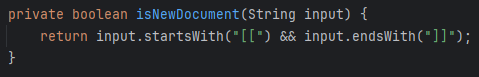
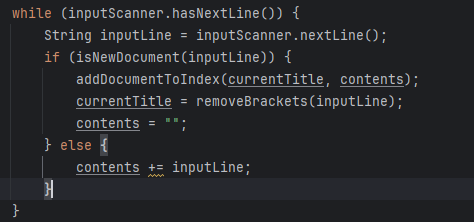
We have implemented 2 solutions, one in Java using the Lucene IR system and one in Python using the Whoosh IR system. This document will contain for both solutions the following:

* the description of the code
* results
* answers for all questions

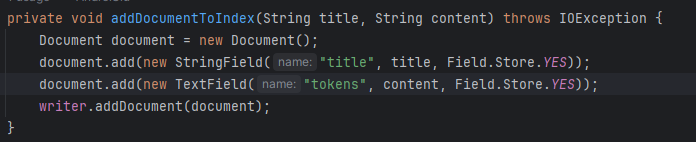
## Lucene

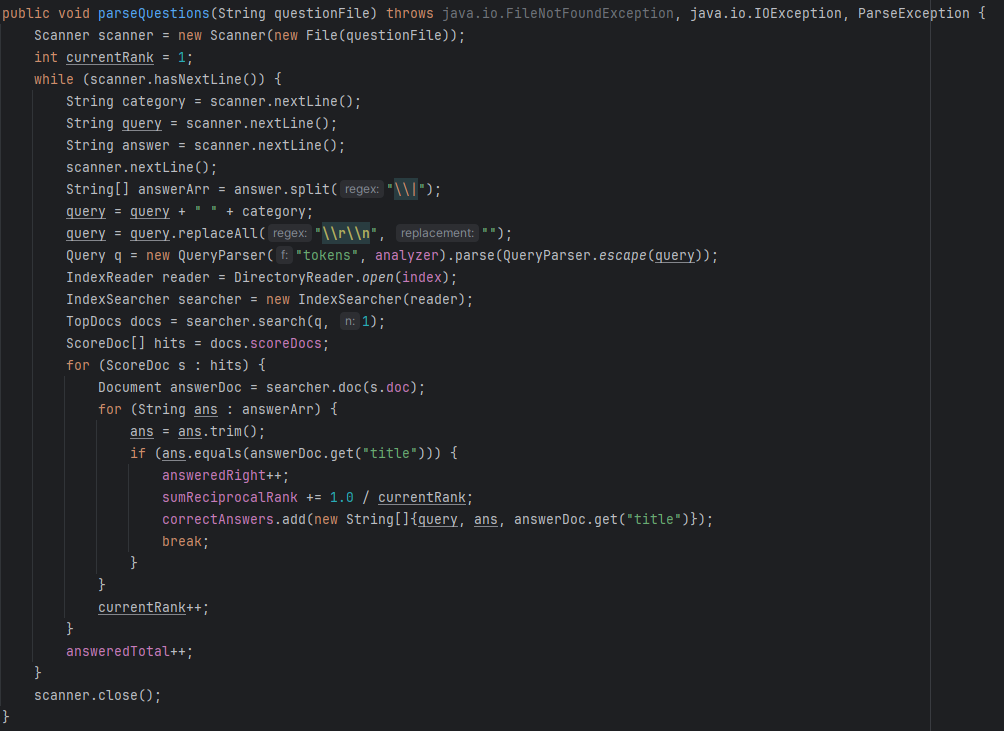
### Description of the code

In the **indexer.java** we separate the documents in the index for ach Wikipedia page:  
  


In this class we have a method called **buildIndex** that indexes pages and processes terms. We have tried ‘stop word’ and ‘stemming’ provided by Lucene`s **EnglishAnalyzer** method but we got better results using Lucene`s **StandardAnalyzer** method.

Another method worth mentioning in this class is the **addDocumentToIndex** method that complements the one mentioned above that adds the pages` title and content to the index

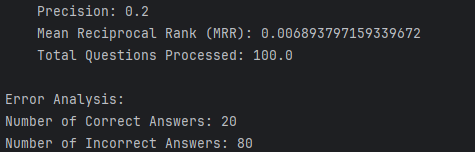


Next is the class called **Watson.java** that searches the indexed files for the answers. Worth mentioning here is the method **parseQuestions**

Here we only yield the first provided answer by the system as to provide the p@1 measurement.

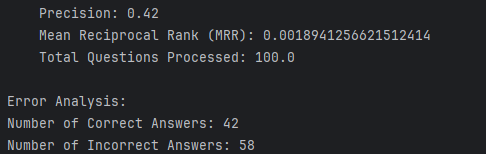
In this class we also measure and return the result of MRR.

