MSCI 541: HW 1

Jonathan Chen
20722167
September 28th, 2021

Question 1
1a) Precision is defined as the total number of relevant items found divided by the total number of items matched. In the context of search engines, precision enhancement could come in the form of increasing the number of relevant items found or decreasing the number of items returned.

On the other hand, recall is defined as the total number of relevant items found divided by the total number of relevant items in the collection. Thus, recall enhancement can be achieved by discovering more relevant items within the collection.

1b) A precision enhancing technique that an indexer could employ for the alphabetical subject card catalog would be to include less information on each card. By including less information, fewer results will be returned and thus the number of false positives would decrease, which would cause the precision to increase. Alternatively, a recall enhancing technique would be to include more information on each card. By including more information, more relevant items would be found because there would be a larger base of information to draw upon. Although there would be more false positives and the precision would decrease, having more information on each card would increase the recall because more relevant items would be found.

Question 2
2. A relational database and a search engine are similar because they are both information retrieval systems that contain information from which data can be queried. However, one big difference between a search engine and a relational database is that search engines can use implicit queries, or even natural language as a query, whereas relational databases always use a formal query language (SQL). Moreover, a database always returns the correct results and there is 100% user satisfaction for the results by the user, but a user can be dissatisfied with the results of a search engine because the quality of the results are subjective and depend on the user. Finally, all optimizations of a relational database centre around improving its speed, but for a search engine, optimizations must also consider a tradeoff between speed and quality.

Question 3
3. The advantage of downcasing all terms in a search engine is that the user does not need to be careful with letter casing in the query, so the search engine will be able to match words at the start of a sentence. This makes the search engine more robust to different formats of user queries by allowing for more patterns to be matched. The disadvantage is that the search engine loses the ability to distinguish certain acronyms and proper nouns. The search engine may return irrelevant

results that are unrelated to the original query because the search engine would not be able to recognize a proper noun from normal terms used in a sentence.

Question 4
For Question 4, I created the IndexEngine and GetDoc programs with Java (version 17, 2021-09-14 LTS, Java(TM) SE Runtime Environment (build 17+35-LTS-2724), using the IntelliJ IDE.

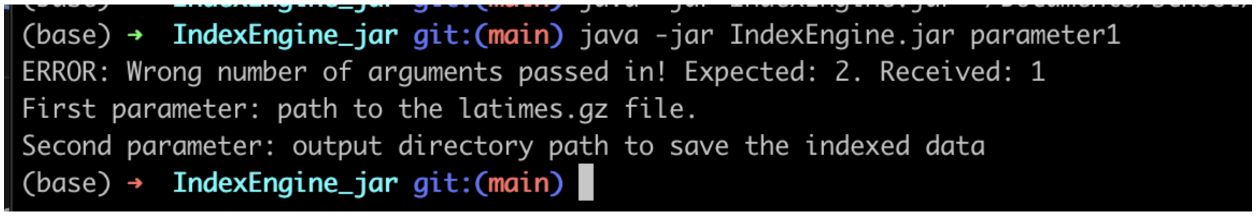
Within IndexEngine, my approach consisted of reading the gzipped LaTimes file line by line in a while loop and appending each line to a StringBuffer object. Once a line containing the "</DOC>" tag is reached, then the program treats this as the end of a document and begins to process the metadata for that document. This involves using regex to retrieve the "docno" and "headline" from the document, and parsing the docno to retrieve the date. All of this information is stored in a hashmap where the key is the metadata type, and the value is the metadata value. The metadata is then stored to a *metadata.txt* file, which contains all of the metadata for every document. Afterwards, the raw document is added to the metadata hashmap, and the rest of the metadata, along with the raw document information, is saved to a unique file named *[DOCNO].txt*, within a file structure of YY/MM/DD. Subsequently, a hashmap that stores the mapping of the internal integer id to the docno is updated. Once both the *metadata.txt* file and the *[DOCNO].txt* files are updated, the metadata hashmap and the StringBuffer object containing the document are cleared. The LaTimes file is read line by line again until the next </DOCNO> tag is reached, and the process repeats. Finally, after all the documents are processed, the hashmap containing the mapping of all the internal integer ids to their respective docnos is saved into the *idMapping.txt* file.

As for the GetDoc program, it takes in as parameters either the internal integer id or the docno. If an internal integer id is passed in, the program reads the *idMapping.txt* file to retrieve the associated docno for that id, otherwise the program stores the docno that is passed in into a variable. Once the docno is known, the program parses the docno to retrieve the file folder location of the desired file using the date contained with the docno. The desired output text file is then read into a string and printed to the console.

