Introduction to Lucene

Presenter

Quach Dinh Hoang

Main References

- Christopher D. Manning and Pandu Nayak, **Lucene Tutorial**, Lecture notes, Stanford, 2019. (https://web.stanford.edu/class/cs276/handouts/lecture-lucene.pptx)
- Michael McCandless, Erik Hatcher, Otis Gospodnetic, **Lucene in Action, 2nd Edition**, Manning, 2010. (https://www.manning.com/books/lucene-in-action-second-edition)
  - Covers Lucene 3.0.1. It's now up to 8.8.0
- Lucene documentations (https://lucene.apache.org/core/8_8_0/index.html)

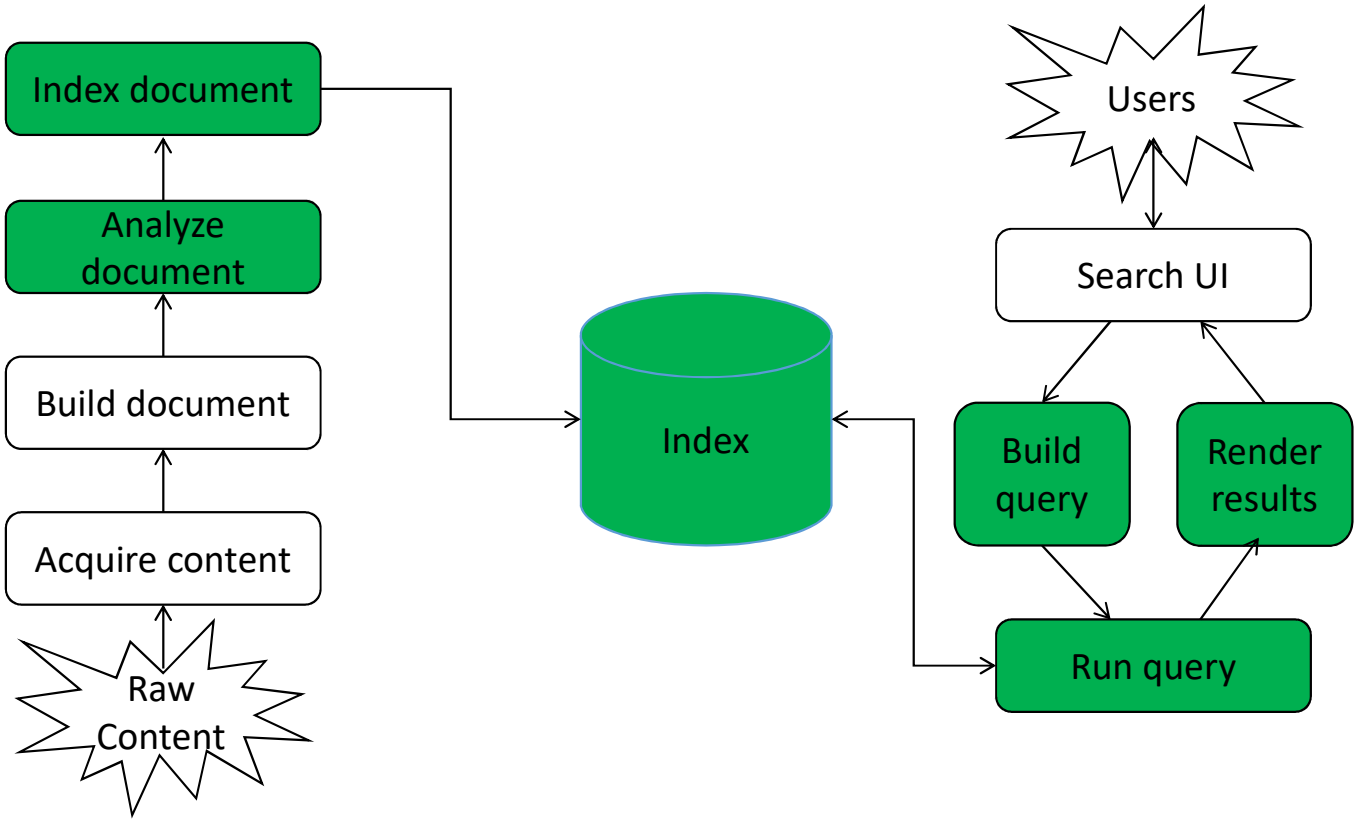
Open source IR systems

- Widely used academic systems
  - **Terrier** (Java, U. Glasgow) http://terrier.org
  - **Indri/Galago/Lemur** (C++ (& Java), U. Mass & CMU)
  - Tail of others (**Zettair**, …)
- Widely used non-academic open source systems
  - **Lucene**
    - Things built on it: **Solr**, **ElasticSearch**
  - A few others (**Xapian**, …)

