Concordia University
Comp 249 – Winter 2019
Object-Oriented Programming II
Assignment 3
Due 11:59 PM - Sunday March 17, 2019

Purpose: The purpose of this assignment is to allow you practice Exception Handling, and File I/O, as well as other previous object-oriented concepts.

Java**S**cript **O**bject **N**otation or **JSON** is a well-known, lightweight data-interchange format, which is easily readable/writable by humans [1][2]. It is also easy for machines to parse and generate. JSON is based on data objects consisting of attribute–value pairs and array data types. It is widely used for asynchronous browser–server communication, including as a replacement for XML in some AJAX-style systems. As such, many journals and scientific servers are supporting this format. This format is easily extendable. The following is a typical sample (modified form an existing IEEE paper).

```
@ARTICLE{

8247289,
author={J. Park and J. N. James and Q. Li and Y. Xu and W. Huang},
journal={IEEE Transactions on Vehicular Technology},
title={Optimal DASH-Multicasting over LTE},
year={2018},
volume={PP},
number={99},
pages={15-27},
keywords={Forward error correction;Long Term Evolution;Maintenance engineering;Multicast communication;Resource management;Static VAr compensators;Streaming media;DASH;LTE;convex optimization;eMBMS;multicasting},
doi={10.1109/TVT.2018.2789899},
ISSN={0018-9545},
month={January},
}
```

In JSON, the fields however do not need to be placed in specific order. For example, in the above example, you can have the "number" filed for instance, be written above the "year" field, and so on.

There are many parsers available for JSON developed in may programming language as library, API, etc. Mendeley [3] is one example that uses this format. With Mendeley you can create your own library from all articles that you have read so far and you can use them when you want to write an article. Consequently, this is very good for inserting the needed references on a published paper and managing them afterwards. In particular, such tools can import the article(s) information from a bibliography file (.bib) and generate the reference(s) in particular format according to the conference or journal publisher standard.

For example, the representation of this file in IEEE format would be (this is a slightly simplified version than the actual one; however, you need to follow this format for the scope of this assignment):

J. Park, J. N. James, Q. Li, Y. Xu, W. Huang. "Optimal DASH-Multicasting over LTE", IEEE Transactions on Vehicular Technology, vol. PP, no. 99, p. 15-27, January 2018.

Or in **ACM** format would be:

[1] J. Park et al. 2018. Optimal DASH-Multicasting over LTE. IEEE Transactions on Vehicular Technology. PP, 99 (2018), 15-27. DOI:https://doi.org/10.1109/TVT.2018.2789899.

And finally, in **Nature Journal (NJ)**, which is one of the most famous journals in natural science, would be
J. Park & J. N. James & Q. Li & Y. Xu & W. Huang. Optimal DASH-Multicasting over LTE. IEEE Transactions on Vehicular Technology. PP, 15-27(2018).

You should have a "very" detailed look at these formats to see how they are mapped from the original record of the article.

In this assignment, you will be designing and implementing an alternative tool (to the existing ones), called ***AuthorBibCreator***. The main task of this tool is to read and process some given .bib files, based on a given author name, and creates 3 formatted files (IEEE, ACM and NJ) with all records of that author(s).

In short, the tool will require the user to enter an author name, then searches all .bib files for any articles for any author(s) with that name, and creates **3** different files with the correct reference formats for IEEE, ACM and NJ with all found records for that author name. The exact names of these files must be called: *author-name*-IEEE.json, *author-name*-ACM.json, and *author-name*-NJ.json, where *author-name* is the exact name of the author being searched. You should notice that the search is based on one single name (i.e. family name), and the tool will create the files to include **all** the records of any/all author(s) with that name. That is, the created files may include records of different authors who happen to have the same name. For instance, records for J. Park, and T. L. Park must both be included if the search was based on an author with the name Park. The resulted files in such case will consequently be called Park-IEEE.json, Park-ACM.json, and Park-NJ.json. As a side note, to distinguish our tool/application from other existing similar software, we will call these files *author-name*-IEEE.json, *author-name*-ACM.json, and *author-name*-NJ.json (although in fact, the created files contain the references to the articles, and not json records!).

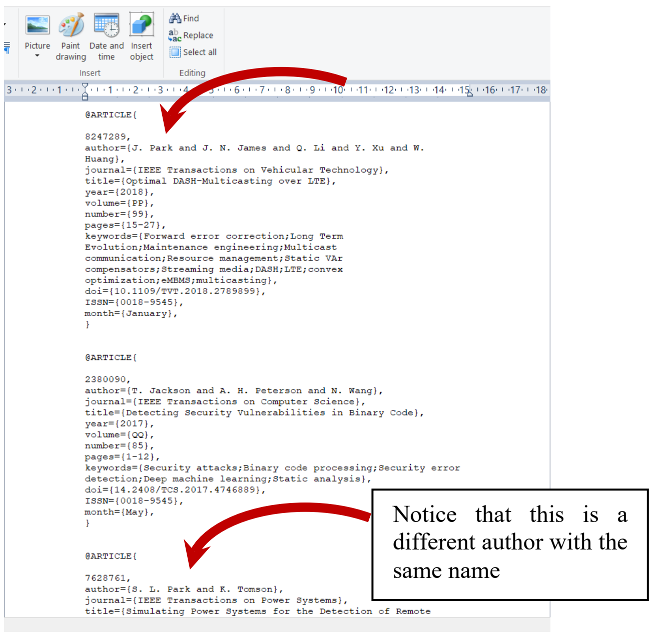
You should notice that the given .bib files themselves may include zero or multiple articles. These articles additionally may or may not belong to any authors with the name being searched.

