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

The user interface will start off with 8 numbered option:

- Add station
    - Will then ask the
        - Name
        - Latitude and longitude
        - Elevated, surface, embankment, at-grade, and/or subway
        - Handicap accessible (Y/N)
        - And which lines it is on (colors)
    - Print out the new stations information using toString()
- Remove station
    - Search for station method
        - Will ask for specification if necessary
    - Then remove using arrayLists
- Edit Station
    - Searches for station
        - Will ask for specification if necessary
    - Provide list of information that they might want to edit:
        - Name
        - Latitude and longitude
        - Elevated, surface, embankment, at-grade, and/or subway
        - Handicap accessible (Y/N)
        - And which lines it is on (colors)
        - Exit
            - Loops through these options until the user exits so they can edit multiple parts of the station
- Search for station
    - Enter name
        - Searches through list of names and returns the toString() of that station and for however many there are
- Search with requirements
    - Provide list of requirements they can search with
        - Handicap accessible (Y/N)
        - On certain line(s)
            - Would provide a list of the colors for them to choose from
- Nearest station
    - Asks the user for their latitude and longitude, calculates the nearest station
        - Using a method in point class
- Generate path

    - Asks for station the user starts at and ends at
    - Checks for what line each of them are on, then looks on those lines to see if they have a common station, if they do, print out stations leading to that one, tell the user to switch lines there, and continue with the other stations until their desired station
- Save
    - Write to file
- Exit
    - Write to file and exit

2. Describe the programmers' tasks:

a. Describe how you will read the input file.

- Read in 1 line at a time while the file still has a nextLine() - starting at index 1 to account for the header
- Split the line by comma, since it is a csv
- Store the station name as an instance of the of the station class
    - The rest of the data will be instance variables of that class
        - Point class
            - Double lat - latitude from csv
            - Double log - longitude from csv
        - Description - String
        - Wheelchair - boolean
        - All the lines will have their own int variable in the station's instance

b. Describe how you will process the data from the input file.

- The instances of the colored lines will be made after reading in the data
- arrayList of all lines will be made and will have a list of stations after checking to see if they DO NOT have the value of -1 for their respective color
- All the lines will then be sorted by numerical value so that pathfinding will be easier - using getRESPECTIVECOLOR() which will return an integer value
    - Sorted as a list of station instances
- Add all stations to a list - sorted alphabetically

c. Describe how you will store the data (what objects will you store?)

- arrayList of type line - different colors will be instantiated - with a list of stations in them
- Station class
    - Point class
        - Double lat - latitude from csv
        - Double log - longitude from csv
    - Every station will have an instance of point in it called location -
        - private point location;

- Description - String
- Wheelchair - boolean
- All the lines will have their own int variable in the station's instance
  - Private int red; private int green...

d. How will you add/delete/modify data?

Will search within the master station list - which is sorted alphabetically - and return the options found - asking for clarification if needed. Then performing the desired task of the user - whether it be delete, print, or allow modification

For delete it will look in the stations instances to find what lines it is on and then remove it from the arrayList for each color and remove it from the master list of stations.

In order to add:

- Add station selected:
  - Will then ask the user for
    - Name
    - Latitude and longitude - create of point
    - Elevated, surface, embankment, at-grade, and/or subway
    - Handicap accessible (Y/N)
    - And which lines it is on (colors)
      - Will add them to the respective color lines the user picked - in order
      - Also add it to the master list
  - Print out the new stations information using toString()