Lucene

- Open source **Java** library for indexing and searching
  - Lets you add search to your application
  - Not a complete search system by itself
  - Written by **Doug Cutting**
- Used by: **Apple**, **Twitter**, **LinkedIn**, **IBM**, **CiteSeerX**, **Eclipse**, **Nutch**, …
  - … and many more (see https://cwiki.apache.org/confluence/display/lucene/PoweredBy)
- Ports/integrations to other languages
  - **C/C++, C#, Ruby, Perl, Python, PHP**, …

Lucene in a search system



Indexing with Lucene

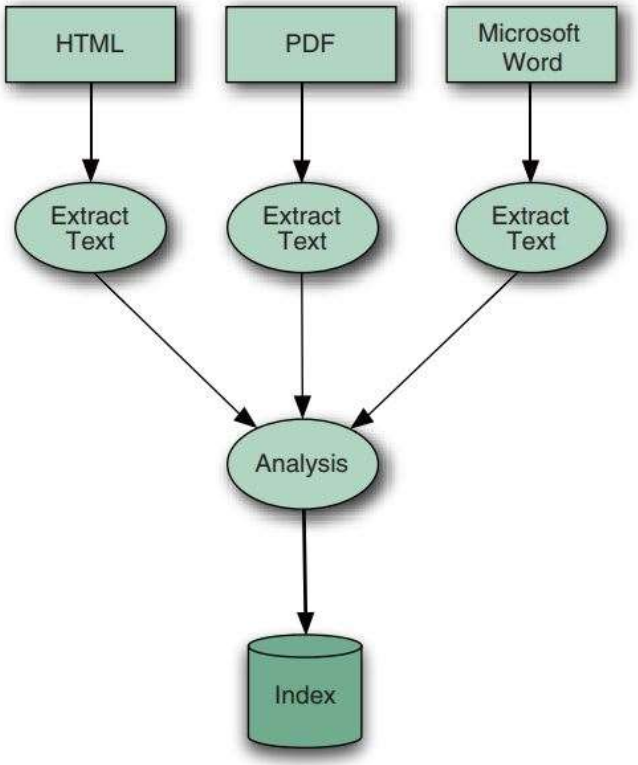


Figure 2.1 **Indexing with Lucene breaks down into three main operations: extracting text from source documents, analyzing it, and saving it to the index.**

Core indexing classes

- **`IndexWriter`**
  - Central component that allows you to create a new index, open an existing one, and add, remove, or update documents in an index
  - Built on an **`IndexWriterConfig`** and a **`Directory`**
- **`Directory`**
  - Abstract class that represents the location of an index
- **`Analyzer`**
  - Extracts tokens from a text stream
- **`Document`**
  - Represents a collection of named **`Fields`**. Text in these **`Fields`** are indexed.
- **`Field`**
  - **`StringFields`** are indexed but not tokenized
  - **`TextFields`** are indexed and tokenized
  - …

Core indexing classes

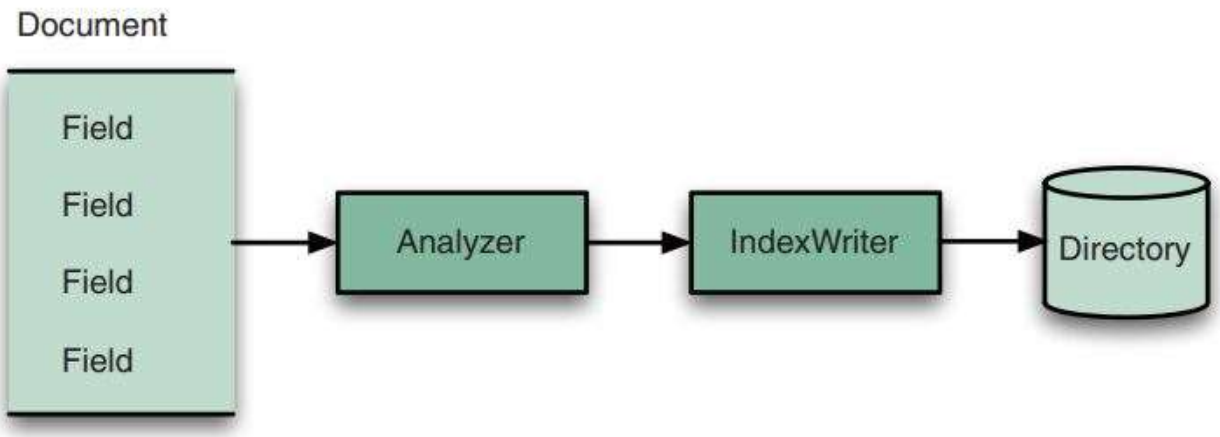


Figure 1.5 Classes used when indexing documents with Lucene

Document and Field

- A **`Document`** is the atomic unit of indexing and searching
  - A **`Document`** contains **`Fields`**
- **`Fields`** have a name and a value
  - You have to translate raw content into **`Fields`**
  - Examples: Title, author, date, abstract, body, URL, keywords, ...
  - Different documents can have different fields
  - Search a field using name:term, e.g., title:lucene