In details, you are given 10 files, called *Latex1.bib* to *Latex10.bib*. Each of these files includes zero or more articles. You AuthorBibCreator tool will require the user to enter an author name to search for, then the tool will search all 10 input files (in one execution), detects all articles that belong to the user name being searched, and creates the needed IEEE, ACM and NJ files. If there are existing files with these names, then these files must be renamed as BU (Back Up) and any old BU files must be deleted (full details on that is given below). Each of these created files **MUST** be represented **exactly** based on the format of these files (one for IEEE format, one for ACM format and the last for NJ format).

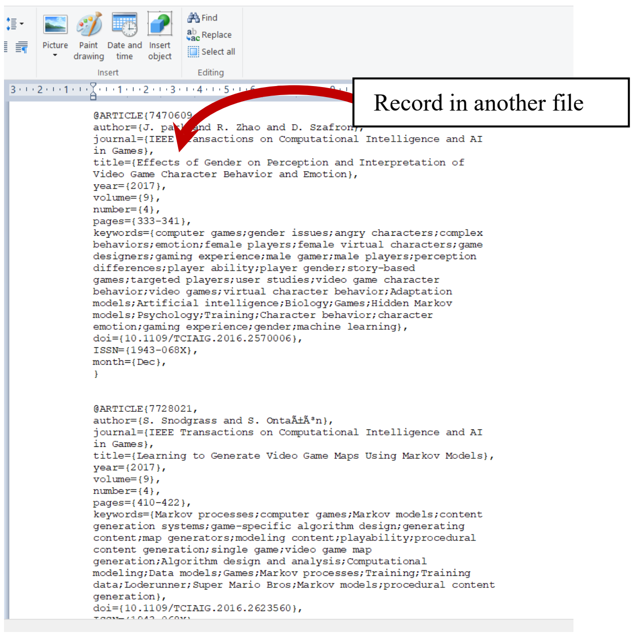
The fine details of what you need to do and how your AuthorBibCreator should work are given in the items below:

1. For the purpose of this assignment, and to provide little simplifications, the following should be assumed in relation to the input files and the articles (records) in these input files:
    a. Each file may have zero (i.e. a file may completely be empty), or more articles; the number is unknown before processing, and your code must assume that;
    b. An article starts with @ARTICLE followed by the body of the article (between "{" and "}"). It is assumed that all articles have enclosing bodies;

c. Inside each of these articles, there exists few fields; i.e. author, volume, year, etc. These fields are assumed to always start with the field name, followed by an "=" sign and a "{" character. For instance: pages={ , month= , etc.
d. It also assumed that each of the bodies of these fields has a closing character for its body. In specific, it is assumed that each of these fields end with "},";
e. For simplicity, it is also assumed that there is a doi field, which maps to https://doi.org/
f. The order of the article is NOT important; i.e., it is okay to have any of these fields above or below other fields;
g. It is also assumed that empty lines can be there within the body of the articles, as well as between different articles;
h. <u>HOWEVER</u> (read carefully), any of the input files may have zero or multiple records for the same author, or other authors with the same partial name (i.e. family name). <u>A search for such name should detects ALL of these records</u>.



```
@ARTICLE{

8247289,
author={J. Park and J. N. James and Q. Li and Y. Xu and W.
Huang},
journal={IEEE Transactions on Vehicular Technology},
title={Optimal DASH-Multicasting over LTE},
year={2018},
volume={PP},
number={99},
pages={15-27},
keywords={Forward error correction;Long Term
Evolution;Maintenance engineering;Multicast
communication;Resource management;Static VAr
compensators;Streaming media;DASH;LTE;convex
optimization;eMBMS;multicasting},
doi={10.1109/TVT.2018.2789899},
ISSN={0018-9545},
month={January},
}


@ARTICLE{

2380090,
author={T. Jackson and A. H. Peterson and N. Wang},
journal={IEEE Transactions on Computer Science},
title={Detecting Security Vulnerabilities in Binary Code},
year={2017},
volume={QQ},
number={85},
pages={1-12},
keywords={Security attacks;Binary code processing;Security error
detection;Deep machine learning;Static analysis},
doi={14.2408/TCS.2017.4746889},
ISSN={0018-9545},
month={May},
}


@ARTICLE{

7628761,
author={S. L. Park and K. Tomson},
journal={IEEE Transactions on Power Systems},
title={Simulating Power Systems for the Detection of Remote
```

