***Group Work – Taxi Company***

**16.17**

The steps provided from the book shows the tasks laid out in an order that doesn’t seem to be following a logical structure, as when it comes to designing software, to figure out what is the best layout to have a minimal viable product, you would want the features that are important to have in the first iteration of the project, so to have the basic features first, I believe the tasks should be in this order:-

* Enable a single passenger to be picked up and taken to their destination by a single taxi
* Enable a single passenger to be picked up and taken to their destination by a single shuttle
* Ensure that details are recorded of passengers for whom there is no free vehicle
* Provide sufficient taxis to enable multiple independent passengers to be picked up and taken to their destinations concurrently.
* Enable a single shuttle to pick up multiple passengers and carry them concurrently to their destinations.
* Ensure that taxis and shuttles are able to operate concurrently
* Provide a GUI to display the activities of all active vehicles and passengers within the simulation.
* Provide all remaining functionality, including full statistical data.

And the reason why is that the structure is more better to work with as you try to do one instance of the taxi to figure out the how to calculate the route, which then is the same for the shuttle, which it’s then you implement the multiple feature as it would be similar code except storing it in an array of some sort.

From the tasks shown, it seems that the complexity of the tasks is just right in terms of implementing and design as it is similar structure of code regardless if the vehicle is taxi or shuttle, due to the fact that it does not ask for many complex features out of scope with the idea of the software.

**16.18**

I think Tests should cover every aspect of possibility of success and failure so that when something that shouldn’t happen, there should be a way to cover that.

**16.19**

The Act Method in simplest form checks for location and passengers, but seeing if there is a location specified and passenger given, and gives an action to if it needs to move to a destination or pick up a passenger.

In the method, first it creates a Location variable called target and calls the targetLocation method which stores the destination. Afterwards it starts an if loop where it checks that if there is a target destination, then it will get the next location destination and store that in a new Location variable called next and set the location with the variable.

If the next variable matches to the target variable, it means the passenger has reached their destination, and the method notifies that they have arrived and offloads the passenger ready for the next one, if there is no passenger is just notifies that they have picked up the passenger.

If there is not target location set to a destination, then it will start to idle, counting up the seconds that it waits.

**16.20**

When it comes to storing the vehicles, depending on performance wise and what could be classed as and efficient way to store the list, I feel that keeping them separate would work well as you can use this to prevent exceptions such as a vehicle being used twice, as just moving between the list would be beneficial, however this is all pending on how much time does it take to switch between the lists, and this can be achieved using isFree() method as it can check if the vehicle is free as well has no passengers, and this can be used to track between the lists.

**16.21**

From the project, and looking at the class, there is use of LinkedList and Maps which can be used to store many key/values of vehicles and passengers, however from the code, the map is <Vehicle, Passenger> which can cause problems as while passengers will be unique values due to the different names, vehicles may not, if the vehicle is stored under Taxi or Shuttle, this will mean that each loop will cause the previous map to be overridden.

From this, it means that the key must be unique to allow the concurrent way to dispatch taxi/shuttles in a structured manner without losing details about customers.