1. In my application, I have a total of four additional classes to Main – State, Tweet, HashTag, and MyTime. I first looked at the assignment requirements at a high level and though about what pieces of information I could collect together. First off, I realized that Tweets had certain information and States had certain information and therefore could be their own classes.

I used these classes to store all the information contained in the respective files of states and tweets. In my main function, I read the tweets and states files and store all information two ArrayLists of states and tweets.

My Tweet class takes in three arguments in the constructor – the text of the tweet, the location of the tweet, and the date-time of the tweet. The tweet class has getter methods for the x coordinate, y coordinate, text of the tweet and the time of the tweet.

My State class takes in three arguments – the name of the state and the x and y coordinates. Again, there are getter methods for these fields. Additionally, in the State class, you can store all the Tweets (object) originating from that particular state. There are additional getter methods for the ArrayList of all Tweets and the number of total Tweets originating from that state.

One can use the main class to access the Tweet and State objects. The information is store in an organized manner as well. One can simply look up a state and get all tweets from it.

The other class is HashTag and it takes the name of the hashtag and the number of times it appears. This class was mainly created to ensure proper sorting of the hashtag in descending order of number of times each hashtag appears using comparators.

Lastly, the MyTime class was created to store the time and the number of tweets in that duration. This was again created to ensure sorting in any way using comparators.

All the additional classes interact in the Main class. The Main class is basically used to read in the data from the files. The data is then essentially organized using the Tweet and State classes. From the main, one can access different states – all the tweets from that state and any other information that is added to the state. From the Main function, one can also access all the tweets and the data associated with each Tweet like location etc. Thus, the Main class becomes the hub ensuring interaction of all the other classes and any changes to reading can be done in Main and any changes to the data can be made in the Tweet and State classes.

2. My design choice for the program is excellent because different types of data are segregated into their own classes and the code becomes much more compartmentalized.
   a) If the source of the input data is changed: the way to read the data and from where to read the data is handled in the Main function. Here, the way of reading data (Using File and BufferedReader currently) is decided and the name of the file is given in the command line arguments. If I need to change the source to some online location or some other directory, only changes need to be made are in the Main function. I can choose to give a different address for a file in the command line

arguments and the Main function can then look it up to read the file using Java i/o. There will be minor changes only in the Main function in the way the data is read if changed to an online location/directory and no other place would the code need to be changed.

b) If the format of the data is changed: The data is read from the file and then stored in the respective classes of State and Tweet. If there are additional fields in the format of the data or there are changes in the way some fields are stored (date and time, for example) then only the classes need to be modified. There are getter methods for every field in the class. The date-field for tweet for example, if changed, then modifications would be made to the way the Tweet class takes in the information and gives it out. The getter would be modified to read the date-time string in a certain manner and then output the string in the manner suggested by the client. There would be minimal changes to the Main class and most of the manipulation would be done in the respective classes. This is the same for the State class.

c) If the items listed in the main menu changes: My Main class has a separate menu() function that basically outputs the available options to the user and expects a response from the command line. After any option is chosen, the main menu looks at the option choice and matches with the respective function in the Main class that would output the desired result. For each option chosen from the main menu, I have separate functions in the Main class that do the job. Thus, if new options were to be added to the Main menu, I would just add the options in the menu() function and add additional functions that do the job for the chosen choice. Hence, to list one more option would just require one extra line of code in the menu() function (and of course a function to do the job in the Main class!).