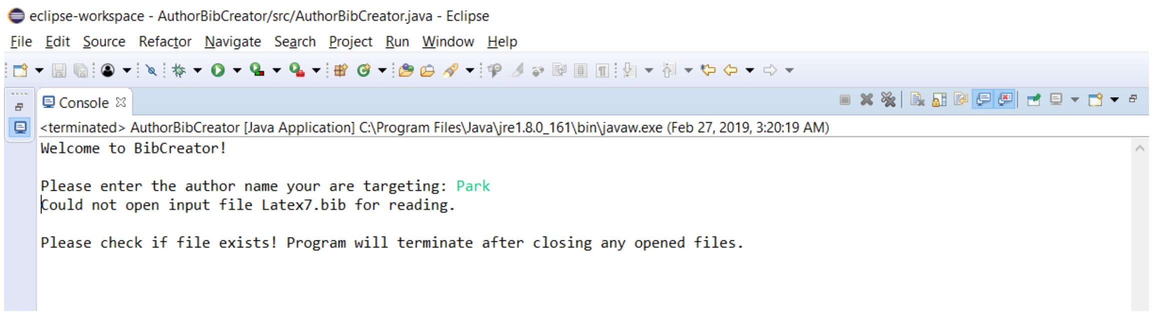


```
@ARTICLE{7470609
author={J. pa[illegible] and R. Zhao and D. Szafron},
journal={IEEE [illegible]ansactions on Computational Intelligence and AI
in Games},
title={Effects of Gender on Perception and Interpretation of
Video Game Character Behavior and Emotion},
year={2017},
volume={9},
number={4},
pages={333-341},
keywords={computer games;gender issues;angry characters;complex
behaviors;emotion;female players;female virtual characters;game
designers;gaming experience;male gamer;male players;perception
differences;player ability;player gender;story-based
games;targeted players;user studies;video game character
behavior;video games;virtual character behavior;Adaptation
models;Artificial intelligence;Biology;Games;Hidden Markov
models;Psychology;Training;Character behavior;character
emotion;gaming experience;gender;machine learning},
doi={10.1109/TCIAIG.2016.2570006},
ISSN={1943-068X},
month={Dec},
}


@ARTICLE{7728021,
author={S. Snodgrass and S. OntaÃ±Ã³n},
journal={IEEE Transactions on Computational Intelligence and AI
in Games},
title={Learning to Generate Video Game Maps Using Markov Models},
year={2017},
volume={9},
number={4},
pages={410-422},
keywords={Markov processes;computer games;Markov models;content
generation systems;game-specific algorithm design;generating
content;map generators;modeling content;playability;procedural
content generation;single game;video game map
generation;Algorithm design and analysis;Computational
modeling;Data models;Games;Markov processes;Training;Training
data;Loderunner;Super Mario Bros;Markov models;procedural content
generation},
doi={10.1109/TCIAIG.2016.2623560},
```

Figure 1. Example of Input Files
(Showing multiple records for authors with name: Park)

2. Write an exception class called **FileExistsException**. The class should have sufficient constructors that allow:
    a. A default message "*Exception: There is already an existing file for that author. File will be renamed as BU, and older BU files will be deleted!*"; and
    b. The passing of any different exception message if desired. This is actually the constructor that you will be using throughout the assignment (see Figure 11 below).

3. In the main() method of the AuthorBibCreator class, require the user to enter an author name to search for, then attempt to open all 10 input files (Latex1.bib to Latex10.bib) for reading. You need to use the **Scanner** class for the reading these files. If "any" of these files does not exist, the program must display an error message indicating "*Could not open input file xxxxx for reading. Please check if file exists! Program will terminate after closing any opened files.*", and then exits. You **<u>MUST</u>** however, close all opened files before exiting the program. For example, if *Latex7.bib* does not exist, then the following image, Figure 2, shows the behavior of the program.


eclipse-workspace - AuthorBibCreator/src/AuthorBibCreator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Console

```
<terminated> AuthorBibCreator [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Feb 27, 2019, 3:20:19 AM)
Welcome to BibCreator!

Please enter the author name your are targeting: Park
Could not open input file Latex7.bib for reading.

Please check if file exists! Program will terminate after closing any opened files.
```

Figure 2. Example of Program Termination – One of the Input Files does not Exist

4. If all 10 input files can successfully be opened, the program must:

    - Check if any files already exists will the names: *author-name*-IEEE.json, *author-name*-ACM.json, and *author-name*-NJ.json, where *author-name* indicates the name being searched.
    - If any of these files is found, the program must throws **FileExistsException** exception. The handling of this exception will be: i) first find if any other files with the names *author-name*-IEEE-BU.json, *author-name*-ACM-BU.json, and *author-name*-NJ-BU.json already exist, and if so, delete these files. ii) Following that, the existing *author-name*-IEEE.json, *author-name*-ACM.json, and *author-name*-NJ.json must be renamed to *author-name*-IEEE-BU.json, *author-name*-ACM-BU.json, and *author-name*-NJ-BU.json.
    - At this point, you are finally ready to create the 3 output files (*author-name*-IEEE.json, *author-name*-ACM.json, and *author-name*-NJ.json,). You need to use **PrintWriter** to open these output files. If "any" of these output files cannot be created, then you must:
        i. Display a message to the user indicating which file could not be opened/created;
        ii. Delete all other created output files (if any). That is, if you cannot create all three output files, then you must clean the directory by deleting all other created files;
        iii. Close all opened input files; then exist the program.

If you reach this step, then finally all 10 input files have been opened successfully and all 3 output files are created successfully (however, these 3 files are surely empty at this point!). The tool is finally ready to start performing what it is supposed to do.

5. Write a method (you should take advantage of static methods throughout the entire assignment!) called **processBibFiles**. This method will represent the core engine for processing the input files and creating the output ones. You can pass any needed parameters to this method, and the method

may return any needed information. This method however must NOT declare any exceptions. In other words, all needed handling of any exceptions that may occur within this method, must be handled by the method. In specific:

a. The method should work on the already opened files;
b. The method must process each of these files to detects all articles that belong to any user with the name being searched;
c. For all found articles, the method must create the proper records in each of the 3 formats (IEEE, ACM and NJ) and store the records in the correct output file;
d. Once all input files have been processed, and the records are stored in the output files, the tool terminates.

