Our goal: improve “top 10 precision” of search engine. For the average query, how many of our search engine’s top 10 results are actually relevant?

Lucene:

index stores information about Documents
each Document has Fields
each Field has a name and a value

IndexWriter: you parse your corpus, add Documents to the IndexWriter, then write out the index to binary files. IndexWriter uses:
- Analyzer: “represents a policy for extracting index terms from text.” Uses a Tokenizer and various TokenFilters. You apply this to the corpus when you index it and to your queries when you parse them. It handles:
  o tokenizing
  o stop words
  o stemming
  o lowercase
  o other minor details (remove dots from acronyms, remove ‘s from words, etc.)
- Similarity: defines the scoring function for search, i.e. how we decide which documents to return for a given query and how to rank them.
  o $\text{score}(q,d) = \sum_{t \in q} \text{tf} \times \text{idf} \times \text{getBoost}(t, \text{field}) \times \text{lengthNorm} \times \text{coord} \times \text{queryNorm}$
  o $\text{tf}$: term frequency. DefaultSimilarity has $\sqrt{\text{freq}}$. In class we mentioned $\text{wf} = 1 + \log \text{tf}$ as an option. There are probably others you should try.
  o $\text{idf}$: inverse document frequency. DefaultSimilarity has $1 + \log (n/(\text{df}+1))$. This is almost the same as what we saw in class, $\log (n/\text{df})$. Again, look for other possibilities.
  o $\text{getBoost}$: you can give certain fields increased weight. In this assignment you should try changing the relative weights of the title and contents fields.
  o $\text{lengthNorm}$: tries to normalize for field length. Area of recent interest in IR – see http://citeseer.ist.psu.edu/singhal96pivoted.html. “Matches in longer fields are less precise, so implementations of this method usually return smaller values when numTokens is large, and larger values when numTokens is small.” Might not be especially helpful with this corpus since we only have two fields and their lengths don’t vary a lot from doc to doc, but maybe it can be helpful.
  o $\text{coord}$: multiply score by factor based on how many terms match from the query. Seems to duplicate part of what $\text{tf.idf}$ is trying to accomplish, but this is more direct.
  o $\text{queryNorm}$: ignore it. It doesn’t affect ranking.

Now your index is built and you run queries on it.
A QueryParser parses your free text queries. Use the same Analyzer that you used to build the index. An IndexSearcher searches the index and returns a Hits object. You can iterate through the Hits and get the top results, then compare them to the qrels lists.