Fields

- **`Fields`** may
  - **Be indexed or not**
    - Indexed fields may or may not be analyzed (i.e., tokenized with an **`Analyzer`**)
      - Non-analyzed fields view the entire value as a single token (useful for URLs, paths, dates, social security numbers, ...)
  - **Be stored or not**
    - Useful for fields that you'd like to display to users (useful for title, abstract)
  - **Optionally store term vectors**
    - Like a positional index on the **`Field`**'s terms
    - Useful for highlighting, finding similar documents, categorization (useful for body)

Analyzer

- Tokenizes the input text
- Common **`Analyzers`**
  - **`WhitespaceAnalyzer`**
    Splits tokens on whitespace
  - **`SimpleAnalyzer`**
    Splits tokens on non-letters, and then lowercases
  - **`StopAnalyzer`**
    Same as **`SimpleAnalyzer`**, but also removes stop words
  - **`StandardAnalyzer`**
    Most sophisticated analyzer that knows about certain token types, lowercases, removes stop words, ...

Analysis example

- Text = “The quick brown fox jumped over the lazy dog”
- **`WhitespaceAnalyzer`**
  - [The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]
- **`SimpleAnalyzer`**
  - [the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]
- **`StopAnalyzer`**
  - [quick] [brown] [fox] [jumped] [over] [lazy] [dog]
- **StandardAnalyzer**
  - [quick] [brown] [fox] [jumped] [over] [lazy] [dog]

Another analysis example

- Text = “XY&Z Corporation – xyz@example.com”
- **`WhitespaceAnalyzer`**
  - [XY&Z] [Corporation] [-] [xyz@example.com]
- **`SimpleAnalyzer`**
  - [xy] [z] [corporation] [xyz] [example] [com]
- **`StopAnalyzer`**
  - [xy] [z] [corporation] [xyz] [example] [com]
- **StandardAnalyzer**
  - [xy&z] [corporation] [xyz@example.com]

What's inside an Analyzer?

- **Char Filter**
  - Preprocesses the string of characters before it is passed to the tokenizer.
  - A character filter may be used to strip out HTML markup, or to convert "&" characters to the word "and".
- **Tokenizer**
  - Breaks a string down into a stream of terms or tokens.
  - A simple tokenizer might split the string up into terms wherever it encounters whitespace or punctuation.
- **Token Filter**
  - Accepts a stream of tokens from a tokenizer and can:
    - modify tokens (eg lowercasing)
    - delete tokens (eg remove stopwords)
    - add tokens (eg synonyms)