To run the programs, the pre-built jar files can be used. To run the IndexEngine, go to the *out/artifacts/IndexEngine_jar/* directory in the command line and then run *java -jar IndexEngine.jar [path_to_latimes.gz] [output_path]*. Similarly, to run the GetDoc program, go to the *out/artifacts/GetDoc_jar/* directory in the command line and then run *java -jar GetDoc.jar [path to indexed data] ['id' or 'docno'] [id or docno value]*.

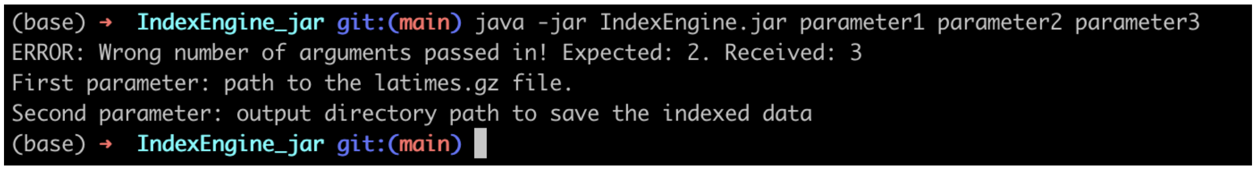
Test Plan
Test Case #1: Too few parameters passed into IndexEngine.

Expectation: Fail with error message detailing how to run IndexEngine.java
Output:



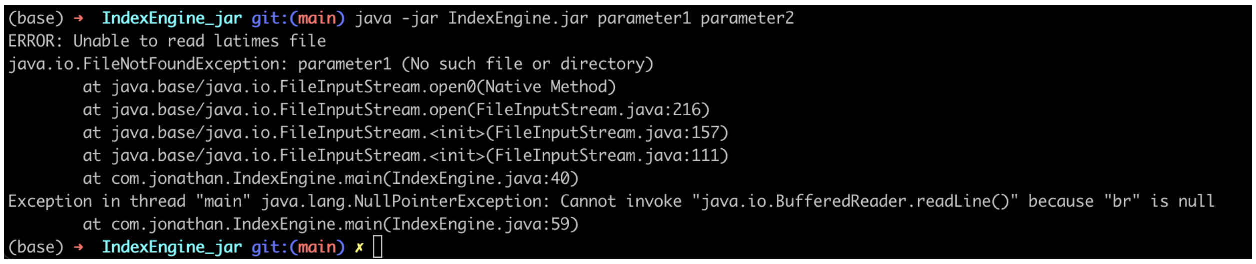
```
(base) ➜  IndexEngine_jar git:(main) java -jar IndexEngine.jar parameter1
ERROR: Wrong number of arguments passed in! Expected: 2. Received: 1
First parameter: path to the latimes.gz file.
Second parameter: output directory path to save the indexed data
(base) ➜  IndexEngine_jar git:(main)
```

Test Case #2: Too many parameters passed into IndexEngine.
Expectation: Fail with error message detailing how to run IndexEngine.java
Output:



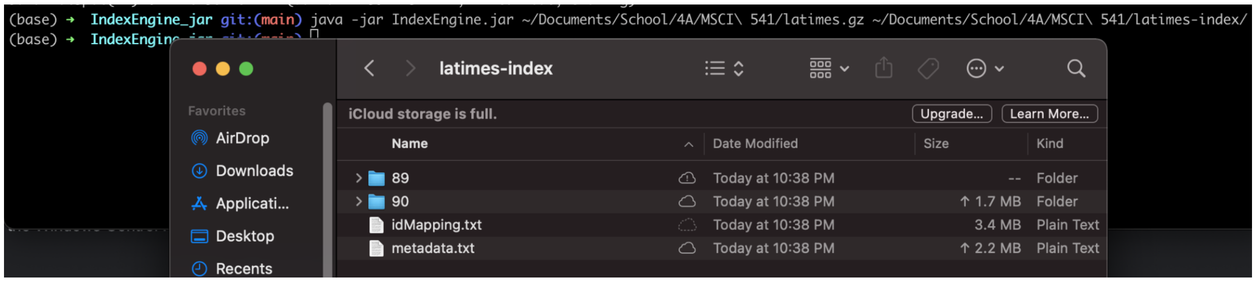
```
(base) ➜  IndexEngine_jar git:(main) java -jar IndexEngine.jar parameter1 parameter2 parameter3
ERROR: Wrong number of arguments passed in! Expected: 2. Received: 3
First parameter: path to the latimes.gz file.
Second parameter: output directory path to save the indexed data
(base) ➜  IndexEngine_jar git:(main)
```

