CS1003 – Programming with Data

Assignment #1: Semi-structured data processing

Deadline: 23 February 2024, 9pm
Credits: 30%
MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

This practical involves retrieving semi structured data from an online data source and using basic text processing techniques to perform the specified task. You will need to decompose the problem into a number of methods as appropriate and classes if necessary. You will also need to test your solution carefully and write a report.

This coursework assessment uses stacscheck tests.

Synopsis

The task is to write a Java program to perform a word frequency analysis on data returned from Bored API. Bored API is a simple, fun API designed to suggest random activities to users. The API provides ideas for things to do when someone is feeling bored. These activities can range from simple tasks to more involved projects, and the API offers options that vary by the number of participants, activity type, accessibility, etc.

See the documentation for examples: https://www.boredapi.com/documentation

The responses returned by this API use JSON. Go to the following URL to access a random activity in your web browser: http://www.boredapi.com/api/activity. Try refreshing the page and you should see a different activity each time!

Web APIs and Bored API

Typically, web APIs like the one we use in this coursework are structured to have a base address, followed by a number of parameters. Each parameter is written as a key value pair, separated by an equals sign (key=value). The parameters are separated using an ampersand sign (&), and the parameter list is separated from the base address using a question mark (?). Here is an example for the Bored API:

```
http://www.boredapi.com/api/activity?type=recreational&participants=1
```

The base address in this example is **`http://www.boredapi.com/api/activity`**, the first parameter is called **`type`** and its value is **`"recreational"`**, the second parameter is called **`participants`** and its value is **`"1"`**.

The full API specification for Bored API can be found on its website.

Reading from a local cache

In addition to making queries to the online Bored API URLs, your program will read data from local files that form a local cache of this API (for tasks 4 and 5). The files in the local cache follow a strict naming convention: for example a file called 5881028.json contains the following information in it.

```
{
  "activity": "Learn a new programming language",
  "accessibility": 0.25,
  "price": 0.1,
  "link": "",
  "type": "education",
  "key": "5881028",
  "participants": 1
}
```

The local cache is available as part of the assignment specification in the following directory:
https://studres.cs.st-andrews.ac.uk/CS1003/Coursework/A1/Tests/cachedir
You can open these files in your web browser and explore the structure of the JSON files.

Code organisation

- Create a file called CS1003Bored.java inside a directory called **`src`**.
- You may create more classes for organising your code. These should be stored in the same directory.
- The code will be compiled by running **`javac *.java`** in the **`src`** directory.
- Make sure you follow these naming conventions for stacscheck to work.

Command line arguments

Your program (CS1003Bored) should take 4 command line arguments.

1. Path to the cache directory
2. Task. Accepted values are “random”, "type", "participants", “key”, or “summary”
3. Value. Required for all query types except summary

Your program should validate the command line arguments and produce error messages that match those expected by the stacscheck tests.

Description of tasks

This section describes the tasks needed for this assignment, gives examples and hints. There are written to help you, make sure you read the section in full before you start programming.

Task 1: Retrieve a random activity

This task is run when the first command line argument is “random”.

Use the following URL to retrieve a random activity:

`http://www.boredapi.com/api/activity`

Print information about the activity to standard output. Format the activity as follows:

```
$ java CS1003Bored /path/to/cachedir random

Found a charity activity.
Number of participants needed: 1
Description: Volunteer at your local food pantry
```

Task 2: Search activity by type

This task is run when the first command line argument is "type".

Construct a URL to search for an activity by type:

```
http://www.boredapi.com/api/activity?type=VALUE
```

Print information about the activity to standard output. The following is an example invocation.

```
$ java CS1003Bored /path/to/cachedir type busywork

Found an activity of type busywork
Number of participants needed: 1
Description: Organize a bookshelf
```

Task 3: Search activity by participant count

This task is run when the first command line argument is "participant".

Construct a URL to search for an activity by type:

```
http://www.boredapi.com/api/activity?participants=VALUE
```

Print information about the activity to standard output. The following is an example invocation.

```
$ java CS1003Bored /path/to/cachedir participants 3

Found an activity of type social
Number of participants needed: 3
Description: Have a picnic with some friends
```

Task 4: Retrieve a task by key

This task is run when the first command line argument is "key".

Do not make an online request for this task. Instead, read the data from a local file stored at **{cache-directory}/{key}.json**.

The following is an example invocation:

```
$ java CS1003Bored /path/to/cachedir key 2790297

Found an activity of type education
Number of participants needed: 1
Description: Learn how to whistle with your fingers
```

Task 4: Summary

This task is run when the first command line argument is "summary".

Iterate over all files in the given cache directory. Calculate and print to standard output the following information:

```
$ java CS1003Bored ../../Tests/cachedir summary

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
```

This output is the result of a word frequency analysis on data returned stored in the cache directory. Your program needs to parse each file in this directory and retrieve the "activity" text field from each file. Finally, it will print a histogram of the most frequent 10 words together with the number of occurrences. Keep reading for more some useful hints!

Hints on cleaning textual data

Textual data you extract from the "activity" field needs to be cleaned before counting words in it. There are many ways of cleaning/normalising textual data, we choose a very straightforward definition in this practical.