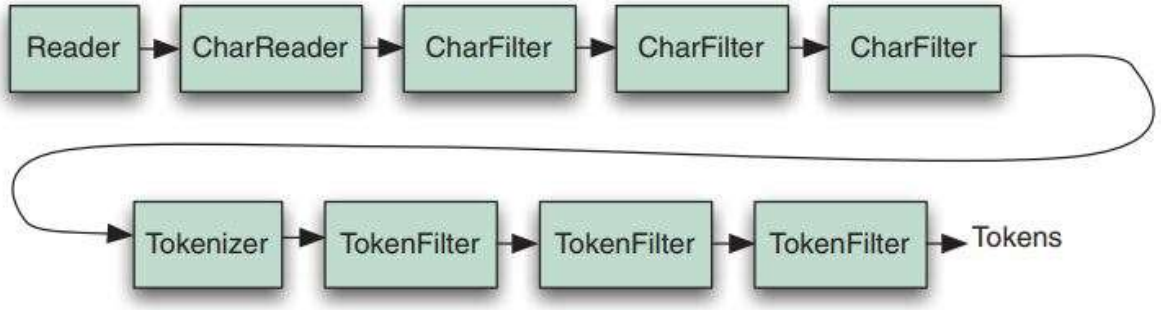
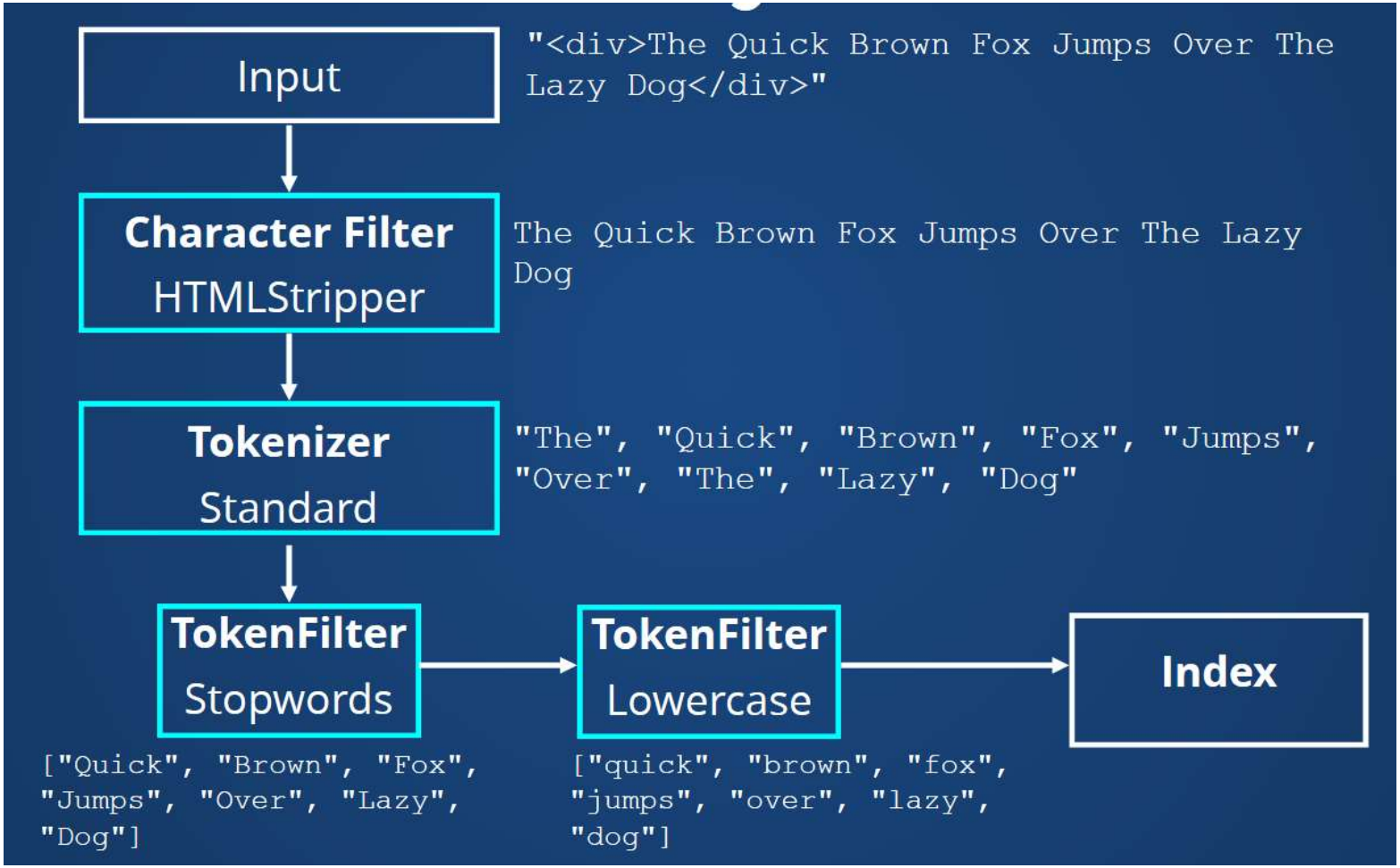


Figure 4.6
Analysis chain that includes character normalization

What's inside an Analyzer?



Source: http://slides.com/maderskog/deck-14/fullscreen#/7/13

Char Filter

- Preprocesses the string of characters before it is passed to the tokenizer.
- A character filter may be used to strip out HTML markup, or to convert "&" characters to the word "and".
  - Mapping Char Filter
  - HTML Strip Char Filter
  - Pattern Replace Char Filter

Tokenizer

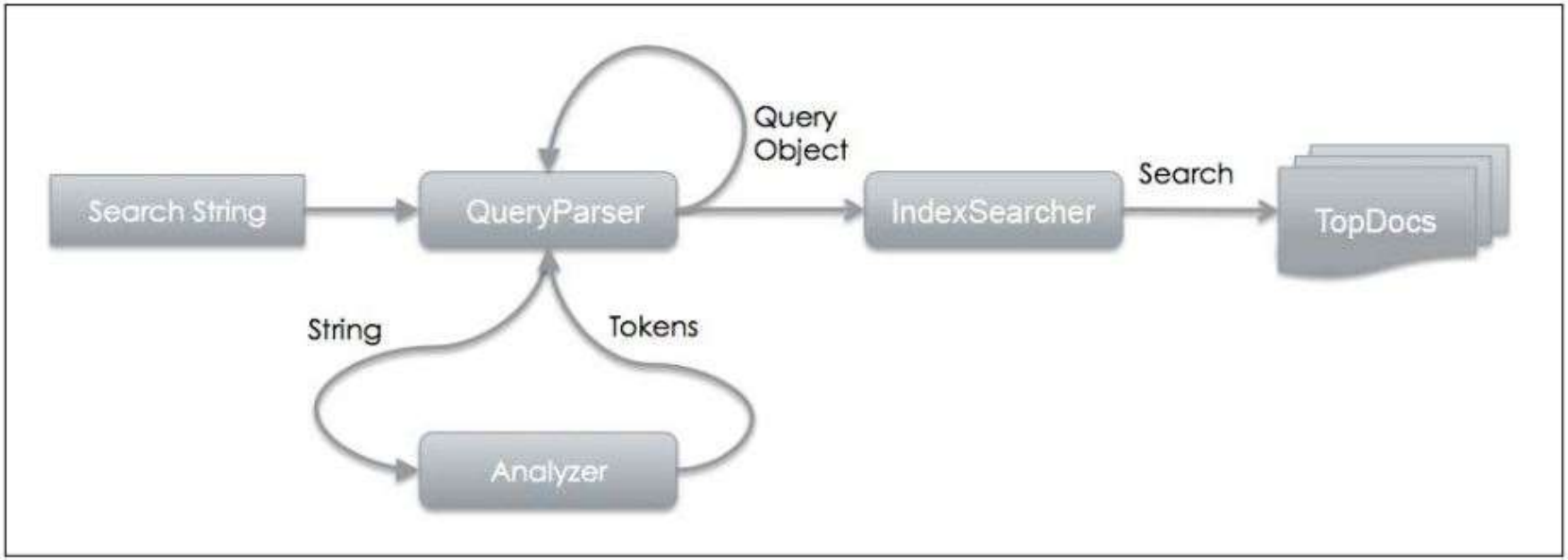
- Breaks a string down into a stream of terms or tokens.
- A simple tokenizer might split the string up into terms wherever it encounters whitespace or punctuation.
  - Standard Tokenizer
  - Edge NGram Tokenizer
  - Keyword Tokenizer
  - Letter Tokenizer
  - Lowercase Tokenizer
  - NGram Tokenizer
  - Whitespace Tokenizer
  - Pattern Tokenizer
  - UAX Email URL Tokenizer
  - Path Hierarchy Tokenizer
  - Classic Tokenizer
  - Thai Tokenizer

Token Filter

- Accepts a stream of tokens from a tokenizer and can:
  - modify tokens (eg lowercasing)
  - delete tokens (eg remove stopwords)
  - add tokens (eg synonyms)

- Standard Token Filter
- ASCII Folding Token Filter
- Length Token Filter
- Lowercase Token Filter
- Uppercase Token Filter
- NGram Token Filter
- Edge NGram Token Filter
- Porter Stem Token Filter
- Shingle Token Filter
- Stop Token Filter
- Word Delimiter Token Filter
- Stemmer Token Filter
- Stemmer Override Token Filter
- Keyword Marker Token Filter
- Keyword Repeat Token Filter
- KStem Token Filter
- Snowball Token Filter
- Phonetic Token Filter
- Synonym Token Filter
- Compound Word Token Filter
- Reverse Token Filter
- Elision Token Filter
- Truncate Token Filter
- Unique Token Filter
- Pattern Capture Token Filter
- Pattern Replace Token Filter
- Trim Token Filter
- Limit Token Count Token Filter
- Hunspell Token Filter
- Common Grams Token Filter
- Normalization Token Filter
- CJK Width Token Filter
- CJK Bigram Token Filter
- Delimited Payload Token Filter
- Keep Words Token Filter
- Keep Types Token Filter
- Classic Token Filter
- Apostrophe Token Filter
- Decimal Digit Token Filter

