

Information Access with Apache Lucene - Part 2

Metodi per il Ritrovamento dell'Informazione

Laurea Triennale in Informatica
Università degli Studi di Bari Aldo Moro

Prof. Cataldo Musto

cataldo.musto@uniba.it

Code Repository & Requirements

Code repository

https://github.com/swapUnibaAcademy/MRI_Lab



Requirements

- Java SDK 1.8+ <https://www.java.com/en/download/>
- IDE: NetBeans, IntelliJ, Eclipse, ...
- **Maven:**
<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

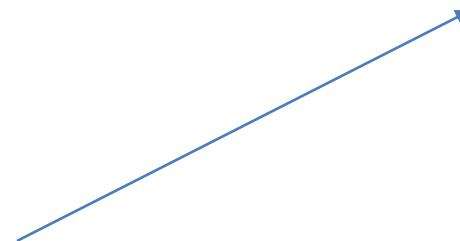


<http://lucene.apache.org>



APACHE LUCENE

Documentation



https://lucene.apache.org/core/10_3_1/index.html



Apache Lucene

Scan for the Doc

- Apache Software Foundation project
 - <http://lucene.apache.org>
- What Lucene is
 - Search library: indexing and searching Application Programming Interface (API)
- What Lucene is not
 - Search engine (no crawling, server, user interface, etc.)