For instance, let us assume that the given input files (**these files can be any files and they are not restricted to the ones provided with the assignment; in fact, the marker will execute your assignment with different files; so your code must work correctly for any given files. You are also allowed to modify the names of these files or add additional articles to test your program**) have records as shown for authors with the name `Park` as shown in Figure 1 above. Before any processing is done, here is how the directory contents would look like:

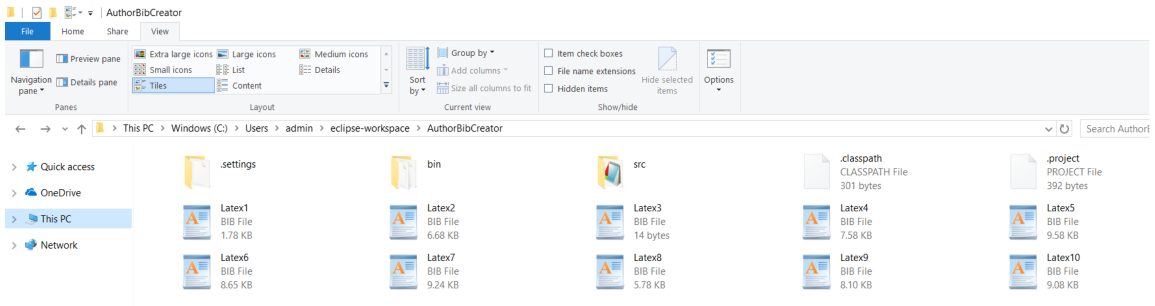
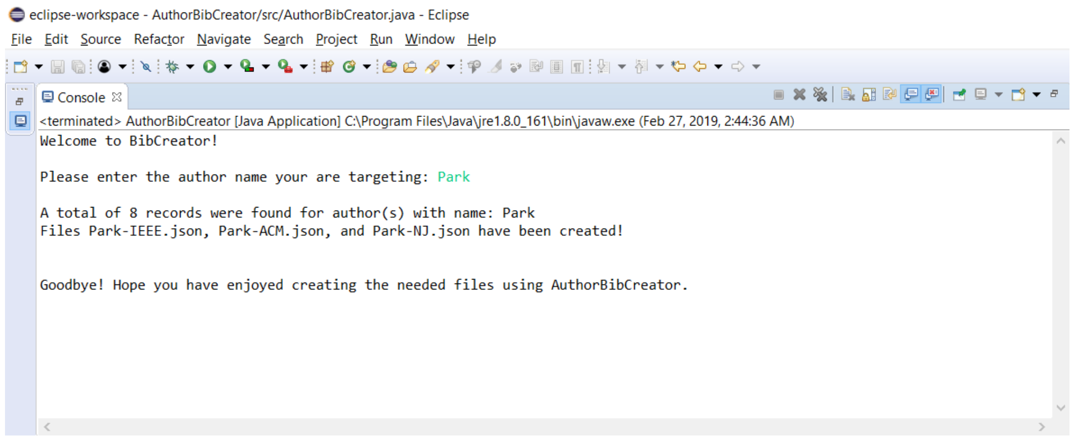


Figure 3. Contents of Directory Before Any Processing Take Place

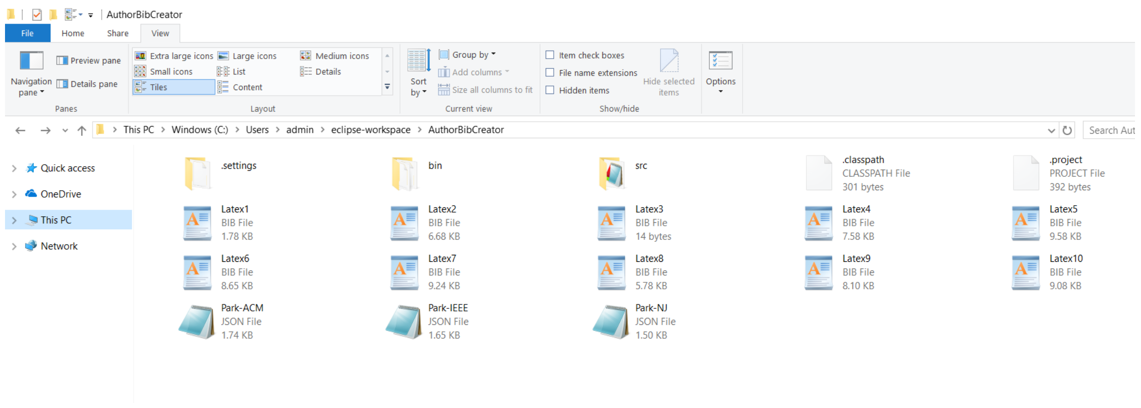
After the user enters the author name to search for, the program output would look like:


eclipse-workspace - AuthorBibCreator/src/AuthorBibCreator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Console

```
<terminated> AuthorBibCreator [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Feb 27, 2019, 2:44:36 AM)
Welcome to BibCreator!

Please enter the author name your are targeting: Park

A total of 8 records were found for author(s) with name: Park
Files Park-IEEE.json, Park-ACM.json, and Park-NJ.json have been created!


Goodbye! Hope you have enjoyed creating the needed files using AuthorBibCreator.
```