### Results



When considering only the questions to which the system got the answer right on the first try we get the above results.

But when we tell the system that the answer should be in the first 10 yielded results we get the following results:



* **Why do you think the correct questions can be answered by such a simple system?:** Wikipedia pages often have structured and well-formatted information  
  The clue captures only the essential information  
  Clue and answer are usually straightforward
* **What problems do you observe for the questions answered incorrectly?**Questions can be ambiguous

### Answers to the questions

* **Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.):** Used StandardAnalyzer for java. It lowercases each token, removes common words and punctuations and uses stop words for the English language.
* **What issues specific to Wikipedia content did you discover, and how did you address them?:** Ambiguity of inconsistent formats, because some pages have a lot of information whereas other don’t and weird title.
* **Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset?:** Implemented using Lucene's IndexSearcher to query the indexed data, returning the most similar Wikipedia page title for a given Jeopardy clue.
* **If the latter, what is the best algorithm for selecting the subset of words from the clue?:**   
  Queries are built using entire clues, optionally combined with category information for context. Lucene's queryparser is used for this purpose.
* **Are you using the category of the question?:** No
* **Note: not all the above metrics are relevant here! Justify your choice and then report performance using the metric(s) of your choice.:**   
  Chose Precision at 1 (P@1) and Mean Reciprocal Rank (MRR) as metrics.

P@1 is relevant for single-answer questions, and MRR accounts for the rank of correct answers. It is very important for a jeopardy show to yield the correct answer in the first try!

## Whoosh

### Description of the code

**naive\_indexer /indexer.py** – This script uses Whoosh for indexing the Wikipedia pages. It is called naïve indexer because the whole unprocessed text is used as content in the document. A screen shot of a computer program

Description automatically generatedHere it can be seen that each Wikipedia page appears as a separate document in the index. The extract\_pages function (found in the same script) take

**Improved\_indexer/indexer.py** - This script uses Whoosh for indexing the Wikipedia pages. It is called improved indexer because the content of the page is processed before being added as content in the document. Processed = lemmatization + stop words A screen shot of a computer

Description automatically generated  
Here it can be seen that Spacy was used in order to retrieve the lemma of a token. If the token is a stop word it will be skipped. The rest of the script is identic with the one from the naïve indexer

**indexer.py** – Run this script with the following arguments: “python ./indexer.py True True” in order to run both the naïve indexer and the improved indexer

**measure\_raw\_clue.py** – Script used for measuring the performance of the Jeopardy system. All 3 types of metrics are being used (P@1, NDCG, MRR)

**measure\_keywords\_clue.py** - Script used for measuring the performance of the Jeopardy system. All 3 types of metrics are being used (P@1, NDCG, MRR). Compared to the raw\_clue measurement script, this one build the query string as a subset of keywords from the clue. A black screen with white text

Description automatically generated  
Spacy was also used for this task. It has been observed that using keywords for the query string yields better results

**measure.py** – Execute this script too run all possible combinations of measurements:

1. Improved indexer – entire clue query string
2. Naïve indexer – full clue query string
3. Improved indexer – keywords clue query string
4. Naïve indexer – keywords clue query string

### Results

A screenshot of a computer

Description automatically generated

* **Perform an error analysis of your best system. How many questions were answered correctly/incorrectly?:** Using the naïve indexer, 8-9 questions have been answered correctly. Using the improved indexer, only 4 questions have been answered correctly. It results that lemmatization and eliminating stop words has not helped in this case
* **Why do you think the correct questions can be answered by such a simple system?:** Wikipedia pages often have structured and well-formatted information  
  The clue captures only the essential information  
  Clue and answer are usually straightforward
* **What problems do you observe for the questions answered incorrectly?**Questions can be ambiguous  
  Were context is relevant system can not perform

### Answers to the questions

* **Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.):** Used spacy from Python, which is a Industrial-Strength Natural Language Processing library. It has implementation for lemmatization of a word and stop words for the English languageA black background with white text

  Description automatically generated  
  Also tried without processing the text, by creating a document containing the entire content of the page. Surprisingly this yielded better results compared to the processed text.
* **Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset?:** Tried both options, using the entire clue and just a set of keywords from it (again using spacy)  
  **A black screen with text

  Description automatically generated**  
  The searches yield better results when the query string is built out of keywords
* **If the latter, what is the best algorithm for selecting the subset of words from the clue?:** Only tried the above solution
* **Are you using the category of the question?:** No, using the category of the question yields worse results
* **Note: not all the above metrics are relevant here! Justify your choice and then report performance using the metric(s) of your choice.:** Tried all the 3 metrics discussed (P@1, NDCG, MRR), but I believe in our case Precision at 1 is the most suitable, because in the context of the Quiz Show Jeopardy, only the first answer matters 😊