Overview

In this practical assignment, we were tasked to write a program that performs a word frequency analysis on the data returned from Bored API. Bored API is a simple API that suggests random activities to users, which can be filtered using the number of participants specified by the activities, the type of activity, the price of the activity etc. Apart from making queries to the online Bored API URLs, our program also needs to be able to read data from local files that form a local cache.

My solution achieves all parts of the specification. It successfully implements classes to create the required functionality as stated in the specification.

Code Design

There was a total of five distinct tasks that our program should be able to accomplish:

1. Search for a random activity using a Bored API URL.
2. Search for an activity of a type specified by the user using a Bored API URL.
3. Search for an activity with a specified number of participants using a Bored API URL.
4. Retrieve an activity based on its key, which would be specified by the user.
5. Iterate over all the files given in the cache directory and print the 10 most frequent words that appear the “activity” text field in each file together with the number of occurrences.

Before I could focus on programming each task, I first had to make sure that the user keyed in appropriate command line arguments, which I could then use to obtain the data required. I did this by implementing a class called “ArgsValidation”, which was where most of my error-checking took place. Within this class, I created methods that would check if a valid number of arguments were entered, whether a valid task name was entered, and if the path to the cache directory that the user entered exists. If not, an error message would be printed according to the type of error the user’s inputs caused.

Once all of the user’s arguments have been verified, my main method then decides on which method to call depending on the task specified by the user. I did this by using a switch statement, whose cases were the different possible tasks supported by my program. Upon matching a given task, control of the program will be passed on to another method in a class that provides the functionality to run the task specified by the user. For example, if the user typed in “random” as the task given to the program to achieve, then a method called “getActivity()” in the “Random” class would be called.

For the first three tasks (with reference to the list of tasks above), the program will need to use another class called “HTTPRequest” which sends a HTTP request using a Bored API URL. If the task specified is “random”, then the base URL “http://www.boredapi.com/api/activity” will be used to retrieve a random activity. If the task specified is “type” or “participants”, then my program will append the value specified by the user in their third command line argument to the end of the base URL, which would end up looking like “https://www.boredapi.com/api/activity?type=recreational”. Once my program receives the response from the Bored API, which is a JSON formatted string of text, my program converts this string into a JsonObject object. By taking advantage of the standard naming structure of the JSON strings returned by the Bored API, my program then directly accesses the JSON values using the required keys to output the activity in the format needed by the specification.

For the fourth task, which requires my program to get an activity from the local cache directory using a key specified by the user, I decided to make use of the strict naming convention of the files in the local cache directory, whose names are all formatted as such: "/path/to/cachedir/KEY.json". By taking the path to the cache directory and the value of the key specified by the user, my program constructs the path to the JSON file needed. This path is then used by a JsonParser object to read the file, which then extracts the required data from the JSON contents within the file to print the required formatted string of the activity.

For the fifth task, my program simply iterates through every single file in the cache directory provided by the user. Similar to task four, a JsonParser would read and extract data from the file (in this case, we only need the text from the "activity" field). This string of text would then be formatted split into a list. This list would then be iterated through, and each word would be either stored into a HashMap, or have its count (which is its corresponding value in the HashMap) incremented. To sort the words according to their word count, I implemented another class named "Word", which only has two attributes – a string which stores the word itself, and an integer value for its word count. I also used the Comparator class to create a comparator that would aid me in sorting the words in decreasing order according to their count. I created an ArrayList with elements of type Word and populated it with Word objects constructed using the keys and values of the HashMap (i.e. all words and their corresponding counts that appear across all "activity" text fields). Finally, I sorted the ArrayList using the comparator in the Word class. To obtain the final output, I iterated through the ArrayList from indices one to 10 to print the word and its corresponding count.

