1) It is located inside the src folder in the Scorer and ScorerMain

2) First, the JSON file was read using the library JSON-simple for java. The read-in objects were then converted to a document object which stores all its important information. From there an inverted index was created. I chose to use the inverted index i created in the previous project as it had all the necessary information and methods to use for this project. A scorer class was then created with the index as its input. The score class has the methods bm25 and Ql which perform the ranking on the data using each of the ranking algorithms. I chose to make the parameters for the algorithms k1,k2, b, and mew as final variables in the code. This enabled me to change it in one place in the code to get different outputs but was also simpler to implement than command line parameters. There is also a format methods that given certain parameters can return the correct output properly formatted which always all the quires to be made and format quickly and with ease.

3) Libraries used:
java.util.ArrayList -- Used to store data
java.util.Collections -- Called to sort data
java.util.HashMap -- Used to store data
java.util.Map -- used to store data
InvertedIndex -- used to create index
Document -- used to create index.
java.io.File -- used to write and read data to file
java.io.FileReader used to write data to file
java.io.FileWriter -- used to write data to file
java.io.IOException -- used to handle errors
java.io.PrintWriter - used to write data to file
org.json.simple.JSONArray -- Used to read in JSON file
org.json.simple.JSONObject -- Used to read in JSON file
org.json.simple.parser.JSONParser -- Used to read in JSON file
org.json.simple.parser.ParseException -- Used to read in JSON file

4) I expect it to be a very bad query as all the words will be found in nearly every document and you will get very negative scores for all documents that will tell you very little about the actual ranking of the query.

5) I do not expect setting the scene to be a good score. First due to the term in the query. The will be found in all the documents and will hurt not help the search relanacney. In addition setting and scene are both words that are broad and by themselve provide to little context on weather a search is good so even if the words is in them it wont definitely be relevant to what the user wants.

6) The code preforms well on the Q5 search with most of the searches being at least somewhat relevant for both BM25 and QL. In Addition there was a lot of overlap

between the 2 sets of results and the overlapped one where mainly relevant. That being said the BM25 performed slightly better for this search when compared to the QL algorithm. When you only look at the result in which they don't overlap, BM25 had the highest average relevance based on my opinion of the scenes when compared to QL. Mainly as BM25 had more scene from plays like Macbetha and Hamlet than which are more relevant to words like jester and fool than QL which had plays like A mid nigh summer. Both prefrem well but BM25 performed slightly better.