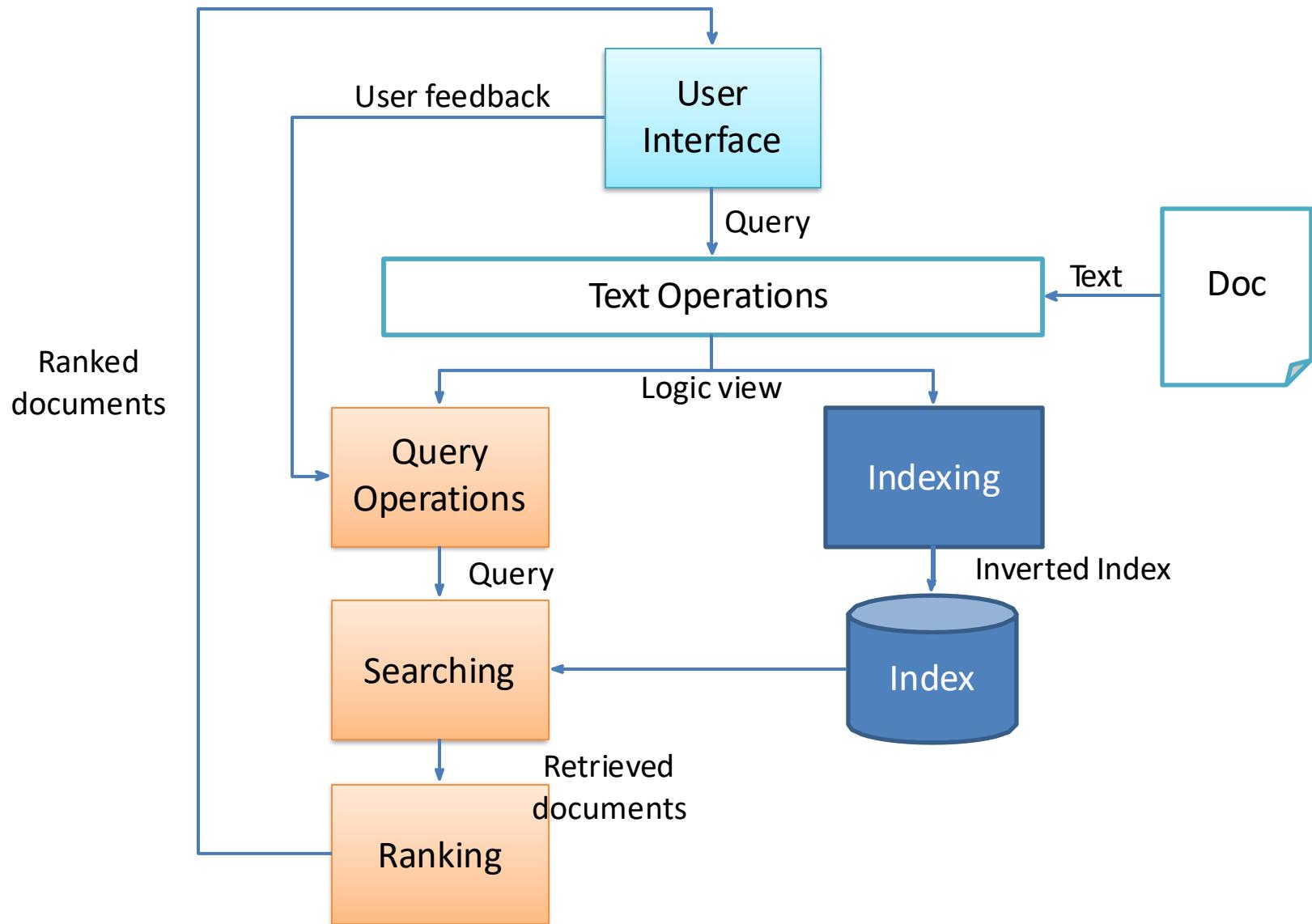


Lucene

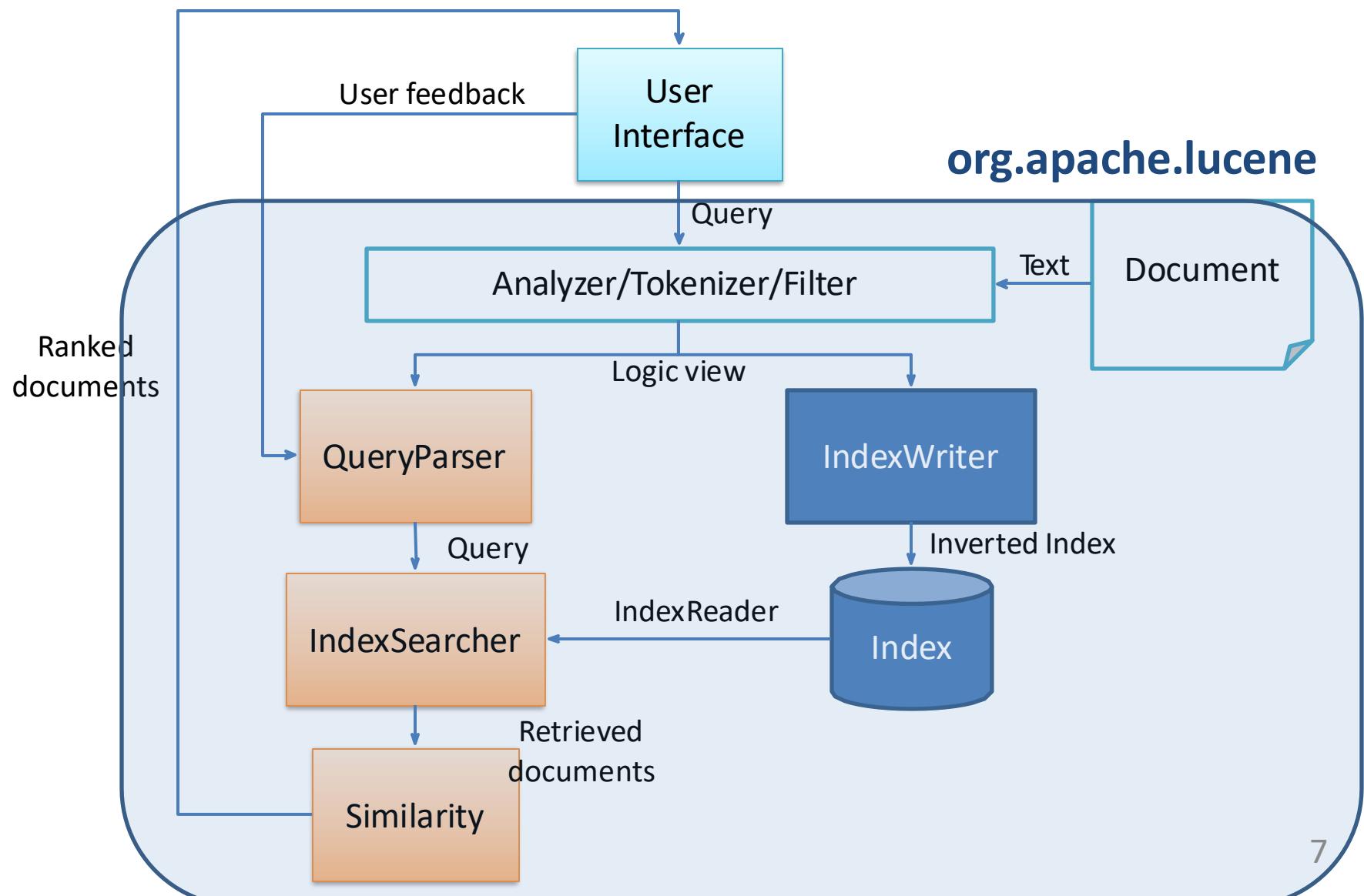
Scan for the Doc

- Full-text search
- High performance
- Scalable
- Cross-platform
- 100%-pure Java
- Porting in other languages:
 - Perl, C#, C++, Python, Ruby e PHP

Information Retrieval Process



From Concepts to Java Classes



Document

- The main entities involved in a IR process
- A collection of *Fields* which give structure to the document
- A Field has three parts
 - Name
 - Type
 - Value
 - text (String, Reader or pre-analyzed TokenStream)
 - binary (byte[])
 - numeric (a Number)
- Each field needs to be added to the document
- A Field may be stored in the index (TYPE_STORED), so it may be returned with hits on the document.

