Overview:

This project fell under the topic of data processing and analysis, required utilising Java to extract, filter, and analyse JSON data regarding Nobel laureates. To find laureates by category and year, find the most frequently used terms in laureate citations while removing stop words, and ensure accurate JSON parsing and output formatting, we had to develop a command-line application. All the essential features are implemented effectively by my solution, which closely follows the output format specifications. I have carried out extensive testing, verifying accuracy in a variety of input settings. Additional optimisations, including improving word frequency analysis or increasing the effectiveness of large-scale data processing, have also been implemented to make my program as efficient and effective as possible.

Code Design:

The main method:

This is the main method responsible for orchestrating the entire process of reading, filtering, and analysing JSON data related to Nobel laureates. The first thing it does is it handles command-line arguments and checks if the number of arguments is less than two. If so, it displays a usage message. Next, it stores the arguments as the file paths and validates the file paths using the validateFilePath() method. A LinkedHashMap has been used because it is used to maintain the insertion order of filters, ensuring predictable behaviour when processing data. This is crucial when multiple filters are applied, preserving the logical sequence for processing. It then loads the JSON data and stopwords. The program then extracts motivation texts and performs a word frequency analysis to identify the most common words, excluding stopwords. Lastly it presents the detailed filtered entries, and the top 10 frequent words based on the analysis. The whole method is placed in a try-catch block for the purpose of error handling for invalid arguments, JSON parsing issues, and I/O errors.

The LoadJSON method:

The purpose of this method is to load and parse the JSON file into a list of JsonObject entries. I have chosen a list as the data structure as it maintains the order of entries as they appear in the JSON file. The JSONReader used is a part of the javax.json package and it is used to read JSON data in a streaming fashion. The method uses a JsonReader from the javax.json package, which is designed for reading JSON data in a streaming manner.[1] The Json.createReader() method creates a JsonReader instance from the provided file input stream, and reader.readObject() reads the entire JSON content, converting it into a JsonObject.[2] jsonObject.getJsonArray("prizes")[3] extracts the JSON array associated with the "prizes" key. The method then checks if the "prizes" field exists and is not null. If the field is missing or null,

[1] https://docs.oracle.com/javaee/7/api/javax/json/JsonReader.html
[2] https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html
[3] https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html

it throws an IllegalArgumentException with a descriptive error message to ensure data integrity. Each element of this array represents an individual prize, which then the method checks if the element is a JSON object before adding it to the list.

The loadStopwards method:

This method loads stopwords from the specified file into a Set for efficient lookup. A set has been chosen because firstly, the time complexity of a HashSet is O(1), allowing for constant word lookup regardless of input size.[4] Additionally, a Set ensures that no duplicate words are stored which is ideal for stopwords where uniqueness is important. The method checks if the file is empty or not to throw the IllegalArgumentException. A try block gets put to read each line from the file and it gets added to the set of strings which store the stopwords.

The filterEntries method:

The purpose of this method is to filter the JSON entries based on the given key-value pairs. It Iterates through each entry, checks for matching filter conditions, and adds valid entries to the result list. The reason behind the use of the map is for quick retrieval of filter values based on keys, ensuring efficient filtering and for the list is that it maintains the order of filtered entries as mentioned previously. The reason why the keys are being checked before filtering is that the program avoids NullPointerException when a key doesn't exist in a JsonObject. The big for loop iterates through all the Json Objects in the list of entries, iterates over each key value in the filters map, checks if the json object has a filter key, retrieves the value associated with the key, converts it to lowercase for case-insensitive comparison, and compares it to the filterValue. If it doesn't match, match is set to false, and the loop breaks—there's no need to check other filters for this entry. If match remains true after checking all filters, the entry satisfies all filter conditions and is added to the filtered list.