Figure 4. Program Output – First-time Search for Author `Park`

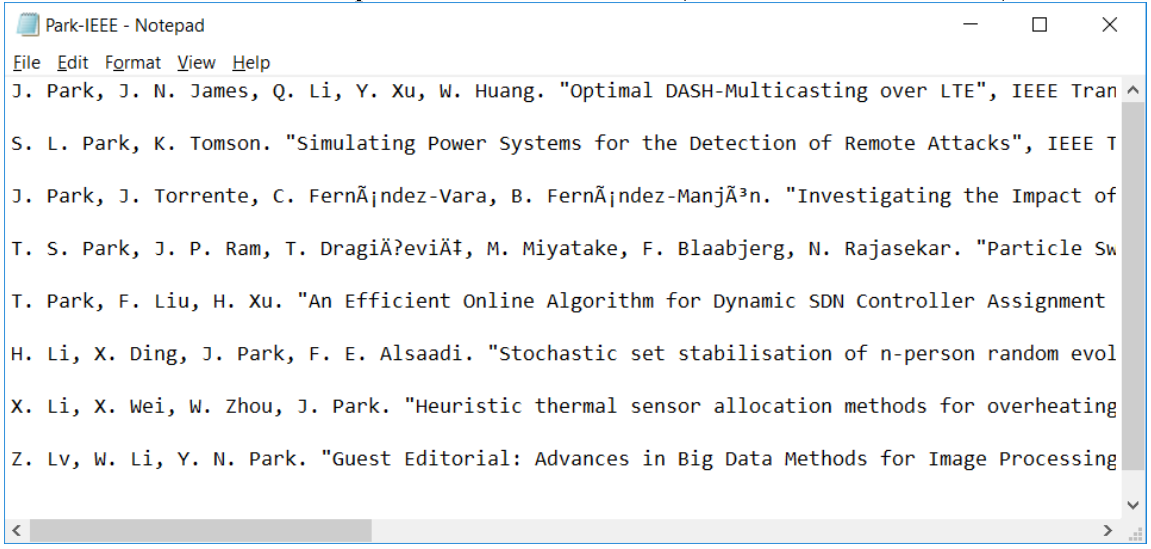
The contents of the directory after this execution would look like:



```
This PC > Windows (C:) > Users > admin > eclipse-workspace > AuthorBibCreator

.settings
bin
src
.classpath   CLASSPATH File   301 bytes
.project     PROJECT File     392 bytes
Latex1       BIB File         1.78 KB
Latex2       BIB File         6.68 KB
Latex3       BIB File         14 bytes
Latex4       BIB File         7.58 KB
Latex5       BIB File         9.58 KB
Latex6       BIB File         8.65 KB
Latex7       BIB File         9.24 KB
Latex8       BIB File         5.78 KB
Latex9       BIB File         8.10 KB
Latex10      BIB File         9.08 KB
Park-ACM     JSON File        1.74 KB
Park-IEEE    JSON File        1.65 KB
Park-NJ      JSON File        1.50 KB
```

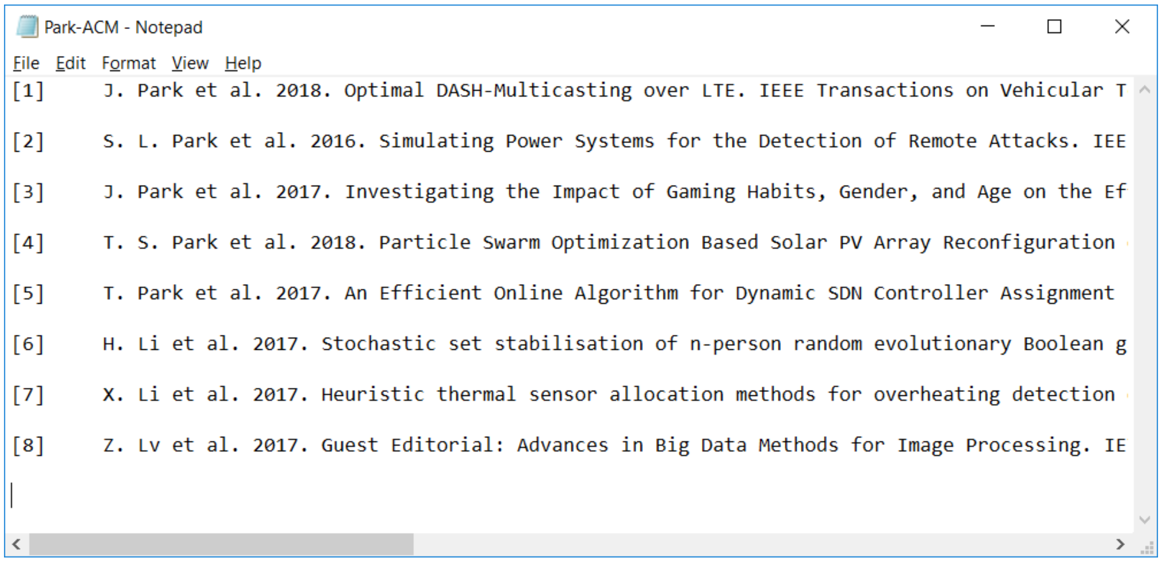
Figure 5. Directory Contents After First Search

Here is how the created output files would look like (notice the case of ACM!):