Document: Fields Representation

<http://www.bbc.co.uk/news/technology-15365207>

NEWS TECHNOLOGY

Home | UK | Africa | Asia-Pac | Europe | Latin America | Mid-East | South Asia | US & Canada | Business | [More](#)

19 October 2011 Last updated at 11:55 GMT

41 Share [Facebook](#) [Twitter](#) [Email](#)

Rory Cellan-Jones

Technology correspondent

[More from Rory](#) | [Follow Rory on Twitter](#)



Android serves up its Ice Cream Sandwich

[Comments \(62\)](#)

October could prove a key month in the smartphone wars. Last week saw the launch of the latest iPhone, next week we expect to see the first Nokia Windows Phone 7 handsets - and today Google has unveiled the latest version of its Android operating system.

At first glance, Ice Cream Sandwich looks as though it will set a new benchmark for what a clever phone should be able to do.

"Ice Cream Sandwich?" I hear non-Android users ask.

Each version of the operating system is named after a popular dessert, so you have to get used to bizarre statements like "People are at the heart of Ice Cream Sandwich", as Google's blog said this morning.

The delights of ice cream, and the Samsung Galaxy Nexus phone designed to showcase the system, were unveiled at an event in Hong Kong, and there's a [YouTube video showing what it's about](#).



The new system uses facial recognition to unlock the traditional passcode to unlock the

Comments

[Sign in or register to comment and rate comments.](#)

All posts are [reactively-moderated](#) and must obey the [house rules](#).

All Comments (62)

Order by: [Oldest First](#) [Highest Rated](#) [Lowest Rated](#)

1. GrahamZ

19TH OCTOBER 2011 - 13:14

At Last !!



+8

Can't wait for it to be rolled out across the majority of handsets (ultimately unifying them). Hopefully it won't be too long before I can update my Galaxy....

Document: Fields Representation

<http://www.bbc.co.uk/news/technology-15365207>

NEWS TECHNOLOGY

Home | UK | Africa | Asia-Pac | Europe | Latin America

19 October 2011 Last updated at 11:55 GMT

Rory Cellan-Jones

Technology correspondent

More from Rory | Follow Rory on Twitter

Android serves up its Ice Cream Sandwich

COMMMENTS (62)

October could prove a key month in the smartphone wars. Last week saw the launch of the latest iPhone, next week we expect to see the first Nokia Windows Phone 7 handsets - and today Google has unveiled the latest version of its Android operating system.

At first glance, Ice Cream Sandwich looks as though it will set a new benchmark for what a clever phone should be able to do.

"Ice Cream Sandwich?" I hear non-Android users ask.

Each version of the operating system is named after a popular dessert, so you have to get used to bizarre statements like "People are at the heart of Ice Cream Sandwich", as Google's blog said this morning.

The delights of ice cream, and the Samsung Galaxy Nexus phone designed to showcase the system, were unveiled at an event in Hong Kong, and there's a YouTube video showing what it's about.

Content:

Date:

URL:

Authors:

Title:

Abstract:



Comments

Sign in or register to comment and rate comments.

All posts are reactively-moderated and must obey the house rules.

All Comments (62)

Order by: Oldest First Highest Rated Lowest Rated

1. GrahamZ

19TH OCTOBER 2011 - 13:14

At Last !!

Can't wait for it to be rolled out across the majority of handsets (ultimately unifying them). Hopefully it won't be too long before I can update my Galaxy....

Comments:

Document: Fields Representation

<http://www.bbc.co.uk/news/technology-15365207>

URL: StringField

NEWS TECHNOLOGY

Home | UK | Africa | Asia-Pac | Europe | Latin America

19 October 2011 Last updated at 11:55 GMT

Date: StringField (see DateTools)

Rory Cellan-Jones

Technology correspondent

More from Rory | Follow Rory on Twitter

Authors: TextField

Android serves up its Ice Cream Sandwich

Title: TextField

October could prove a key month in the smartphone wars. