Code review criteria:

1. I first checked if all the variables and function had meaningful names. This was to prevent confusion for anyone who has to review or maintain my code in the future. I was able to give all variables unique names but just to make it easier, I added comments to explain what each variable and function meant and was supposed to do.
2. All data errors were considered in each method. However no tests were written for them as I assumed that if I already check them in the method and it throws an exception if the wrong data is inputted then I would not have to test it.

```
lic static ArrayList<String> nearestNeighbourApproach(Graph<LongLat,NodeEdges> g,Ha
 if(g==null || pickUpNodes == null || deliveryNodes == null){
     System.err.println("Input cannot be null. please check the inputs");
 }
```

3.
4. I decided not to throw exceptions in the program. Instead I used System.err which gives an error message instead of the JVM printing a confusing stack trace for the user. All errors were appropriately given an error message.
5. I also checked the format of the code and refactored the code in many places to make it easier to read and to optimize performance. For example, I refactored my if else statements with more than 2 cases to switch blocks.
6. I made sure the code was easy to review, maintain and upgrade by adding proper documentation to each function as shown below.

```
/**
 * since the drone can only move to nodes on the graph, the delivery coordinates of an order may not exactly be a node
 * the method finds the node that is closest to the delivery coordinate. since the distance between each node is 0.00015 degrees
 * there will definitely be a node that is considered "close to" the delivery coordinate
 * @param hexGraph the graph of valid nodes and edges
 * @return a hashmap of order numbers to the delivery nodes
 */
1 usage   Pradnesh Sanderan *
private static HashMap<String,LongLat> getDeliveryNodes(Graph<LongLat,NodeEdges> hexGraph){
    if(hexGraph==null){
        System.err.println("Input cannot be null");
    }
    ArrayList<String> orderNos = Orders.orderNos;
```

Code review techniques :

1. I conducted pair programming with a peer to get feedback on the quality of my code, its correctness and readability.
2. I used code refactoring with the built in refactoring tool in the intellij editor to help change the design of the code to make it more readable sometimes optimized.
3. I intended to use code review tools such as FindBugs and JArchitect to help find any defects and improve the quality and functionality of the code but did not have time to do so.

Constructing a CI pipeline:

1. I would first choose a CI tool that supports the programming language used in my project. In this case, I would choose Jenkins since it offers support for Java.
2. I would then store the code in a version control system. I would choose git as it is widely used and I am familiar with it.
3. I would then automate the tests with a testing framework. Since I am using Jenkins I would choose Junit or TestNG
4. To check the quality of the code before deployment, I would use a tool like CodeClimate.

5. The deployment would be automated so that only the latest version of the software is deployed each time.
6. I would automate the sending of feedback to the development team on the success or failure of the pipeline.

How the pipeline operates:

1. Whenever code changes are made to the source code repository and are merged to the main branch, the pipeline is triggered
2. The automated tasks stated above (automated tests, code checking and sending of feedback) are executed.
3. If a task were to fail then the pipeline comes to a stop and the developer who merged to the main branch would be notified of the failure and where it failed so that they can make the appropriate adjustments.
4. If it passes all the automated tasks instead, then this newest version of the software is deployed instead.