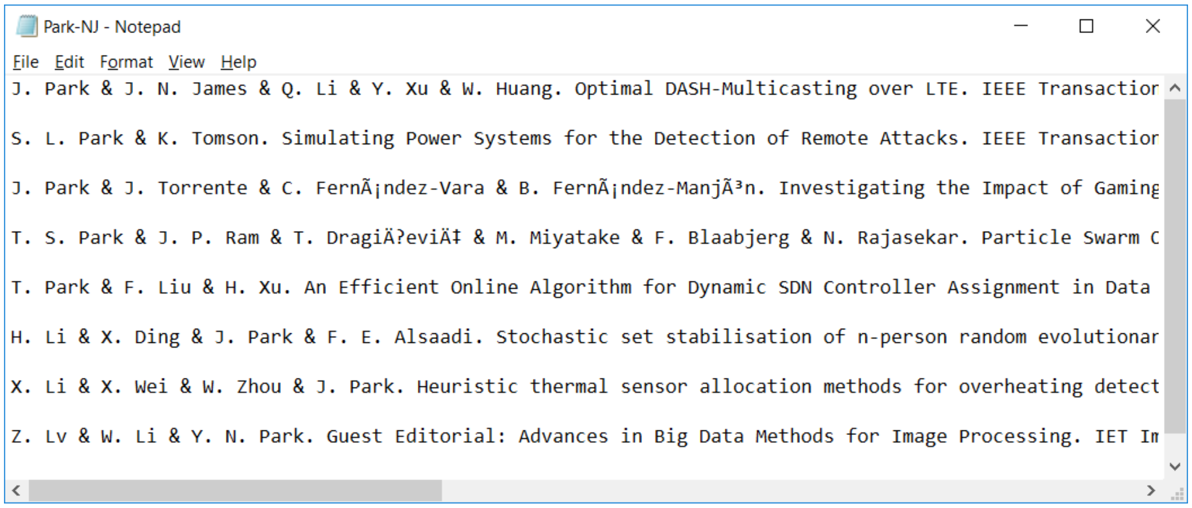
```
Park-IEEE - Notepad

J. Park, J. N. James, Q. Li, Y. Xu, W. Huang. "Optimal DASH-Multicasting over LTE", IEEE Tran

S. L. Park, K. Tomson. "Simulating Power Systems for the Detection of Remote Attacks", IEEE T

J. Park, J. Torrente, C. FernÃ¡ndez-Vara, B. FernÃ¡ndez-ManjÃ³n. "Investigating the Impact of

T. S. Park, J. P. Ram, T. DragiÄ?eviÄ‡, M. Miyatake, F. Blaabjerg, N. Rajasekar. "Particle Sw

T. Park, F. Liu, H. Xu. "An Efficient Online Algorithm for Dynamic SDN Controller Assignment

H. Li, X. Ding, J. Park, F. E. Alsaadi. "Stochastic set stabilisation of n-person random evol

X. Li, X. Wei, W. Zhou, J. Park. "Heuristic thermal sensor allocation methods for overheating

Z. Lv, W. Li, Y. N. Park. "Guest Editorial: Advances in Big Data Methods for Image Processing
```

Figure 8. Partial View of the Created IEEE File for Author Name: Park



```
Park-ACM - Notepad

[1]     J. Park et al. 2018. Optimal DASH-Multicasting over LTE. IEEE Transactions on Vehicular T

[2]     S. L. Park et al. 2016. Simulating Power Systems for the Detection of Remote Attacks. IEE

[3]     J. Park et al. 2017. Investigating the Impact of Gaming Habits, Gender, and Age on the Ef

[4]     T. S. Park et al. 2018. Particle Swarm Optimization Based Solar PV Array Reconfiguration

[5]     T. Park et al. 2017. An Efficient Online Algorithm for Dynamic SDN Controller Assignment

[6]     H. Li et al. 2017. Stochastic set stabilisation of n-person random evolutionary Boolean g

[7]     X. Li et al. 2017. Heuristic thermal sensor allocation methods for overheating detection

[8]     Z. Lv et al. 2017. Guest Editorial: Advances in Big Data Methods for Image Processing. IE
```

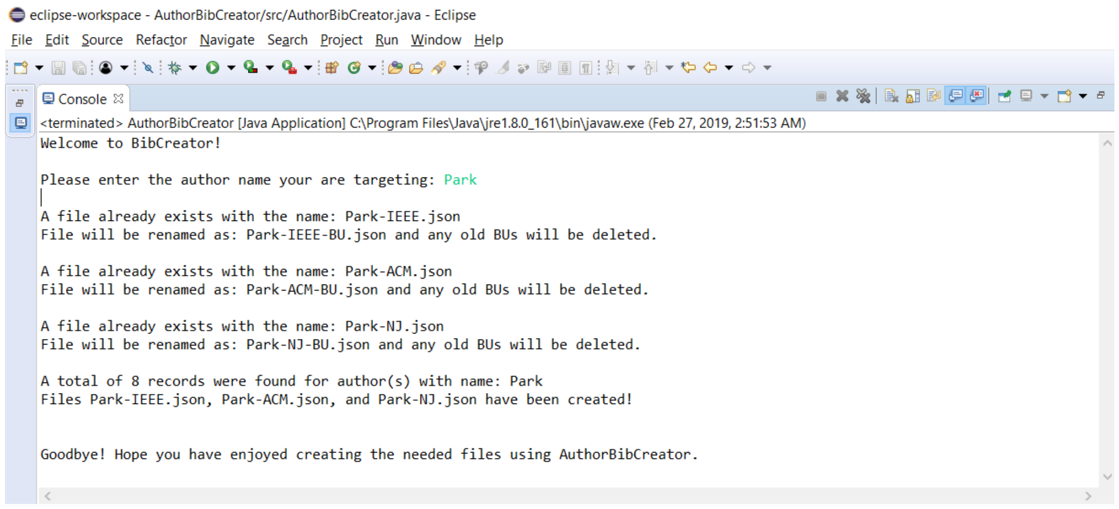
Figure 9. Partial View of the Created ACM File for Author Name: Park



```
J. Park & J. N. James & Q. Li & Y. Xu & W. Huang. Optimal DASH-Multicasting over LTE. IEEE Transaction

S. L. Park & K. Tomson. Simulating Power Systems for the Detection of Remote Attacks. IEEE Transaction

J. Park & J. Torrente & C. FernÃ¡ndez-Vara & B. FernÃ¡ndez-ManjÃ³n. Investigating the Impact of Gaming

T. S. Park & J. P. Ram & T. DragiÄ?eviÄ‡ & M. Miyatake & F. Blaabjerg & N. Rajasekar. Particle Swarm C

T. Park & F. Liu & H. Xu. An Efficient Online Algorithm for Dynamic SDN Controller Assignment in Data

H. Li & X. Ding & J. Park & F. E. Alsaadi. Stochastic set stabilisation of n-person random evolutionar

X. Li & X. Wei & W. Zhou & J. Park. Heuristic thermal sensor allocation methods for overheating detect

Z. Lv & W. Li & Y. N. Park. Guest Editorial: Advances in Big Data Methods for Image Processing. IET Im
```

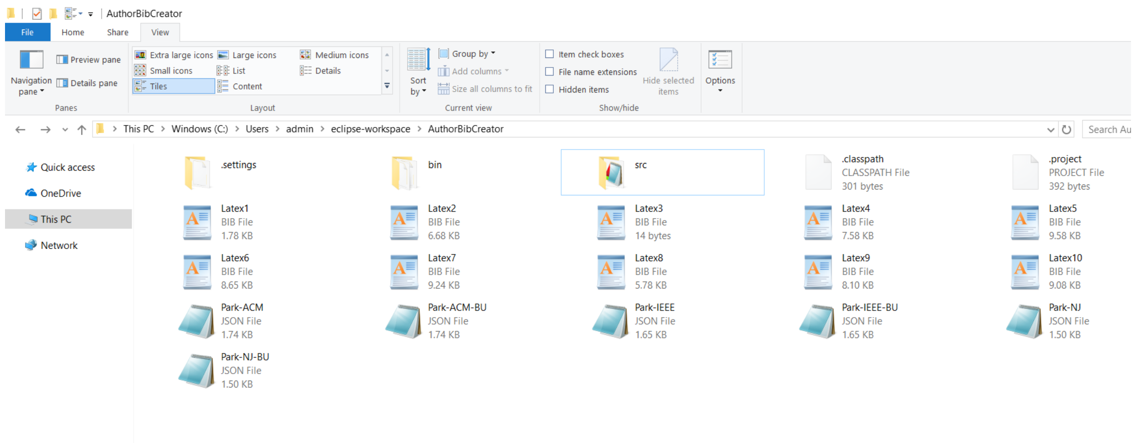
Figure 10. Partial View of the Created NJ File for Author Name: Park

6. Now, another search for author Park, would result in the program output and the contents of the directory afterwards looking as shown in Figure 11 and Figure 12:


eclipse-workspace - AuthorBibCreator/src/AuthorBibCreator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Console

```
<terminated> AuthorBibCreator [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Feb 27, 2019, 2:51:53 AM)
Welcome to BibCreator!

Please enter the author name your are targeting: Park

A file already exists with the name: Park-IEEE.json
File will be renamed as: Park-IEEE-BU.json and any old BUs will be deleted.

A file already exists with the name: Park-ACM.json
File will be renamed as: Park-ACM-BU.json and any old BUs will be deleted.

A file already exists with the name: Park-NJ.json
File will be renamed as: Park-NJ-BU.json and any old BUs will be deleted.

A total of 8 records were found for author(s) with name: Park
Files Park-IEEE.json, Park-ACM.json, and Park-NJ.json have been created!


Goodbye! Hope you have enjoyed creating the needed files using AuthorBibCreator.
```

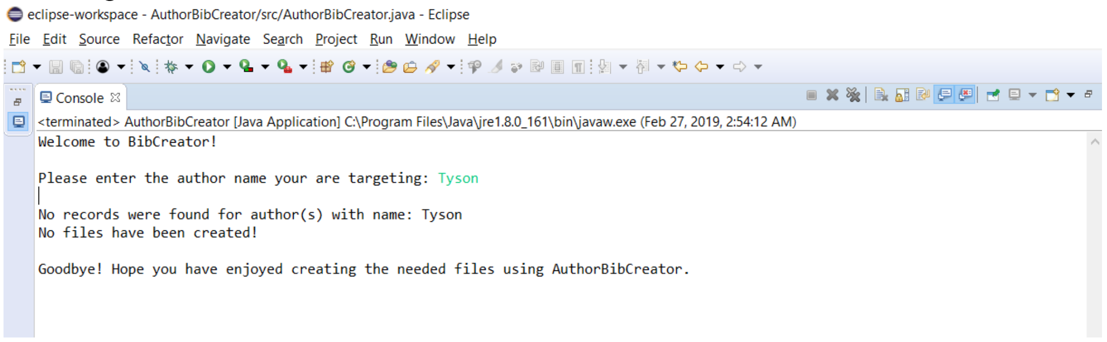
Figure 11. Program Output – Second/Following Search(s) for Author Park



```
.settings            bin                  src                  .classpath           .project
                                                               CLASSPATH File       PROJECT File
                                                               301 bytes            392 bytes
Latex1               Latex2               Latex3               Latex4               Latex5
BIB File             BIB File             BIB File             BIB File             BIB File
1.78 KB              6.68 KB              14 bytes             7.58 KB              9.58 KB
Latex6               Latex7               Latex8               Latex9               Latex10
BIB File             BIB File             BIB File             BIB File             BIB File
8.65 KB              9.24 KB              5.78 KB              8.10 KB              9.08 KB
Park-ACM             Park-ACM-BU          Park-IEEE            Park-IEEE-BU         Park-NJ
JSON File            JSON File            JSON File            JSON File            JSON File
1.74 KB              1.74 KB              1.65 KB              1.65 KB              1.50 KB
Park-NJ-BU
JSON File
1.50 KB
```

Figure 12. Directory contents after Second/Following Search(s)

7. A search for an author name where no records exist, will result in the program displaying the following:


eclipse-workspace - AuthorBibCreator/src/AuthorBibCreator.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Console

```
<terminated> AuthorBibCreator [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Feb 27, 2019, 2:54:12 AM)
Welcome to BibCreator!

Please enter the author name your are targeting: Tyson

No records were found for author(s) with name: Tyson
No files have been created!

Goodbye! Hope you have enjoyed creating the needed files using AuthorBibCreator.
```

Figure 13. Program Output –Search for Author Name where No Records Exists

You should notice that the contents of the directory will remain unchanged in that case (no empty files are created (or left behind)). The details are up to you; however, after all processing is done, all files must be closed and any empty (unsuccessfully created) files MUST be deleted.