Test Case #3: Pass in an invalid location for the latimes.gz file to IndexEngine.
Expectation: Fail with error message
Output:



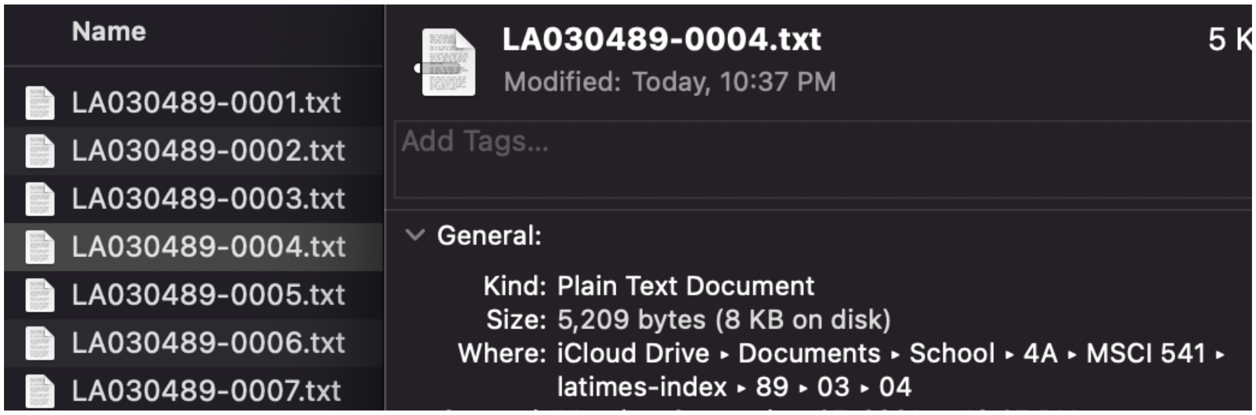
```
(base) ➜  IndexEngine_jar git:(main) java -jar IndexEngine.jar parameter1 parameter2
ERROR: Unable to read latimes file
java.io.FileNotFoundException: parameter1 (No such file or directory)
	at java.base/java.io.FileInputStream.open0(Native Method)
	at java.base/java.io.FileInputStream.open(FileInputStream.java:216)
	at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
	at java.base/java.io.FileInputStream.<init>(FileInputStream.java:111)
	at com.jonathan.IndexEngine.main(IndexEngine.java:40)
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.io.BufferedReader.readLine()" because "br" is null
	at com.jonathan.IndexEngine.main(IndexEngine.java:59)
(base) ➜  IndexEngine_jar git:(main) ✗
```

Test Case #4: Happy path with proper parameters passed to IndexEngine.
Expectation: Pass with directory created at the desired file location, containing the indexed data
Output:



```
(base) ➜  IndexEngine_jar git:(main) java -jar IndexEngine.jar ~/Documents/School/4A/MSCI\ 541/latimes.gz ~/Documents/School/4A/MSCI\ 541/latimes-index/
(base) ➜  IndexEngin
```

Name	Date Modified	Size	Kind
89	Today at 10:38 PM	--	Folder
90	Today at 10:38 PM	↑ 1.7 MB	Folder
idMapping.txt	Today at 10:38 PM	3.4 MB	Plain Text
metadata.txt	Today at 10:38 PM	↑ 2.2 MB	Plain Text

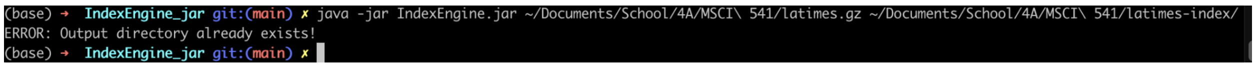


```
Name
LA030489-0001.txt
LA030489-0002.txt
LA030489-0003.txt
LA030489-0004.txt
LA030489-0005.txt
LA030489-0006.txt
LA030489-0007.txt
```