3. A combination of variable name choice, indentation, comments, function headers make my code very easy to comprehend. These are explained below:

a) Variable name choice: there are multiple variables that have to be created and sometimes when the scope of a function is very large, it becomes difficult to keep track of what is what. To solve this problem, my code exercises the use of very descriptive names for variables. For example, in the counting frequency of hashtag function I use variables such as SplitText to check when text is split and LimitChecker to check for any characters limiting the string such as any punctuation.

b) Descriptive function headers: For every major function that I have, there is a function header that basically gives a summary of what the function does along with the expected input and expected output. For example, the tweetsPerHour() function has the function header which explains that the function prints out the date and time to the nearest hour and the number of tweets containing the chosen phrase in that hour. This also tells that there is no expected input. A user readin the code for the first time will thus get a very good descriptive high level understanding of what the function does and would not need to go through it completely.

c) Well commented, indented, and non-superfluous code: The overall style of the code in terms of the ending and spacing between the various chunks of code renders the code effortlessly comprehensible. Just by looking at the code, the user understands if a certain piece of code does something related or unrelated to the code above and below. Furthermore, the code does not go beyond a certain set of characters on a line so the user does not have to scroll right from time to time. Additionally, there are small comments sprinkled intermittently in the code that aid in understanding. If a user who knew nothing about the code reads the code, he will have these aids in getting the gist of the code.

4. While my code generally follows standard best coding practices, there are two specific instances of memory efficiency and speed efficiency that I highlight below:
   a) String as opposed to StringBuilder: the String data type is very inefficient when it comes to concatenation – StringBuilder does the job in O(N) time as opposed to String's O(N^2) time. However, the String data type is much quicker when it comes to using substring and is almost twice as fast as the string builder. In my program, I use the substring method countless times and therefore use the String data type for this purpose. For example, in trying to read hashTags from the text of the tweet, the substring method was necessary and if I had used StringBuilder, I would have had to use the substring method almost 150K times and the StringBuilder data type would have detrimentally impacted my efficiency. Hence, using String for substring purposes, I greatly improve the speed of my computations. This is an example of strength reduction to optimize the code.
   b) Computations on demand: In my Main class, I only do computations on the data when required. For example, after parsing the data files, I do not right away calculate the origins of each tweet. I only compute the Euclidean distances of each state whenever a function asks for it. Additionally, since I have to go through all the tweets and all the states to determine whether a tweet comes from a specific state, in that one computation I store all the tweets in respective states. In this manner, I do not have to do the distance computations again and again. I just save all the tweets from a particular state in that state class. In this way, I just do the computation to get tweets from one state and in return get to organize all the tweets on all respective states. Thus, if I have gotten all the popular hashtags in Wyoming, I do not need to determine all the tweets coming from New Jersey when finding the most popular hashtag from there.

5. Since my code is compartmentalized into different classes and all the different classes are organized into various functions, it becomes very easy to unit test these separate functions.

For purposes of testing, I use a variety of things – Print systems to test for expected outputs, JUnit tests to test whether functions work on a set of inputs, smaller data files to easily determine whether the outputs are correct, and testing on invalid input – explained in detail below.

For example, in checking whether the file readings are done properly, I first test for irregular input in the command lines. I use fewer arguments to check whether an error is shown. I use the right number of arguments to but with invalid file names to check whether an error is thrown or not. Then, I put a different file name that I created to check whether the data is read correctly. I use this small file and print statements to determine whether the data is read correctly and is organized into the different classes.

Furthermore, to check the whether the functions in the Main file work properly, I use JUnit testing using the new small file that I created. For maximum number of tweets in states, for example, in my JUnit test, I check whether the output is as expected or not. This is basically a list of three states that I created with a known correct output and my JUnit test case checks for that.

To check for most popular hashtags, I again use JUnit testing. Here, I check for multiple things. I check for abnormal input to see if there are errors thrown and the program terminates. Then, I check for other smaller parts. Whether the total number of hashtags read is correct, whether there are 10 or less than 10 hashtags in the output, whether the name of the state was read correctly, whether the log file is recording the data. I try to test in an incremental manner where I check for very basic things like whether total number of hashtags is correct or not then I move on to bigger parts which incorporate the smaller parts like whether the hashtags have proper counts or not.

In this manner, I test the correctness of my program and check all functions of all classes as well. This ensure that my program is implemented correctly.