Searching with Lucene



Source: Edwood Ng, Vineeth Mohan, Lucene 4 Cookbook, Packt Publishing, 2015

Core searching classes

- **`IndexSearcher`**
  - Central class that exposes several search methods on an index
  - Accessed via an **`IndexReader`**
- **`Query`**
  - Abstract query class. Concrete subclasses represent specific types of queries, e.g., matching terms in fields, boolean queries, phrase queries, …
- **`QueryParser`**
  - Parses a textual representation of a query into a **`Query`** instance
- **`TopDocs`**
  - Contains references to the top documents returned by a search
- **`ScoreDoc`**
  - Represents a single search result

IndexSearcher

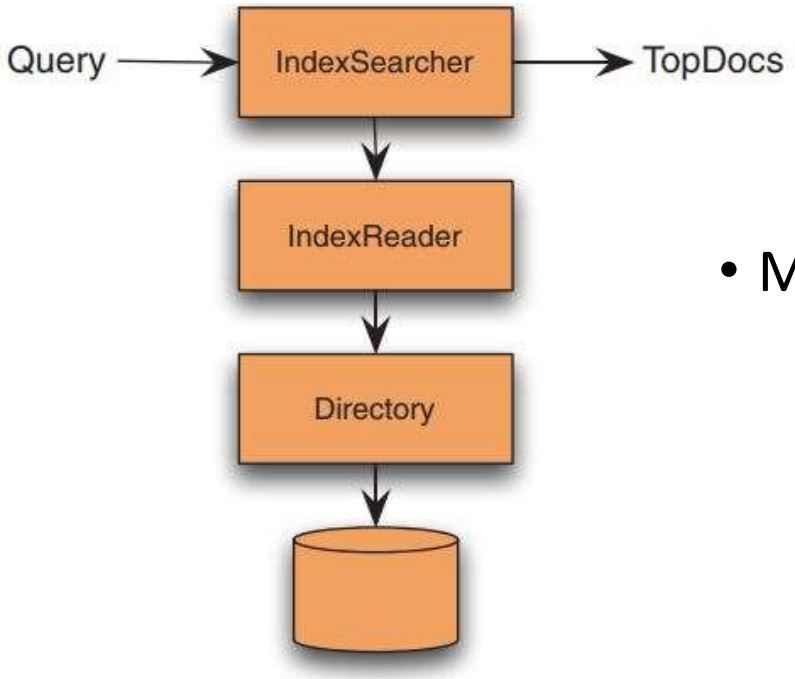


Figure 3.2 The relationship between the common classes used for searching

- Methods
  - `TopDocs search(Query q, int n);`
  - `Document doc(int docID);`

QueryParser

- Constructor
  - `QueryParser(String defaultField, Analyzer analyzer);`
- Parsing methods
  - `Query parse(String query) throws ParseException;`
  - ... and many more

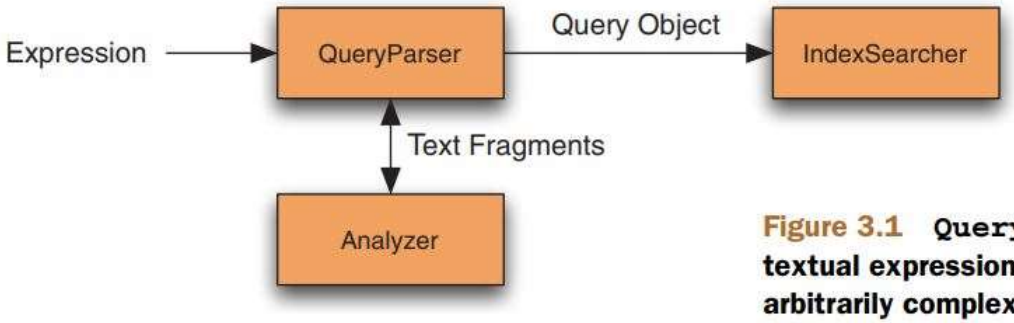


Figure 3.1 **`QueryParser` translates a textual expression from the end user into an arbitrarily complex query for searching.**

