Penn Engineering
ONLINE LEARNING

CIT 594 Group Project Description

In this project, you will apply the design principles 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.

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
- Use a Java library to read data stored in a JSON file

Background

The OpenDataPhilly portal makes over 300 data sets, applications, and APIs related to the city of Philadelphia available for free to government officials, researchers, academics, and the general public so that they can analyze and get an understanding of what is happening in this vibrant city (which, as you presumably know, is home to the University of Pennsylvania!). The portal's data sets cover topics such as the environment, real estate, health and human services, transportation, and public safety.

In this assignment, you will design and develop an application that analyzes data from OpenDataPhilly regarding street **parking violations**, e.g. parking in a no-stopping zone, not paying a meter, parking for too long, etc.

Each incident of a parking violation includes various pieces of data such as the date and time at which it occurred, the reason for the violation, and identifying information about the vehicle.

A parking violation also has an associated **fine**, i.e. the money that was charged as penalty to the vehicle, as well as information about the location at which it occurred. For our purposes, we are concerned with the violation's **ZIP Code**, which is a numerical code used by the US Postal System to indicate a certain zone, e.g. a city or a neighborhood, as in the case of Philadelphia.

The program that you will write in this project will display the **total fines per capita** for each of Philadelphia's 48 residential ZIP Codes. That is, given a set of data about individual parking violations, your program must calculate the total amount of fines for each ZIP Code, and divide it by the ZIP Code's population, which will be provided to you in a separate input file.

Information about the format of the input data files, 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 set of parking violations from both a comma-separated file and from a JSON file, which is described below; the user of the program will specify which input type to use.

Note that the OpenDataPhilly data sets are quite large and detailed, so for the purposes of this assignment, we will provide a smaller data set to you; you do not need to download anything from the OpenDataPhilly site.

Parking Violations: Comma-Separated

Each line of the file contains the data for a single parking violation and contains seven comma-separated fields:

1. The timestamp of the parking violation, in *YYYY-MM-DD*T*hh:mm:ss*Z format.
2. The fine assessed to the vehicle, in dollars.
3. The description of the violation.

4. An anonymous identifier for the vehicle; note that some vehicles have multiple violations.
5. The U.S. state of the vehicle's license plate. For instance, "PA" = Pennsylvania, "NJ" = New Jersey, and so on.
6. A unique identifier for this violation.
7. The ZIP Code in which the violation occurred, if known.

Parking Violations: 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 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, based on the code sample above.

Population Data

In order to determine the population for each ZIP Code, your program will need to read a whitespace-separated file containing that data. Each line of the file contains the data for a single ZIP Code and contains the ZIP Code, then a single space, then the population.

Sample Input Files

Your program will be evaluated using the following input files:

- a set of around 25K parking violations in comma-separated format ("parking.csv")
- the same set of parking violations in JSON format ("parking.json")
- a text file listing the populations of the Philadelphia ZIP Codes ("population.txt")

Download **CIT594-project-files.zip**, unzip it, and add the three files 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.

0. Runtime arguments

The runtime arguments to the program should be as follows, in this order:

- the format of the parking violations input file, either "csv" or "json"
- the name of the parking violations input file
- the name of the population input file

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 runtime arguments is incorrect
- The first argument is neither "csv" nor "json" (case-sensitive)

- The specified input 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 input files are well-formed according to the specified formats**, assuming they exist and can be opened. However, note that the ZIP Code is missing for some entries in the input files; this is intentional and is something your program must handle as described below.

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!

1. Determining Fine for Each Violation

Although the overall goal of the assignment is to display the total fines per capita in each ZIP Code, your program must write the results of some intermediate calculations to a text file to assist with grading.

As your program reads the parking violations input file, it must write the ZIP Code and fine for each entry to a file called "fines.txt", which should be in the root directory of your Eclipse project.

However, your program must ignore (i.e., not write to the output file and not consider as part of further calculations) any parking violation for which the ZIP Code is unknown or for which the vehicle's license plate state is not "PA".