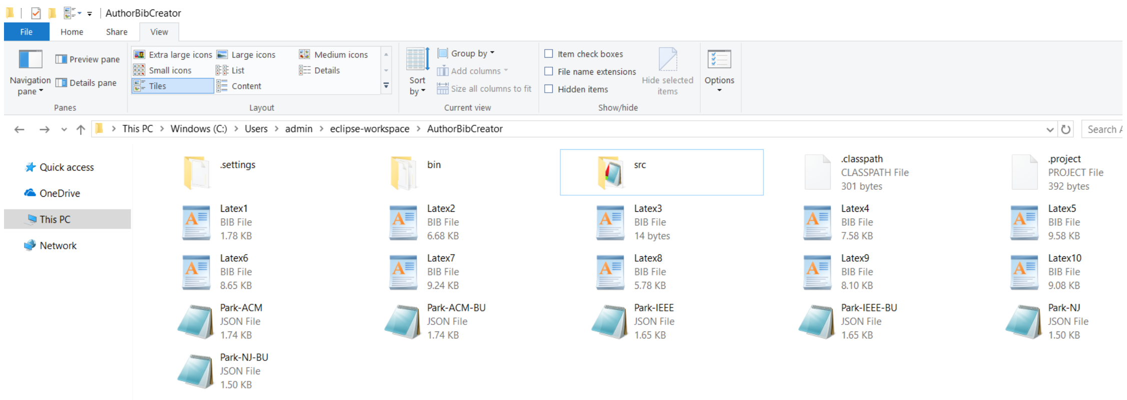


Figure 14. Contents of Current Directory after above Processing Example

8. Finally, here are some general information:
   a. It may assist you greatly if you take advantage of static variables/attributes and static methods throughout the assignment; in fact, it is not necessary to utilize other aspects such as Inheritance, Polymorphism, etc.
   b. You must exactly match the format and look of the expected output files. For instance, use & or et al. for the authors as expected, follow the exact order/format of the contents, use vol. instead of volume when as expected. In other words, a small difference in the expected output will surely result in mark deduction;
   c. For the processing of the authors (breaking the info in the name fields), you may want to use the **StringTokenizer** class; however, other alternatives are allowed;
   d. **<u>You should minimize opening and closing the files as much as possible; a better mark will be given for that</u>**; **<u>Ideally, each file should be opened once and closed once throughout the entire assignment;</u>**
   e. Do not use any external libraries or existing software to produce what is needed; that will directly result in a 0 mark!
   f. Again, your program must work for any input files. The files provided with this assignment are only one possible version, and must not be considered as the general case when writing your code.

General Guidelines When Writing Programs:

- Include the following comments at the top of your source codes

```
// ---------------------------------------------------
// Assignment (include number)
// Question: (include question/part number, if applicable)
// Written by: (include your name and student id)
// ---------------------------------------------------
```

- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program.
- Display a welcome message which includes your name(s).
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy to read format.
- End your program with a closing message so that the user knows that the program has terminated.

JavaDoc Documentation:

Documentation for your program must be written in **JavaDoc**.

In addition, the following information must appear at the top of each file:

```
Name(s) and ID(s)     (include full names and IDs)
COMP249
Assignment #          (include the assignment number)
Due Date              (include the due date for this assignment)
```

Submitting Assignment 3

- For this assignment, you are allowed to work individually, or in a group of a maximum of 2 students (i.e. you and one other student). You and your teammate must however be in the same section of the course. Groups of more than 2 students = zero mark for all members!
- Only electronic submissions will be accepted. Zip together the source codes.
- Students will have to submit their assignments (one copy per group) using the Moodle/EAS system (please check for your section submission). Assignments must be submitted in the right DropBox/folder of the assignments. **Assignments uploaded to an incorrect DropBox/folder will not be marked and result in a zero mark. No resubmissions will be allowed. Your instructor will indicate whether you should upload your assignment to EAS (**https://fis.encs.concordia.ca/eas/**) or to Moodle.**
- Naming convention for zip file: Create one zip file, containing all source files and produced documentations for your assignment using the following naming convention:

  The zip file should be called *a#_StudentName_StudentID*, where # is the number of the assignment and *StudentName/StudentID* is your name and ID number respectively. Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. For example, for the first assignment, student 12345678 would submit a zip file named like: *a1_Mike-Simon_123456.zip.* if working in a group, the name should look like: *a1_Mike-Simon_12345678-AND-Linda-Jackson_98765432.zip.*
- Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded and all others will be disregarded.
- If working in a team, only one of the members can upload the assignment. Do **NOT** upload the file for each of the members!

⇨ Important: Following your submission, a demo is required (please refer to the courser outline for full details). The marker will inform you about the demo times. Please notice that failing to demo your assignment will result in zero mark regardless of your submission.

Evaluation Criteria for Assignment 3 (10 points)

Total	**10 pts**
JavaDoc documentations	1 pt
Item # 2, Item # 3	1 pt
Item # 4	1 pt
Item # 5 & Item # 6	4 pts
Item # 7	1 pt
General Quality & Correctness of the Assignment	2 pts