The printEntryDetails method:

The purpose of this method is that it prints formatted details of each Nobel prize entry, including year, category, and laureates' information and it handles cases when there are details missing. I have used the getString() methods with defaults like “N/A” to handle cases with missing details. Motivation is usually provided with double quotes, so the method stripQuotes() has been used to ensure a clean output by removing unnecessary quotation. The “\t” has been used for tabbing and formatting the outputs. There has been conditional checking throughout to ensure only existing data gets printed out.

[4] https://medium.com/@chakravartyutkarsh/understanding-why-hashset-provides-o-1-search-time-complexity-15cee2f96cec#:~:text=One%20of%20the%20most%20efficient,an%20element%20is%20incredibly%20fast

The getMotivationText method:

By determining whether "motivation" or "overallMotivation" fields are present, this method extracts and concatenates motivation texts from laureates and overall motivations, then merges the text. A StringBuilder has been used build the final motivation text by appending multiple strings. Firstly, there is a check to see if Entry contains Laureates, then the array of the laureates, then it loops through each laureate and checks if the laureate has a "motivation" key. If yes, it uses that; otherwise, it falls back to "overallMotivation" and each motivation is appended to the motivationText with a space. The stripQuotes() method removes the quotation marks. The second case handles where there is motivation, but it does not present under the laureates key, using the same logic as the other case, the StringBuilder appends.

The stripQuotes method:

The purpose of this method is to remove surrounding quotes from a string where I learned about extracting substrings from strings.[5] The method starts by trimming the string to remove any leading or trailing whitespace using trim(). A conditional check is then applied to see if the string both starts and ends with quotation marks ("), and whether the string length is greater than 2 to prevent trimming an empty string. If these conditions are met, the method returns the substring starting from index 1 and ending at text.length() - 1, removing the surrounding quotes. If the conditions are not met (i.e., the string doesn't have surrounding quotes or is too short), the original string is returned unchanged.

The countWords method:

The purpose of this method is to process the text to count word occurrences, excluding stopwords and single-character words. A map has been used to store word counts, with the word as the key and the count as the value. This allows efficient updates and retrieval of word frequencies. To clean up the text, all the non-alphanumeric characters have been removed and replaced with " ". The split() method splits the text by whitespace characters. Lastly the put()[6] method has been implemented to update the count of the word filtering out empty words, words with length being 1 and words mentioned in the stopwords file.

[5] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html
[6] https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

The printTopwords Method:

This method is designed to print the top ‘n’ most frequent words from a word count map in a formatted manner. The process starts by converting the map entries into a list, as sorting is more straightforward with lists. To arrange the words by frequency, the method uses a selection sort algorithm. It was preferred to use selection sort over bubble sort as Selection short has a the time complexity of O(n2) while the time complexity of Bubble sort is O(n)[7] which means selection sort would be a more efficient sorting algorithm. It sets the first value as the highest value and iteratively compares all entries, ensuring that words with higher frequencies are positioned first. If two words share the same frequency, they are sorted in ascending alphabetical order using the compareTo() method. The sorting ensures that the final output reflects the correct ranking of word frequencies. After sorting, the method prints the top ‘n’ entries, formatting (by tabbing) each line with the word's frequency followed by the word itself.

The validateFilePath method:

This method ensures that the provided file path is valid and accessible before any file operations are performed. It creates a File object from the given filePath and first checks if the file exists using file.exists(). If the file is missing, it throws an IllegalArgumentException with an error message. Next, it verifies if the file is readable using file.canRead(), ensuring the program can access the file's contents. If the file isn't readable, it throws another IllegalArgumentException. This validation step is crucial for preventing runtime errors and ensuring smooth execution when handling file input.

[7] https://testbook.com/key-differences/difference-between-bubble-sort-and-selection-sort#:~:text=The%20time%20complexity%20of%20Bubble,both%20best%20and%20worst%20cases.&text=Bubble%20sort%20is%20generally%20less,efficient%20compared%20to%20bubble%20sort.