- Remove all non-alphanumeric characters by replacing them with a space character.
  ```
  String text = …;
  text = text.replaceAll("[^a-zA-Z0-9]", " ");
  ```
- Convert the comment text to lowercase.
  ```
  text = text.toLowerCase()
  ```
- Split text on whitespace (space, tab and new line characters) to get an array of words.
  ```
  String[] words = text.split("[ \t\n\r]")
  ```

Hints on downloading

Once you construct a request as a String, create a URL object by passing the String to the URL's constructor. Call openConnection to create a URLConnection. A URLConnection is like a FileConnection, but with a URL instead. Once you open the connection, you can read from the input stream in the same way you normally read from standard input, for example using a Scanner.

```
URL url = new URL(urlString);
URLConnection connection = url.openConnection();
// You can now read from connection.getInputStream()
```

Hints on JSON parsing

We make two JSON parsing libraries available, you can choose which one to use: **`javax.json-1.0.jar`**, **`json-java-20231013.jar`**. The first one of these libraries provides a streaming API and the second provides an object model API. We have used the first one during our exercises and had examples in our lectures and the second one is documented here: http://stleary.github.io/JSON-java/index.html. Should you choose to use the object model API, it is your responsibility to read and understand its documentation.

The jar files can be found in the **`/cs/studres/CS1003/Coursework/A1/Tests`** directory. Both of these will be added to the classpath by stacscheck during automated testing, you may want to your add your chosen library to the classpath during development too.

Testing

This assignment makes use of the School's automated checker stacscheck. You should therefore ensure that your program can be tested using the auto-checker. It should help you see how well your program performs on the tests we have made public and will hopefully give you an insight into any issues prior to submission. The automated checking system is simple to run from the command line in your **`src`** directory:

```
stacscheck /cs/studres/CS1003/Coursework/A1/Tests
```

You can run subsets of the test suite independently as well. Try the following commands:

```
stacscheck /cs/studres/CS1003/Coursework/A1/Tests/badargs
stacscheck /cs/studres/CS1003/Coursework/A1/Tests/wellformed/summary
stacscheck /cs/studres/CS1003/Coursework/A1/Tests/wellformed/key-4101229
stacscheck /cs/studres/CS1003/Coursework/A1/Tests/wellformed/type-music
```

The first one only checks command line argument handling behaviour. The second and third one run tests against the local cache. Only the fourth one actually downloads data from Bored API and the test case only ensures that the program does not crash.

Make sure to type the commands exactly - occasionally copying and pasting from the PDF specification will not work correctly. If you are struggling to get it working, ask a demonstrator. Use a lab computer for running stacscheck for consistency between the development environments.

The automated checking system will only check the basic operation of your program. It is up to you to provide evidence that you have thoroughly tested your program.

Submission

Your report must be structured as follows (a report template is provided on StudRes):

- Overview: Give a short overview of the practical: what were you asked to do, and what did you achieve? Clearly list which parts you have completed, and to what extent.
- Design: Describe the design of your program. Justify the decisions you made. In particular, describe the classes you chose, the methods they contain, a brief explanation of why you designed your solution in the way that you did, and any interesting features of your Java implementation.
- Testing: Describe how you tested your program. In particular, describe how you designed different tests. Your report should include the output from a number of test runs to demonstrate that your program satisfies the specification. Please note that simply reporting the result of stacscheck is not enough; you should do further testing and explain in the report how you convinced yourself that your program works correctly.
- Evaluation: Evaluate the success of your program against what you were asked to do.
- Conclusion: Conclude by summarising what you achieved, what you found difficult, and what you would like to do given more time.

Don't forget to add a header including your matriculation number, the name of your tutor and the date. Do not include your name.

Upload

Create a zip archive that contains the **`src`** directory as well as a PDF copy of your report, and submit it using MMS, in the slot for Assignment #1. After doing this, it is important to verify that you have uploaded your submission correctly by downloading it from MMS. You can then run stacscheck directly on your zip file to make sure that your code still passes stacscheck. For example, if your file is called 99900000-Bored.zip, save it to your Downloads directory and run:

```
cd ~/Downloads
stacscheck --archive 99900000-Bored.zip
/cs/studres/CS1003/Coursework/A1/Tests
```

Run the above command on a lab computer. This is the exact command your marker will use to check your submission.

Marking Rubric

1-6	Very little evidence of work, software which does not compile or run, or crashes before doing any useful work. You should seek help from your tutor immediately.
7-10	An acceptable attempt to complete the main task with serious problems such as not compiling or crashing often during execution.
11-13	A competent attempt to complete the main task. Serious weaknesses such as using wrong data types, poor code design, weak testing, or a weak report riddled with mistakes.
14-16	A good attempt to complete the main task together with good code design, testing and a report.
17-18	Evidence of an excellent submission with no serious defects, good testing, accompanied by an excellent report.
19-20	An exceptional submission. A correct implementation of the main task, extensive testing, accompanied by an excellent report. In addition, it goes beyond the basic specification in a way that demonstrates use of concepts covered in class and other concepts discovered through self-learning.

See also the standard mark descriptors in the School Student Handbook:
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html

Lateness penalty

The standard penalty for late submission applies (Scheme A: 1 mark per 24-hour period, or part thereof):
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html

Good academic practice

The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/students/rules/academicpractice

Going Further

If you want to experiment with additional features, consider combining different kind of queries and try to filter by type as well as price/accessibility etc. If you decide to do this, make sure you make a copy of your basic solution in order not to inadvertently introduce errors in it.