Francesco Fanizza – A20079172
CS 201 – Final Project Design – Phase 1
11/18/2016

1. Describe the user interface. What are the menu options and how will the user use the application?

The user interface will be a console-based (text input and output) user interface. The menu options will be:
(1) Find the nearest station to a user-specified location
(2) Find (search for) a station
(3) Generate a route between a starting station and destination station (both user-specified) (this should handle transferring between CTA lines)
(4) Add a new station to the CTA system
(5) Delete a station from the CTA system
(6) Exit the program

The user will access menu options by entering a number corresponding to a menu action. Following a valid input, the program will run an application class method to run the option-specific code. If there is a bad input, the program will tell the user so, and re-prompt the user. More functionality may be added later on if time permits. All these menu options will be nested in a do-while loop that exits upon a flag being changed when the exit option is selected.

2. Describe the programmers' tasks.
    a. Describe how you will read the input file.
    b. Describe how you will process the data from the input file.
    c. Describe how you will store the data (what objects will you store?)
    d. How will you demonstrate polymorphism?
    e. How will you add/delete/modify data?
    f. How will you search data?

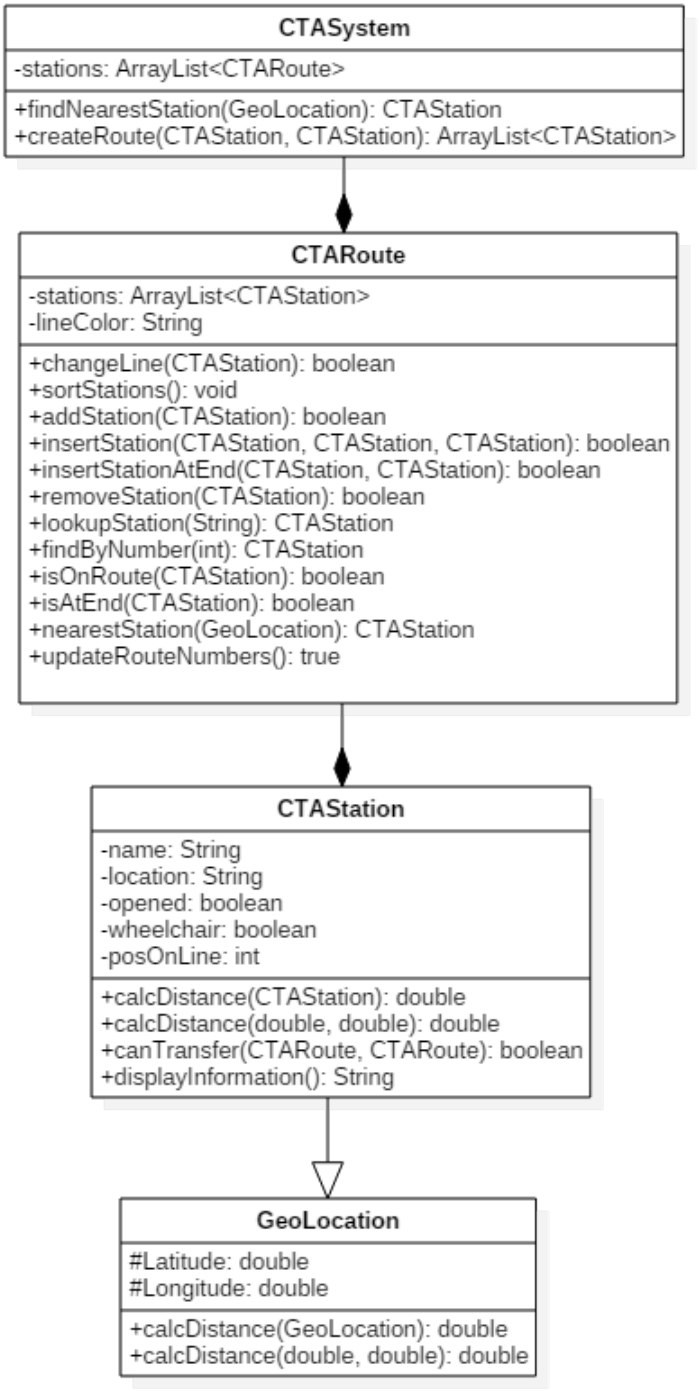
The file will be read in a separate method, which will be called at the beginning, and after any modification to the data file. The reading method will loop over all lines of the csv file, and store each field in a CTAStation instance variable. Collections of CTAStations will be placed into CTARoute variables (as ArrayLists), and collections of CTARoute variables (as ArrayLists) will be placed into a CTASystem instance variable. Polymorphism will be demonstrated in the getDistance() methods shared by GeoLocation and CTAStation. Data will be added/deleted/modified with class methods, and then overwriting the input file upon exiting. Similarly, data will be searched with class methods which reference isEqual() methods. Later on, time permitting, I will expand the search method to ignore non-letter symbols, and possibly search for best matches using substrings to compare to the station names.

3. List the classes you will need to implement your application.

The classes I will use in my application are GeoLocation, CTAStation, CTARoute, and CTASystem. Depending on whether the functionality of my application expands, I may choose to add more classes.

4. Draw a UML class diagram that shows all classes in the program and their relationships.

My initial prototype for my CTA system program will follow the following UML Class diagram. It is likely that parts of this diagram will be altered as I discover better ways to complete certain tasks, or organize data.



5. Think how you will determine if your code functions are expected. Develop a test plan based on the above description; how will you test that the expected outputs have been achieved? Be sure this test plan is complete. Your test plan should minimally:

a. Test each option in the menu-driven user interface for
    i. Expected behavior
    ii. Error-resistant input
b. Test that you can demonstrate polymorphism with your application-specific method
    i. Test results should confirm polymorphism

I am hoping to develop unit tests to test the functionality of each class' methods. These tests will check common (and expected) values and boundary values (0 and anywhere where conditionality specifies a certain value). These tests will be able to be run quickly and test a large scope of the project, and hopefully cover most of the sources of potential errors.

To test the input from the user, I intend on testing a sample normal input, including boundary values. Together these should test the expected behavior of the menu and program. I will also test bad user inputs, including null entries and incorrect variable types. These tests should ensure that my code is error-resistant, even from the most incapable user.

To confirm polymorphism, I will test a method in its super class and in the inheriting class. Each test should result in an appropriate result. I hope to use unit tests again to verify that the intended method is being used in my code, but I can also create flags, use the debug menu, or simply write out to the console during testing.