# CIT 594 Module 11 Programming Assignment

In this assignment, you will apply the design principles and design patterns that we recently covered in class in developing a Java application to read text files as input and perform some analysis.

Although the correctness of your application is important, much of the grade on this assignment will be determined by how it's designed.

Please note that you have two weeks to complete this assignment, but it is likely to be more time-consuming than previous assignments. Please be sure to give yourself time to think about the design of your program and the organization of your code before you start implementing it.

## Learning Objectives

In completing this assignment, you will learn how to:

- Design a software system using an N-tier architecture
- Design software using the principles of modularity, functional independence, and abstraction
- Apply the Singleton design pattern
- Use a Java library to read data stored in a JSON file

## Background

Government agencies such as the Centers for Disease Control can use social media information to get an understanding of the spread of infectious disease. By analyzing the use of words and expressions that appear over time on platforms such as Twitter, Facebook, etc., it is possible to estimate where the disease is affecting people and whether it is spreading or appears to be contained.

In this assignment, you will design and develop an application that analyzes a small set of Twitter data to look for tweets regarding occurrences of the flu and determines the US states in which these tweets occur.

Information about the format of the input data, the functional specification of the application, and the way in which should be designed follows below. Please be sure to read all the way through before you start programming!

# Input Data Format

Your program needs to be able to read the tweets from both a tab-separated file and from a JSON file; the user of the program will specify which input type to use. The formats of these files are described below.

## Tweets: Tab-Separated

Each line of the file contains the data for a single tweet and contains four tab-separated fields:

1. The latitude and longitude of the location of the tweet. This field is demarcated by square brackets and the latitude and longitude are separated by a comma.
2. An identifier used by the collector of the tweets. This field can be ignored for our purposes.
3. The date of the tweet in YYYY-MM-DD hh:mm:ss format
4. The text of the tweet.

The following is an example of the data for a single tweet:

```
[41.38, -81.49]    6     2019-01-28 19:02:28      Yay, homework!
```

## Tweets: JSON

JSON ("JavaScript Object Notation") is a popular standard for exchanging data on the World Wide Web. Put simply, a JSON "object" is delineated by curly braces, and each field is indicated as comma-separated key/value pairs, like this:

```
{
 "product": "MacBook Pro",
 "price": 1350.0,
 "in stock": true,
 "shipping options": ["ground","air"]
}
```

Note that, in the above examples, "shipping options" is mapped to an array of values (containing "ground" and "air"), whereas the other keys are mapped to a single value.

There are numerous Java libraries for reading JSON objects, and numerous tutorials on how to use them. For this assignment, we're going to be using the JSON.simple library. Download the **json-simple-1.1.1.jar** file and add it to your Eclipse project's build path. Documentation for this library is available here.

Your Java program can use the JSON.simple library to open a file containing an array of JSON objects and then parse each object and get its constituent parts.

For instance, let's say we have a file named "dogs.json" that contains the following text:

```
[{"name":"Jake","age":4},{"age":5,"name":"Riley"}]
```

This represents an array of JSON objects, each of which has a "name" and an "age"; the fact that these are in different orders in each object does not matter.

The following code would use the JSON.simple library to read the file and print each dog's name:

```
// create a parser
JSONParser parser = new JSONParser();

// open the file and get the array of JSON objects
JSONArray dogs = (JSONArray)parser.parse(new FileReader("dogs.json"));

// use an iterator to iterate over each element of the array
Iterator iter = dogs.iterator();

// iterate while there are more objects in array
while (iter.hasNext()) {

      // get the next JSON object
      JSONObject dog = (JSONObject) iter.next();

      // use the "get" method to print the value associated with that key
      System.out.println(dog.get("name"));
}
```

*Note*: Although you may use the above code as a starting point, be careful about copy/pasting it into Eclipse, as the double-quotes and other special characters may cause errors. You may need to change those before compiling your code.

**Also: do not attempt to write your own code to parse the JSON file!** Although it is possible, it would be extremely time-consuming, and not the point of the assignment. Use the JSON.simple library that we have provided and base your code off the sample above.