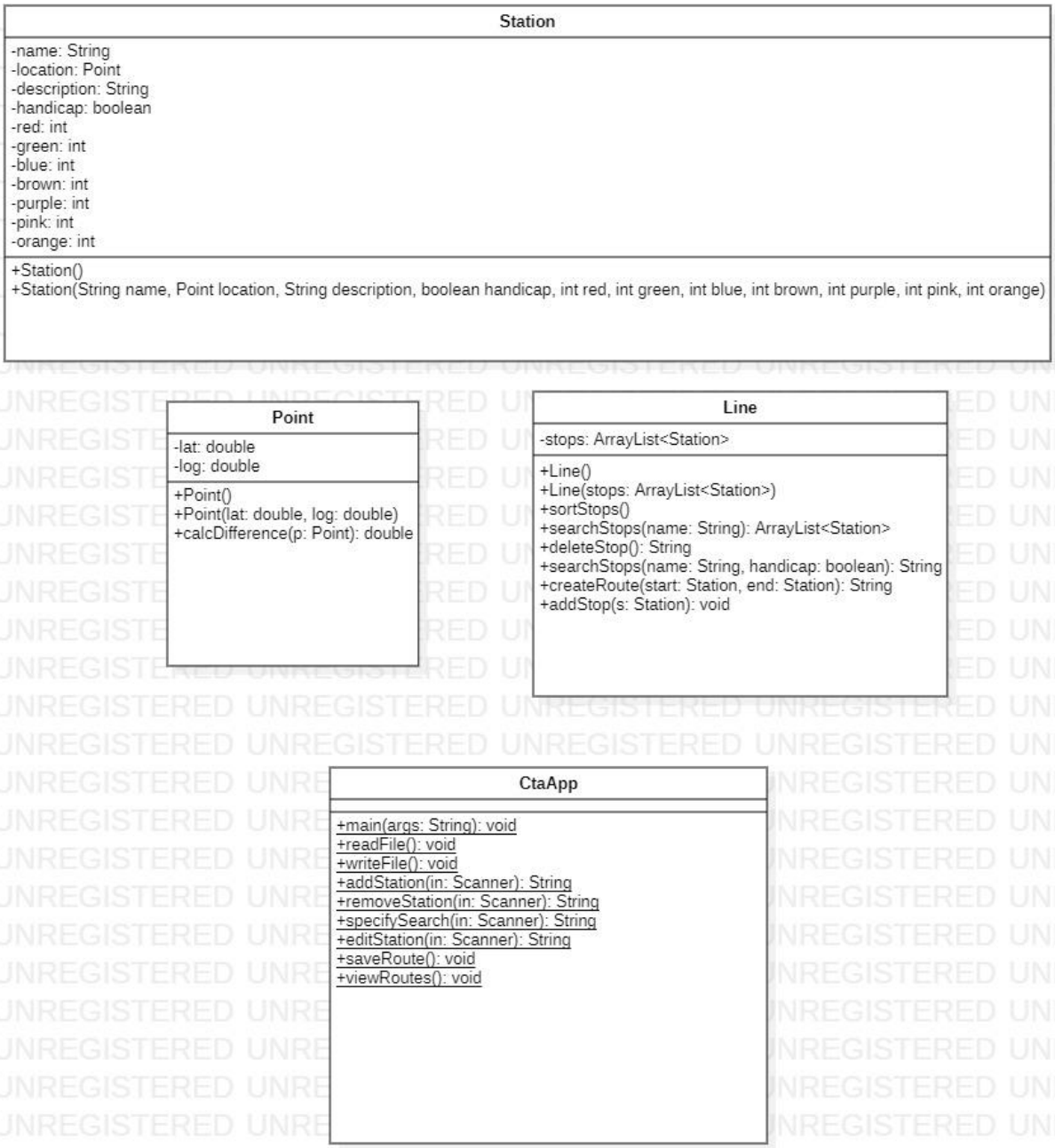
e. How will you search the data?

Search within the alphabetical list of stations returning multiple if needed

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

- Stations
- Point
- CTAapp
- Line

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



4. **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 for every menu option?**

Point:	Station:	Line:	CtaApp:
Test constructors: default and non-default	Test constructors: default and non-default	Test constructors: default and non-default	User menu prints properly
test getters and setters	test getters and setters	test getters and setters	reads the file properly
test calcDifference for computational accuracy		test sorting stops - will ensure accuracy by printing out the arrayList after sorting	can print to the file properly
Try to break the program: prevent exception/crashing		test adding stop - print the arrayList after adding the stop to ensure proper placement	addStation - calls the function from the line class properly
		test searching stops - print out the found stop, make sure the program asks for clarification if needed	remove station - calls the function from the line class properly and executes as expected
		test searching stops with parameters- checking data output for accuracy with the stop name and desired parameters - asking for clarification if needed	specifySearch - will take an ArrayList from search and provide the options found to the user to ask for clarification. Making sure does so properly and in a user friendly way

		test deleting stops - print out arrayList after deleting to ensure the function executed properly	editStation - properly is able to call the setter methods and lets the user know if there is an error with their input
		test createRoute - check the CTA map online to the make sure that the route works, outputs the stops on the way correctly, and is efficient route	saveRoute - saves the route that the user just create to a seperate csv file that can be viewed later
			viewRoutes - dispays all routes from the csv file that the user has saved