Nested Query with QueryParser's grouping

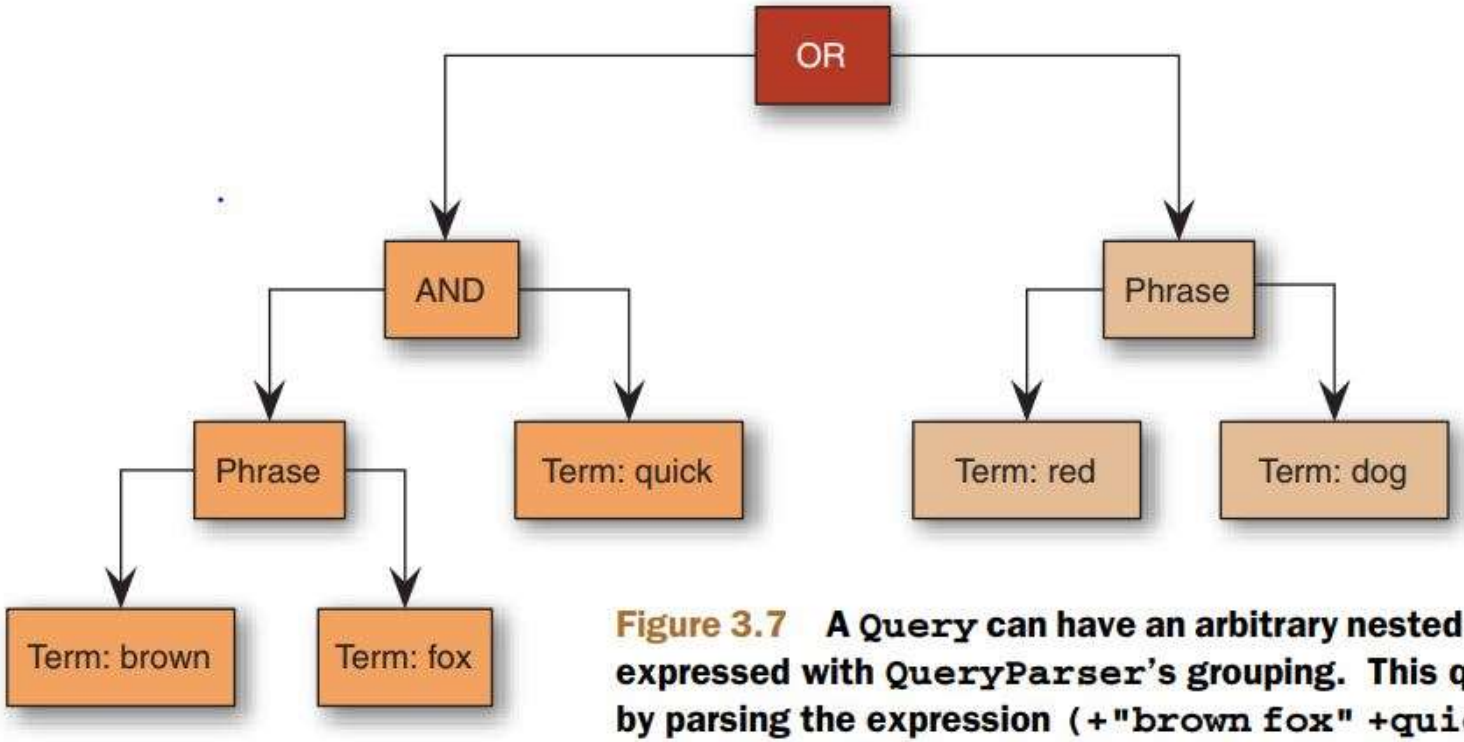


Figure 3.7 A `Query` can have an arbitrary nested structure, easily expressed with `QueryParser`'s grouping. This query is achieved by parsing the expression `(+"brown fox" +quick) "red dog"`.

QueryParser syntax examples

Query expression	Document matches if…
java	Contains the term java in the default field
java junit java OR junit	Contains the term java or junit or both in the default field (the default operator can be changed to AND)
+java +junit java AND junit	Contains both java and junit in the default field
title:ant	Contains the term ant in the title field
title:extreme –subject:sports	Contains extreme in the title and not sports in subject
(agile OR extreme) AND java	Boolean expression matches
title:”junit in action”	Phrase matches in title
title:”junit action”~5	Proximity matches (within 5) in title
java*	Wildcard matches
java~	Fuzzy matches
lastmodified:[1/1/09 TO 12/31/09]	Range matches

Construct Queries programmatically

- `TermQuery`
  - Constructed from a `Term`
- `TermRangeQuery`
- `NumericRangeQuery`
- `PrefixQuery`
- `BooleanQuery`
- `PhraseQuery`
- `WildcardQuery`
- `FuzzyQuery`
- `MatchAllDocsQuery`