## States

In order to determine the state from which each tweet originated, your program will also need to read a file that contains the latitude and longitude of the center of each of the 50 states, plus Washington DC, in comma-separated format. Each line of the file contains the data for a single state and contains the name of the state, the latitude, and the longitude. Following is an example:

```
Alabama,32.7396323,-86.8434593
```

### Sample Input Files

Your program will be evaluated using the following input files:

- a set of 10,000 tweets in tab-separated format (flu_tweets.txt)
- the same 10,000 tweets in JSON format (flu_tweets.json)
- a CSV file listing the centers of the 50 U.S. states and Washington, D.C. (states.csv)

Download the three files and add them to your Eclipse project's root directory so that you can test your program.

You can, of course, create your own input files for testing, but the correctness of your program will be determined using the files we provide.

## Functional Specifications

This section describes the specification that your program must follow. Some parts may be under-specified; you are free to interpret those parts any way you like, within reason, but you should ask a member of the instruction staff if you feel that something needs to be clarified. Your program must be written in Java.

### Runtime arguments

The runtime arguments to the program should specify, in this order:

- the format of the tweets input file
- the name of the tweets input file
- the name of the states input file
- the name of the log file (for logging debug information; see "Logging" below)

Do not prompt the user for this information! These should be specified when the program is started (e.g. from the command line or using an Eclipse Run Configuration). If you do not know how to do this, please see the documentation at https://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html.

The program should display an error message and immediately terminate upon any of the following conditions:

- the number of arguments is incorrect
- the format is neither “json” nor “text” (case-insensitive)
- the specified tweets file or states file does not exist or cannot be opened for reading (e.g. because of file permissions); take a look at the documentation for the java.io.File class if you don’t know how to determine this

For simplicity, **you may assume that the tweets file and states file are well-formed according to the specified formats**, assuming they exist and can be opened.

You can also assume that if the first argument is “csv”, then the second argument specifies a well-formed CSV file, assuming it exists, and that if it is “json”, then the second argument specifies a well-formed JSON file, assuming it exists.

These are pretty big assumptions but will greatly simplify this assignment!

Last, if the log file does not exist already, your program should create it and not show any error message. That is, **you should not assume that the log file already exists.**

## Identifying relevant tweets

Your program should print the number of “flu tweets” per state in the input file.

A “flu tweet” is a tweet in which the text satisfies one of the following:

- contains the word “flu”; note that it can be at the start, in the middle, or at the end
- contains the hashtag “#flu”
- contains a string that starts with “flu” and is followed by a *single* character that is not a letter, e.g. “flu!”

Your program should treat the tweet text as case-insensitive, e.g. it should consider instances of both “flu” and “FLU”.

The table below provides examples of text that should and should not be considered “flu tweets”:

| Text | Flu tweet? |
|---|---|
| I feel like I have the **flu** and I hate it | Yes |
| **Flu** symptoms are the worst | Yes |
| I definitely have the **flu** | Yes |
| I think I have the #**flu** I'm so sick | Yes |
| How would I know if I have the **flu**? | Yes |
| Five days I've had the **flu**! so sad | Yes |
| That bunny is so **flu**ffy I wanna squeeze it | No |
| Don't be in**flu**enced by fake news | No |
| so sick with the #**flu**e gonna go home now | No |

Although the last one is arguably a tweet that we should consider, you do not need to do so for the purposes of this assignment.

Please do not spend too much time worrying about what is and what is not a "flu tweet," as that is not the main point of the assignment. We're not trying to trick you, we promise! As long as your code works correctly for the examples provided above, and is case-insensitive, you'll be fine.

As you've surely noticed, this approach to identifying "flu tweets" is extremely simplified and probably not super-accurate, but determining whether a tweet really does indicate that someone has the flu is waaaay outside the scope of this assignment. So even if you come up with a more accurate solution, please be sure to follow the specifications described here!

## Determining location of tweet

Once you have found the "flu tweets," you will need to determine the state in which each originated. To identify the location of each tweet, use the coordinates of the center of each state (according to the states file) and assume that the tweet originated in the state to which its coordinates are closest; you can break ties in any manner you choose. Determining the distance between coordinates is a little tricky since the earth is a sphere and not a plane, but for simplicity you should use the Euclidean distance. Although this approach is not completely accurate, it's good enough for our purposes.

As with detecting flu tweets, you may discover that this approach to determining the state from which a tweet originated is not entirely accurate either, e.g. Philadelphia is closer to the center of New Jersey than it is to the center of Pennsylvania. Regardless, please use this approach for simplicity.