Testing

I started testing my program by ensuring that all the automated stacscheck tests passed. As shown in the stacscheck output below, my program satisfied all of the provided tests.

```
Testing CS1003 Assignment #1 - Bored API
- Looking for submission in a directory called 'src': Already in it!
* BUILD TEST - build : pass
* COMPARISON TEST - badargs/0arg/prog-expected.out : pass
* COMPARISON TEST - badargs/1arg/prog-expected.out : pass
* COMPARISON TEST - badargs/1argwrong-cache/prog-expected.out : pass
* COMPARISON TEST - badargs/2arg2-location/prog-expected.out : pass
* COMPARISON TEST - badargs/2args-key/prog-expected.out : pass
* COMPARISON TEST - badargs/2args-participants/prog-expected.out : pass
* COMPARISON TEST - badargs/2args-random-wrong-cache/prog-expected.out : pass
* COMPARISON TEST - badargs/2args-summary-wrong-cache/prog-expected.out : pass
* COMPARISON TEST - badargs/2args-type/prog-expected.out : pass
* COMPARISON TEST - badargs/3args-random/prog-expected.out : pass
* COMPARISON TEST - badargs/3args-summary/prog-expected.out : pass
* COMPARISON TEST - badargs/3args-type-wrong/prog-expected.out : pass
* COMPARISON TEST - wellformed/key-1638604/prog-expected.out : pass
* COMPARISON TEST - wellformed/key-2790297/prog-expected.out : pass
* COMPARISON TEST - wellformed/key-3305912/prog-expected.out : pass
* COMPARISON TEST - wellformed/key-4101229/prog-expected.out : pass
* COMPARISON TEST - wellformed/key-5881028/prog-expected.out : pass
* TEST - wellformed/participants-1/test : pass
* TEST - wellformed/participants-2/test : pass
* TEST - wellformed/participants-3/test : pass
* TEST - wellformed/participants-4/test : pass
* TEST - wellformed/participants-5/test : pass
```

```
* TEST - wellformed/random/test : pass
* COMPARISON TEST - wellformed/summary/prog-expected.out : pass
* TEST - wellformed/type-busywork/test : pass
* TEST - wellformed/type-charity/test : pass
* TEST - wellformed/type-cooking/test : pass
* TEST - wellformed/type-diy/test : pass
* TEST - wellformed/type-education/test : pass
* TEST - wellformed/type-music/test : pass
* TEST - wellformed/type-recreational/test : pass
* TEST - wellformed/type-relaxation/test : pass
* TEST - wellformed/type-social/test : pass
34 out of 34 tests passed
```

Figure 1: All stacscheck tests were successfully passed.

Apart from using stacscheck, I also tested my program using my own inputs to ensure that everything was working as intended. I first tested my program by using an invalid number of arguments to check if the program outputs the correct error message. This test was important there were multiple different types of error messages that could be printed depending on the number of command line arguments given, as well as the task specified.

```
Expected two or three arguments, but got: 4
Usage: java CS1003Bored CACHEDIR MODE [VALUE]
```

Figure 2: Appropriate error message is thrown after attempting to use four command line arguments.

```
Expected a third argument when MODE is set to key
But got 2 arguments
Usage: java CS1003Bored CACHEDIR MODE [VALUE]
```

Figure 3: Appropriate error message is thrown after attempting to use two command line arguments for the “key” task.

```
Expected two arguments when MODE is set to summary
But got 3 arguments
Usage: java CS1003Bored CACHEDIR MODE [VALUE]
```

Figure 4: Appropriate error message is thrown after attempting to use more than two command line arguments for the “summary” task.

Another thing I tested for was if I had passed an invalid cache directory path as a command line argument into the terminal. In this case, my program successfully detected that the path to the directory was invalid, and printed the corresponding error message.

```
Cache directory does not exist: no_such_cachedir
Usage: java CS1003Bored CACHEDIR MODE [VALUE] [VALUE]
```

Figure 5: Invalid cache directory error message is printed when a non-existed cache directory is passed as a command line argument.

Lastly, for the initial tests, I also made sure that my program printed the correct error message if a user keyed in an invalid task.

```
Unexpected value for MODE: wrong_task
Expected one of: random, type, participants, key, summary
Usage: java CS1003Bored CACHEDIR MODE [VALUE]
```

Figure 6: Invalid task error message is printed when a wrong task is passed as a command line argument.

After the initial tests that check the user's command line arguments, the next error that could occur would be if the user's value did not produce a result. For instance, if the user ran the "type" task and specified "swimming" as a value, there would be no possible result. I took advantage of the standard error message returned by the Bored API when an invalid activity was requested to print a suitable message for when this error occurred.

```
Unexpected value for type: swimming
Expected one of: busywork, charity, cooking, diy, education, music, recreational, relaxation, so-
cial
Usage: java CS1003Bored CACHEDIR MODE [VALUE]
```

Figure 7: Correct error message printed when an invalid activity type is specified by the user.

