will have.
* One of the biggest challenges in this project is designing the *dependencies* that each class will have with one another. Some classes may have lists of other classes, which makes the aggregator class *dependent* on the smaller class.
* You’ll also want to consider who ‘knows’ about who. Does a passenger ‘know’ what car its in? Does a car ‘know’ what station it’s at? The answer can vary depending on how you’re choosing to implement a solution!
* When starting your implementation, keep the number of cars, passengers and stations manageable. It will help to put things in a predetermined position so that you can check for bugs more easily. Only once you have confidence your system is working in a non-random environment should you start incorporating random starts.
* Keeping track of revenue from each car should likely be your last step of implementation.
* This project will be helpful to include toString() methods for each object you create so you can easily see the information about each object. If you want to also include the memory address of the object in its toString(), you can use super.toString() in your implementation. Here is an example of what the toString might look like for your car…

/\*\*

\* Overwrites the toString to produce useful information on the car

\*/

public String toString(){

return super.toString() +

“[loc=” + loc +

“,dest=” + dest +

“,passengers=” + passengers + “]”;

}

**Rubric**

|  |  |
| --- | --- |
| **Description** | **Points Possible** |
| The Rideshare program effectively and accurately keeps track of information regarding passengers, cars and stations throughout the program. | 30 |
| The Rideshare program simulates the scenario described accurate to the logic of the problem described. This includes components such as handling car capacity, picking up and dropping off passengers and stopping passenger/car movement once a destination is reached. | 30 |
| The program accurately calculates and displays the average revenue per mile of cars once the simulation is complete | 5 |
| The runner function of the program displays information through each iteration of the simulation (ideally through the use of toString methods). The information displayed does not need to be stylized, but should be understandable on execution. Only one of the two scenarios needs to be ready to run, but through manipulation of variables or input the other scenario should be easy to reconfigure. | 15 |
| The submitted project includes a README.txt or README.md file that gives a short description of the program as well as the results of running both simulation scenarios. The file should also describe any components you’re particularly proud of and challenges you faced while making the project. | 5 |
| Code is neatly organized with comments as needed (particularly for less-obvious methods) for readability. | 5 |
| Proper encapsulation of objects is used (i.e., the use of private fields) throughout the program. | 5 |
| A git file is included with reasonable commit history or the program is hosted on GitHub. | 5 |