LA030489-0004.txt
Modified: Today, 10:37 PM

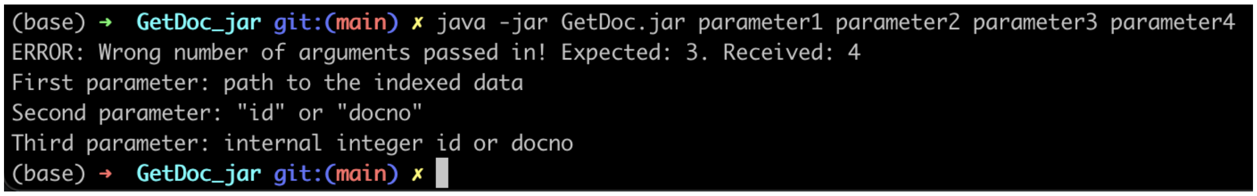
General:
Kind: Plain Text Document
Size: 5,209 bytes (8 KB on disk)
Where: iCloud Drive ▸ Documents ▸ School ▸ 4A ▸ MSCI 541 ▸ latimes-index ▸ 89 ▸ 03 ▸ 04

Test Case #5: Output directory for IndexEngine already exists.
Expectation: Fail with error message saying that the directory already exists
Output:



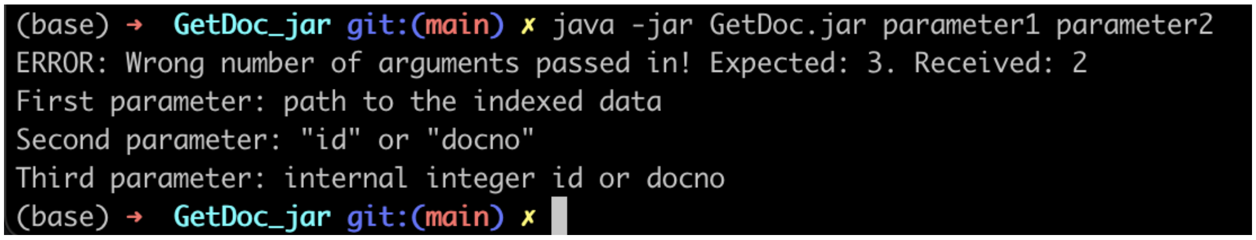
```
(base) ➜  IndexEngine_jar git:(main) ✗ java -jar IndexEngine.jar ~/Documents/School/4A/MSCI\ 541/latimes.gz ~/Documents/School/4A/MSCI\ 541/latimes-index/
ERROR: Output directory already exists!
(base) ➜  IndexEngine_jar git:(main) ✗
```

Test Case #6: Too many parameters passed into GetDoc.
Expectation: Fail with error message detailing how to run GetDoc.java
Output:



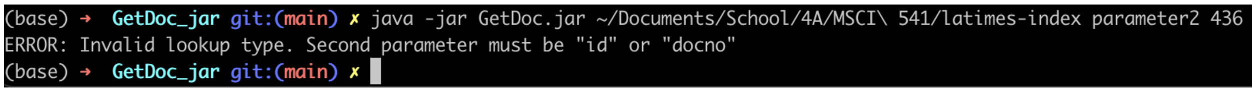
```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar parameter1 parameter2 parameter3 parameter4
ERROR: Wrong number of arguments passed in! Expected: 3. Received: 4
First parameter: path to the indexed data
Second parameter: "id" or "docno"
Third parameter: internal integer id or docno
(base) ➜  GetDoc_jar git:(main) ✗
```

Test Case #7: Too few parameters passed into GetDoc.
Expectation: Fail with error message detailing how to run GetDoc.java
Output:



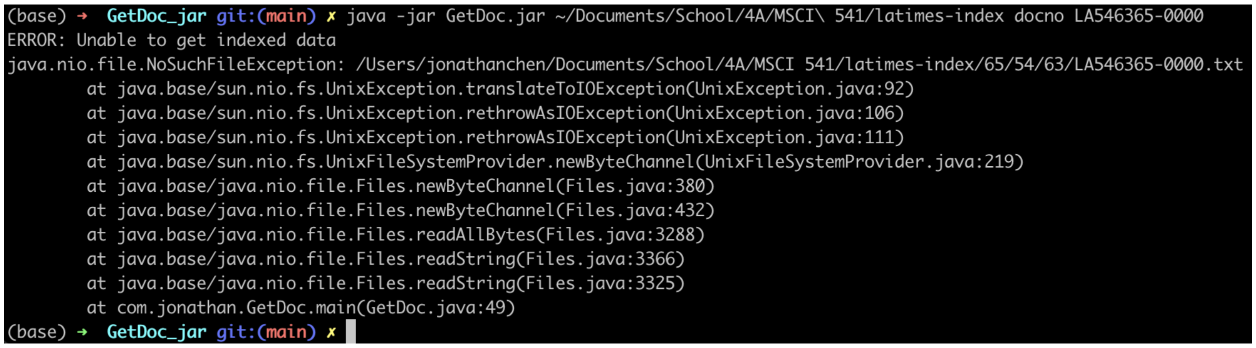
```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar parameter1 parameter2
ERROR: Wrong number of arguments passed in! Expected: 3. Received: 2
First parameter: path to the indexed data
Second parameter: "id" or "docno"
Third parameter: internal integer id or docno
(base) ➜  GetDoc_jar git:(main) ✗
```