Options for Constructing Lucene Queries

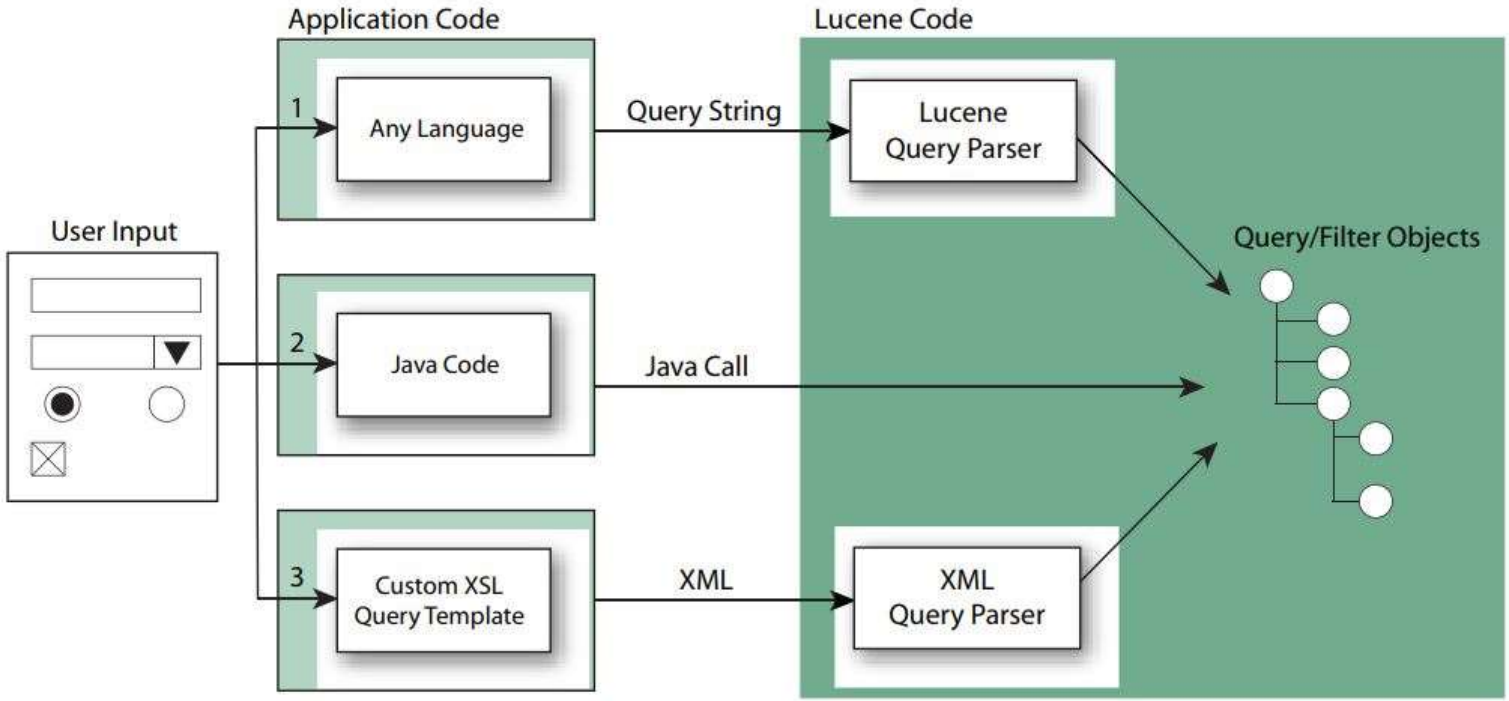


Figure 9.3 **Three common options for building a Lucene query from a search UI**

TopDocs and ScoreDoc

- `TopDocs` methods
  - Number of documents that matched the search
    `totalHits`
  - Array of `ScoreDoc` instances containing results
    `scoreDocs`
  - Returns best score of all matches
    `getMaxScore()`
- `ScoreDoc` methods
  - Document id
    `doc`
  - Document score
    `score`

Scoring

- Original scoring function uses basic tf-idf scoring with
  - Programmable boost values for certain fields in documents
  - Length normalization
  - Boosts for documents containing more of the query terms
- **`IndexSearcher`** provides an **`explain`**() method that explains the scoring of a document

Lucene 8.8.0 Scoring

- As well as traditional **tf.idf vector space model**, Lucene 8.8.0 has:
  - **Boolean** (**BooleanSimilarity**)
  - **BM25** (**BM25Similarity**)
  - **DRF** - Divergence from Randomness (**DFRSimilarity**)
  - **DFI** - *Divergence from Independence* (**DFISimilarity**)
  - **IB** - information (theory)-based similarity (**IBSimilarity**)
  - **LM** – language Model based similarity (**LMDirichletSimilarity**, **LMJelinekMercerSimilarity**)
  - Combining evidence from multiple similarity (**MultiSimilarity**)

```
Option 1:  indexSearcher.setSimilarity(new BM25Similarity());
Option 2:  BM25Similarity custom = new BM25Similarity(k1, b);
           indexSearcher.setSimilarity(custom);
```

Summary

Lucene in a search system