When writing to the "fines.txt" output file, write one violation per line and list the ZIP Code, then a single space, then the fine, like this:

```
19104 26
19123 76
19104 51
```

… and so on. The order in which you write these values to the file is up to you.

Also, your program should overwrite any data that is already in the "fines.txt" file each time your program runs, i.e. it should not append data to the file.

Be sure you write this information to a file called "fines.txt" and **not** to the screen!

2. Determining Total Fines for Each ZIP Code

In addition to determining the fine for each individual parking violation, your program will also need to calculate the total aggregate fines for each ZIP Code.

Once those values are determined, write the ZIP Code and the total aggregate fines to a file called "total.txt", which should be in the root directory of your Eclipse project.

When writing to the "total.txt" output file, write one ZIP Code per line and list the ZIP Code, then a single space, then the total aggregate fines, like this:

```
19104 3432
19103 1176
```

… and so on. The order in which you write these values to the file is up to you.

Please note that the above values are for demonstration purposes only and are not necessarily correct!

As mentioned above, be sure that your program is ignoring any parking violation for which the ZIP Code is unknown or for which the vehicle's license plate state is not "PA".

Also, your program should overwrite any data that is already in the "total.txt" file each time your program runs, i.e. it should not append data to the file.

Be sure you write this information to a file called "total.txt" and **not** to the screen!

3. Displaying Total Fines Per Capita

Finally, your program should display the total fines per capita for each ZIP Code, i.e. the total aggregate fines (as discussed above) divided by the population of that ZIP Code,

as provided in the population input file. This information should be displayed on the screen using System.out.println or similar.

When writing to the screen, write one ZIP Code per line and list the ZIP Code, then a single space, then the total fines per capita, like this:

```
19103 0.0284
19104 0.0312
```

… and so on.

Unlike in the previous parts, though, the ZIP Codes must be written to the screen in ascending numerical order (hint: think about which data structure you can use to make this a bit easier!).

Please note that the above values are for demonstration purposes only and are not necessarily correct!

As mentioned above, be sure that your program is ignoring any parking violation for which the ZIP Code is unknown or for which the vehicle's license plate state is not "PA".

Also, you should not list ZIP Codes for which there are no fines, 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
- any other classes related to data that is shared by the tiers should be in the **edu.upenn.cit594.data** package

If you do not know how to work with packages in Java, please see the documentation at https://docs.oracle.com/javase/tutorial/java/package/index.html

Your Main class should be responsible for reading the runtime arguments (described above), creating all the objects in the N-tier architecture, arranging the dependencies between modules, etc. but should not otherwise perform any actions typically associated with the N-tier architecture.

Because your program must be able to read the set of parking violations from either a comma-separated file *or* from a JSON file, you must design your application so that you have two separate classes to read the parking violations input file: one that reads from a comma-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.

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 19123 has an average of 0.052 per capita; is that right?” It’s important that all groups 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 in the Data Management tier?” since answering that is pretty much the point of the assignment!

If you need help and think that your question may give away a solution, please post a **private** note in the discussion forum or ask a TA during private office hours.

How to Submit

To submit your project, please create a .zip file containing all of your source code and upload it to the “Group Project Submission” assignment. Only one member of your group needs to submit the code.

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 40 points, based on the correct use of the N-tier architecture and the packages as described above. Each package/tier is scored as follows:

- Main: 2 points
- ui package: 10 points
- processor package: 12 points
- datamanagement package: 12 points
- data package: 4 points

The implementation of your system is worth 60 points, based on its ability to correctly do the following:

- Read commandline arguments (5 points)
- Read the input files (10 points)
- Identify ZIP Code and fine for each violation (15 points)
- Calculate total amount of fines for each ZIP Code (15 points)
- Display total fines per capita for each ZIP Code (10 points)
- Correct formatting of output files and display data (5 points)