Similarly, I did the same for when an invalid value is entered for the "participants" and "key" task options. Although these were not tested as part of the stacscheck, I decided to implement these tests as well as it did not make sense to have the value validation for the "type" task and not the other tasks. Note that for the "key" task, I checked for the existence of a file with the given key rather than use a Bored API URL to determine if the file exists or not.

```
Unexpected value for participants: 9
Usage: java CS1003Bored CACHEDIR MODE [VALUE]
```

Figure 8: Correct error message printed when an invalid number of activity participants is specified by the user.

```
Unexpected value for key: 1000001
Usage: java CS1003Bored CACHEDIR MODE [VALUE]
```

Figure 9: Correct error message printed when an invalid key is specified by the user.

After ensuring that all the potential errors were handled, I finally ended my testing by running each task with valid inputs to ensure that the specified activities were printed in the desired format. For example, I checked that the type of activity I specified matched the activity displayed by the program.

```
Found an activity of type recreational
Number of participants needed: 1
Description: Play a video game
```

Figure 10: Output from using the "random" task.

```
Found an activity of type diy
Number of participants needed: 1
Description: Learn woodworking
```

Figure 11: Output from using the “type” task with the “diy” value.

```
Found an activity of type social
Number of participants needed: 4
Description: Have a paper airplane contest with some friends
```

Figure 12: Output from using the “participants” task with “4” as the number of participants.

```
Found an activity of type social
Number of participants needed: 4
Description: Invite some friends over for a game night
```

Figure 13: Output from using the “key” task with the key value being “6613428”.

```
Most frequent words in the activity fields:
    124 a
    53  your
    44  to
    32  learn
    31  with
    20  friends
    20  you
    17  go
    16  some
    15  the
```

Figure 14: Output from using the “summary” task.

After making sure that all my constructed tests returned either an appropriate error message or activity output, I was satisfied that my program fulfilled all the tasks detailed by the specification. However, to further confirm that my program was indeed fully tested, I also included JUnit tests as an additional layer of testing. To run these tests, I created another class called “ArgsValidationTest”, which I used to check that my ArgsValidation class was working as intended.

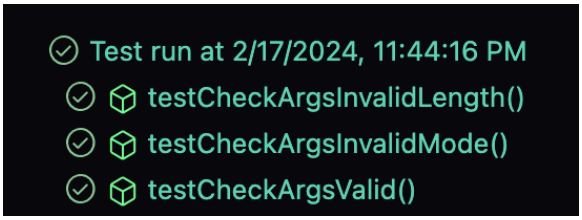

Figure 15: Successful JUnit tests

Evaluation

Overall, my program successfully implements all the requirements in the assignment. I have run my program extensively to ensure that it works as intended and also included comments as a form of documentation within each Java file.

Conclusion

This assignment provided an enjoyable and insightful test of my programming skills. It served as a good consolidation of what I have learnt over the past few weeks. In particular, I was able to use what I had learned about the Java Collections framework, file reading, and semi-structured data to finish my program. I also used managed to use concepts that were covered in the exercises to build methods that could achieve the desired functionalities.

If there was later deadline to this project, and if I were given more freedom, I would like to extend my program to be able to deal with JSON data from other APIs, as I found this portion to be one of the more interesting parts of the assignment.

Overall, I viewed this assignment as a meaningful conclusion for all the concepts that I have learnt over the first few weeks of the CS1003 module, and I'm content with the final solution I came up with.

References

[1] Do a Simple HTTP Request in Java. (2024, February 8). Baeldung.com. https://www.baeldung.com/java-http-request

[2] How to Sort ArrayList using Comparator? (2020, December 14). GeeksforGeeks. https://www.geeksforgeeks.org/how-to-sort-arraylist-using-comparator/

[3] JsonObject (Java(TM) EE 7 Specification APIs). (2015, June). Oracle.com. https://docs.oracle.com/javaee%2F7%2Fapi%2F%2F/javax/json/JsonObject.html

[4] JsonParser (Java(TM) EE 7 Specification APIs). (2015, June). Oracle.com. https://docs.oracle.com/javaee%2F7%2Fapi%2F%2F/javax/json/stream/JsonParser.html