Testing:

All the tests were written in the file CS1003P2MyTests.java. All the tests have passed. For the tests that reuire, I created new json and stopwords files using FileWriter[8] and I use the delete() method when I have tested successfully. I use the FileWriter to write the json files.

What is being test	Name of the test method	Preconditions	Expected outcome	Actual Outcome	Evidence
Performing the provided tests	Running ./test.sh	All Preconditions were provided in the bash script file	All tests passing	All tests pass	Screenshot attached in Appendix A1
Testing with no filters	testNoFilters()	Valid JSON file and stopwords file provided. No filters specified.	Program processes all entries and prints motivation word counts for the entire file.	Program processed entries and printed word counts correctly.	Screenshot attached in Appendix A2
Results with multiple filters	testMultipleFilters()	Valid JSON file and stopwords file provided. Filters (e.g., year, category) specified.	Program filters entries based on the provided filters and prints motivation word counts for the filtered entries.	Program filtered entries and printed word counts correctly.	Screenshot attached in Appendix A3
Invalid JSON file path	testInvalidJSONFilePath()	Invalid JSON file path provided. Stopwords file is valid.	Program throws IllegalArgumentException with an error message.	Program threw an error: Error: File does not exist: invalid.json	Screenshot attached in Appendix A4
Invalid stopwords file path	testInvalidStopwordsFilePath()	Valid JSON file provided. Invalid stopwords file path provided.	Program throws IllegalArgumentException with an error message.	Program threw an error: Error: File does not exist: invalid.txt.	Screenshot attached in Appendix A5
Invalid filter key	testInvalidFilterKey()	Valid JSON file and stopwords file provided. Invalid filter key specified.	Program ignores the invalid filter and processes entries without filtering.	Program processed entries without filtering but did not print any word counts.	Screenshot attached in Appendix A6
Empty JSON File	testEmptyJSONFile()	Empty JSON file provided. Stopwords file is valid.	Program throws IllegalArgumentException with a message indicating the file is empty or invalid.	Program threw an error: Error: JSON file is missing the 'prizes' field: empty.json.	Screenshot attached in Appendix A7

[8] https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html

JSON File with Multiple Laureates and No Motivation	testJSONFileWithMultipleLaureatesAndNoMotivation()	JSON file contains multiple laureates but no motivation fields. Stopwords file is valid.	Program processes entries and skips motivation analysis for laureates without motivation.	Program processed entries but did not print any word counts (expected behavior).	Screenshot attached in Appendix A8
JSON File with Only OverallMotivation	testJSONFileWithOnlyOverallMotivation()	JSON file contains only overallMotivation field. Stopwords file is valid.	Program processes overallMotivation and prints word counts.	Program processed overallMotivation and printed word counts correctly.	Screenshot attached in Appendix A9
JSON File with Only Motivation	testJSONFileWithOnlyMotivation()	JSON file contains only motivation field for laureates. Stopwords file is valid.	Program processes motivation fields and prints word counts.	Program processed motivation fields and printed word counts correctly.	Screenshot attached in Appendix A10
JSON File with No Laureates	testJSONFileWithNoLaureates()	JSON file contains no laureates. Stopwords file is valid.	Program processes entries without laureates and skips motivation analysis.	Program processed entries but did not print any word counts (expected behavior).	Screenshot attached in Appendix A11
JSON File with Duplicate Entries	testJSONFileWithDuplicateEntries()	JSON file contains duplicate entries. Stopwords file is valid.	Program processes entries and includes duplicates in the output.	Program processed entries and included duplicates in the output.	Screenshot attached in Appendix A12
JSON File with Missing Fields	testJSONFileWithMissingFields()	JSON file is missing required fields (e.g., year, category). Stopwords file is valid.	Program throws IllegalArgumentException or skips entries with missing fields.	Program processed entries but did not print any word counts (expected behavior).	Screenshot attached in Appendix A13
Testing for Motivation Fields with Only Stopwords	testMotivationFieldsWithOnlyStopwords()	JSON file contains motivation fields with only stopwords. Stopwords file is valid.	Program processes entries but skips stopwords in the word count analysis.	Program processed entries but did not print any word counts (expected behavior).	Screenshot attached in Appendix A14
Testing for Empty Stopwords File	testEmptyStopwordsFile()	Valid JSON file provided. Stopwords file is empty.	Program throws IllegalArgumentException with a message indicating the file is empty.	Program threw an error: Error: Stopwords file is empty: empty_stopwords.txt	Screenshot attached in Appendix A15