Test Case #8: Invalid query type passed to GetDoc (i.e. not “id” or not “docno”).
Expectation: Fail with error message on what parameter is expected
Output:



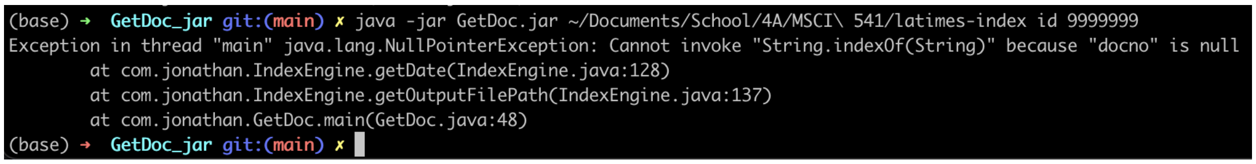
```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar ~/Documents/School/4A/MSCI\ 541/latimes-index parameter2 436
ERROR: Invalid lookup type. Second parameter must be "id" or "docno"
(base) ➜  GetDoc_jar git:(main) ✗
```

Test Case #9: Invalid docno passed to GetDoc.
Expectation: Fail with error message
Output:



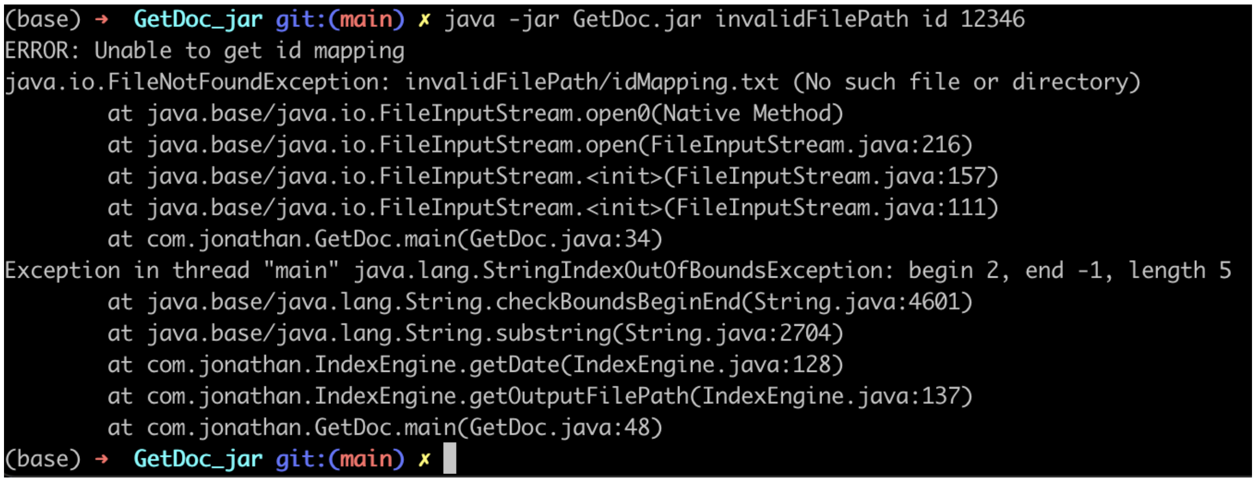
```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar ~/Documents/School/4A/MSCI\ 541/latimes-index docno LA546365-0000
ERROR: Unable to get indexed data
java.nio.file.NoSuchFileException: /Users/jonathanchen/Documents/School/4A/MSCI 541/latimes-index/65/54/63/LA546365-0000.txt
        at java.base/sun.nio.fs.UnixException.translateToIOException(UnixException.java:92)
        at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:106)
        at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:111)
        at java.base/sun.nio.fs.UnixFileSystemProvider.newByteChannel(UnixFileSystemProvider.java:219)
        at java.base/java.nio.file.Files.newByteChannel(Files.java:380)
        at java.base/java.nio.file.Files.newByteChannel(Files.java:432)
        at java.base/java.nio.file.Files.readAllBytes(Files.java:3288)
        at java.base/java.nio.file.Files.readString(Files.java:3366)
        at java.base/java.nio.file.Files.readString(Files.java:3325)
        at com.jonathan.GetDoc.main(GetDoc.java:49)
(base) ➜  GetDoc_jar git:(main) ✗
```