## Logging

For each “flu tweet,” write its text and the name of the state in which it originated to the log file that was specified when the program started. Be sure to write these to the log file and not to the screen! When writing to the log file, you should write a single tweet per line in the format state, tab, tweet. For example:

```
Alabama	I have the flu!
```

Each tweet should only be written to the log file once, but they can be written in any order you’d like. If a log file with the same name already exists when the program starts, its contents should be overwritten.

## Program output

When your program finishes looking for “flu tweets” and determining their locations, it should print the number of “flu tweets” per state to the console using System.out, with the state names listed in alphabetical order (hint: think about which data structure you can use to make this a bit easier!).

When writing to the screen, please format the state name and number of “flu tweets” as below, making sure there is only one listing per line:

```
Alabama: 1
Connecticut: 1
New York: 7
Pennsylvania: 5
West Virginia: 3
```

Note that these are not the correct outputs for the input files we provided; they are just given as an example of the output.

**Please do not post public questions in the discussion forum asking whether a certain output is correct!** It is up to each student to determine the correct output of the program.

You should not list states for which there are zero "flu tweets" and **the code you submit for grading should not print anything to the screen other than this output**.

## Design Specification

In addition to satisfying the functional specification described above, your program must also use some of the architecture and design patterns discussed in the videos.

In particular, you must use the **N-tier** architecture to identify and then separate your application into functionally independent modules.

To help with the organization of your code (and to help with the grading), please adhere to the following conventions:

- the program's "main" function must be in a class called Main, which should be in the **edu.upenn.cit594** package
- the classes in the Presentation/User Interface tier should be in the **edu.upenn.cit594.ui** package
- the classes in the Processor/Controller tier should be in the **edu.upenn.cit594.processor** package
- the classes in the Data Management tier should be in the **edu.upenn.cit594.datamanagement** package
- classes related to logging should be in the **edu.upenn.cit594.logging** package
- any other classes related to data that is shared by the tiers should be in the **edu.upenn.cit594.data** package

Additionally, you must use the **Singleton** design pattern to implement the logging functionality.

Your Main class should be responsible for reading the runtime arguments (described above), creating all the objects, arranging the dependencies between modules, etc. See the "Monolith vs. Modularity" reading assignment in Module 10 for an example if you are unsure how to do this.

Because your program must be able to read the set of tweets from either a tab-separated file *or* from a JSON file, you must design your application so that you have two classes to read the tweets input file: one that reads from a tab-separated file and one that reads from a JSON file. The code that **uses** that class should not care which implementation it's using.

The classes that read the input files should get the name of the input file via their constructor, passed from Main to whichever object creates them. The Singleton class that does the logging should have a static field for the name of the log file, which should be set by Main.

# Logistics

Please be sure to read the section below, as some of the details for this project are different from that of other assignments.

### Getting Help on the Discussion Board

As always, you are welcome to use the discussion board to ask clarification questions, get help with error messages (particularly when using the JSON library), and ask for general advice.

However, **please do not post public questions regarding the correct outputs** for the program. For instance, please do not post public questions along the lines of "My program says that Connecticut has 4 flu tweets; is that right?" It's important that all students determine for themselves whether their program is working correctly. Unlike other assignments, correct answers and test cases will not be provided in advance.

Likewise, please do not post public questions such as "Should I put the code that reads the JSON file in the Processor tier or the Data Management tier?" since answering that is pretty much the point of the assignment!

## How to Submit

To submit your solution, please create a .zip file containing all of your source code and upload it to the “Module 11 Programming Assignment Submission” assignment.

Please do not submit the JSON jar file or any of the input files.

## Assessment

This assignment is graded out of a total of 100 points. It will be graded by members of the instruction staff.

The design of your system is worth 50 points, based on the correct use of the N-tier architecture and the packages as described above. Be sure to follow the naming convention and be sure that the correct functionality is placed into the correct tier/package.

The implementation of the Singleton pattern is worth 10 points.

The implementation of your system is worth 40 points, based on its ability to do the following:

- Read commandline arguments
- Read the input files
- Identify flu tweets
- Determine the state in which a tweet originated
- Produce the output in the expected format
- Write to the log file