Evaluation:

All the requirements for the CS1003P2 project have been successfully implemented in my program. Throughout the development process, I gained a deeper understanding of JSON parsing, command-line argument handling, sorting algorithms and file reading and parsing, which significantly enhanced my programming skills. I effectively utilized data structures such as Lists, Sets, and Maps to ensure efficient data handling and processing, with specific choices like using a LinkedHashMap to maintain filter order for predictable behaviour. Moreover, strong error handling has been incorporated to address issues like missing files, incorrect JSON formats, and general I/O errors, ensuring that the program handles unexpected scenarios gracefully. The program also sticks strictly to the command-line interface specifications, ensuring accurate filtering and output formatting. Areas for potential improvement were identified during development. The filtering logic could be enhanced to accommodate more complex queries or partial matches, increasing the program's versatility. Additionally, while the current output formatting is functional, refining it could improve readability. Providing more detailed error messages and clearer feedback for incorrect inputs would further enhance the overall user experience.

Overall, I am satisfied with my submission, as it meets all the specified requirements and demonstrates my solid understanding of semi-structured data processing in Java. While the current solution is efficient and reliable, future versions could introduce advanced features for greater flexibility and user-friendliness.

Conclusion:

I thoroughly enjoyed working on this assignment, particularly the challenge of processing semi-structured JSON data and performing word frequency analysis. The process of handling command-line arguments, filtering data based on dynamic criteria, and ensuring accurate word count analysis was very rewarding after performing a lot of research. One of the more interesting aspects was implementing and learning about different data handling using Java collections like List, Set, and Map, which deepened my understanding of data structures and their applications. I did face some challenges, especially in ensuring that the filtering logic handled all possible scenarios correctly and that the output adhered to the specified format. However, through iterative testing and refinement, I was able to achieve a functional and reliable solution. Given more time, I would like to enhance the program by introducing more advanced filtering options, supporting partial matches.

Overall, this project has been an excellent learning experience, helping me improve my problem-solving skills and deepen my understanding of data processing in Java. I look forward to applying these skills to future projects and further enhancing my approach to complex programming challenges.

References:

[1] https://docs.oracle.com/javaee/7/api/javax/json/JsonReader.html

[2] https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html

[3] https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html

[4] https://medium.com/@chakravartyutkarsh/understanding-why-hashset-provides-o-1-search-time-complexity-15cee2f96cec#:~:text=One%20of%20the%20most%20efficient,an%20element%20is%20incredibly%20fast

[5] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

[6] https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

[7] https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html

[8] https://app.diagrams.net

[9] https://studres.cs.st-andrews.ac.uk/CS1003/

[10] https://testbook.com/key-differences/difference-between-bubble-sort-and-selection-sort#:~:text=The%20time%20complexity%20of%20Bubble,both%20best%20and%20worst%20cases.&text=Bubble%20sort%20is%20generally%20less,efficient%20compared%20to%20bubble%20sort

Appendix:

Appendix A1

```
[redacted] CS1003-P2 % ./test.sh
Note: CS1003P2Test.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Test case: samples/category-chemistry-year-1980.in samples/category-chemistry-year-1980.out
Running command: java CS1003P2 data/prize.json data/stopwords category chemistry year 1980
Passed.

Test case: samples/category-literature.in samples/category-literature.out
Running command: java CS1003P2 data/prize.json data/stopwords category literature
Passed.

Test case: samples/category-peace.in samples/category-peace.out
Running command: java CS1003P2 data/prize.json data/stopwords category peace
Passed.

Test case: samples/year-1980.in samples/year-1980.out
Running command: java CS1003P2 data/prize.json data/stopwords year 1980
Passed.

Test case: samples/year-2000.in samples/year-2000.out
Running command: java CS1003P2 data/prize.json data/stopwords year 2000
Passed.

Test case: samples/year-2020-category-peace.in samples/year-2020-category-peace.out
Running command: java CS1003P2 data/prize.json data/stopwords year 2020 category peace
Passed.
```

Appendix A2

```
year: 2021
category: physics
Number of laureates: 1
    - firstname: Albert
    - surname: Einstein
    - motivation: for his discovery of the photoelectric effect



Most frequent words in the motivation fields:

    1   discovery
    1   effect
    1   for
    1   his
    1   photoelectric
```

Appendix A3

```
Testing with multiple filters...
Filter year = 2021
Filter category = physics

year: 2021
category: physics
Number of laureates: 1
    - firstname: Albert
    - surname: Einstein
    - motivation: for his discovery of the photoelectric effect



Most frequent words in the motivation fields:

    1   discovery
    1   effect
    1   for
    1   his
    1   photoelectric
```

Appendix A4

```
Testing with invalid JSON file path...
Error: File does not exist: invalid.json
```

Appendix A5

```
Testing with invalid stopwords file path...
Error: File does not exist: invalid.txt
```

Appendix A6

```
Testing with invalid filter key...
Filter invalidkey = value


Most frequent words in the motivation fields:
```

Appendix A7

```
Testing with empty JSON file...

Error: JSON file is missing the 'prizes' field: empty.json
```

Appendix A8

```
Testing JSON file with multiple laureates and no motivation...

year: 2021
category: physics
Number of laureates: 2
    - firstname: Albert
    - surname: Einstein
    - motivation:

    - firstname: Marie
    - surname: Curie
    - motivation:



Most frequent words in the motivation fields:
```

Appendix A9

```
Testing JSON file with only overall motivation...

year: 2021
category: physics
Motivation: for their contributions to the understanding of the universe


Most frequent words in the motivation fields:

   1   contributions
   1   for
   1   their
   1   to
   1   understanding
   1   universe
```

Appendix A10

```
Testing JSON file with only motivation...

year: 2021
category: physics
Number of laureates: 1
    - firstname: Albert
    - surname: Einstein
    - motivation: for his discovery of the photoelectric effect



Most frequent words in the motivation fields:

    1   discovery
    1   effect
    1   for
    1   his
    1   photoelectric
```

Appendix A11

```
Testing JSON file with no laureates...

year: 2021
category: physics


Most frequent words in the motivation fields:
```

Appendix A12

```
Testing JSON file with duplicate entries...

year: 2021
category: physics
Number of laureates: 1
    - firstname: Albert
    - surname: Einstein
    - motivation: for his discovery of the photoelectric effect


year: 2021
category: physics
Number of laureates: 1
    - firstname: Albert
    - surname: Einstein
    - motivation: for his discovery of the photoelectric effect



Most frequent words in the motivation fields:

    2   discovery
    2   effect
    2   for
    2   his
    2   photoelectric
```

Appendix A13

```
Testing JSON file with missing fields...

year: 2021
category: N/A


Most frequent words in the motivation fields:
```

Appendix A14

```
Testing motivation fields with only stopwords...

year: 2021
category: physics
Number of laureates: 1
    - firstname: Albert
    - surname: Einstein
    - motivation: the of in



Most frequent words in the motivation fields:
```

Appendix A15

```
Testing with empty stopwords file...

year: 2021
category: physics
Number of laureates: 1
    - firstname: Albert
    - surname: Einstein
    - motivation: for his discovery of the photoelectric effect


Error: Stopwords file is empty: empty_stopwords.txt
```