Test Case #10: Invalid id passed to GetDoc.
Expectation: Fail with error message
Output:



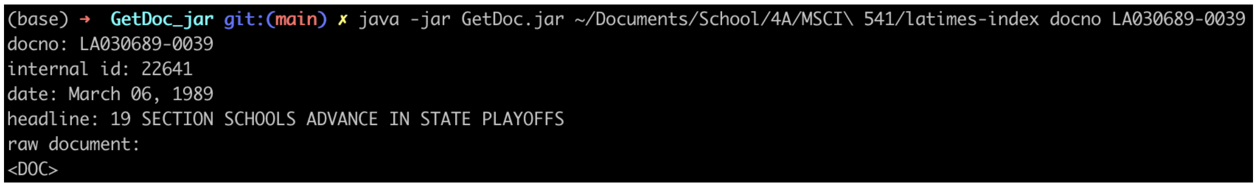
```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar ~/Documents/School/4A/MSCI\ 541/latimes-index id 9999999
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.indexOf(String)" because "docno" is null
        at com.jonathan.IndexEngine.getDate(IndexEngine.java:128)
        at com.jonathan.IndexEngine.getOutputFilePath(IndexEngine.java:137)
        at com.jonathan.GetDoc.main(GetDoc.java:48)
(base) ➜  GetDoc_jar git:(main) ✗
```

Test Case #11: Invalid file path to indexed data passed to GetDoc
Expectation: Fail with error message
Output:



```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar invalidFilePath id 12346
ERROR: Unable to get id mapping
java.io.FileNotFoundException: invalidFilePath/idMapping.txt (No such file or directory)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:216)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:111)
        at com.jonathan.GetDoc.main(GetDoc.java:34)
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 2, end -1, length 5
        at java.base/java.lang.String.checkBoundsBeginEnd(String.java:4601)
        at java.base/java.lang.String.substring(String.java:2704)
        at com.jonathan.IndexEngine.getDate(IndexEngine.java:128)
        at com.jonathan.IndexEngine.getOutputFilePath(IndexEngine.java:137)
        at com.jonathan.GetDoc.main(GetDoc.java:48)
(base) ➜  GetDoc_jar git:(main) ✗
```

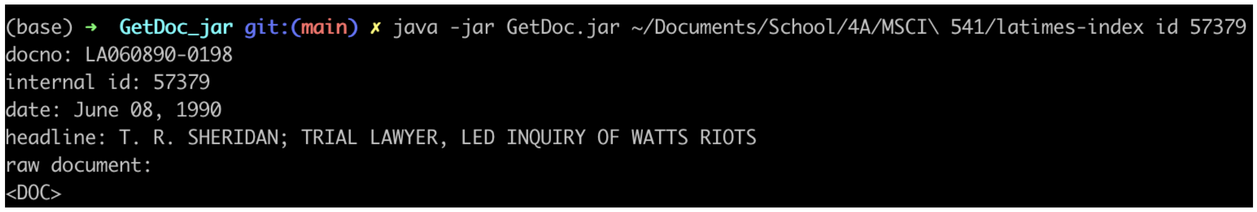
Test Case #12: Valid parameters for docno passed to GetDoc.
Expectation: Metadata and raw document information displayed to console for the docno
Output:



```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar ~/Documents/School/4A/MSCI\ 541/latimes-index docno LA030689-0039
docno: LA030689-0039
internal id: 22641
date: March 06, 1989
headline: 19 SECTION SCHOOLS ADVANCE IN STATE PLAYOFFS
raw document:
<DOC>
```

Note: the output for raw document continues but is not included because of the TREC agreement

Test Case #13: Valid parameters for id passed to GetDoc.
Expectation: Metadata and raw document information displayed to console for the id
Output:



```
(base) ➜  GetDoc_jar git:(main) ✗ java -jar GetDoc.jar ~/Documents/School/4A/MSCI\ 541/latimes-index id 57379
docno: LA060890-0198
internal id: 57379
date: June 08, 1990
headline: T. R. SHERIDAN; TRIAL LAWYER, LED INQUIRY OF WATTS RIOTS
raw document:
<DOC>
```

Note: the output for raw document continues but is not included because of the TREC agreement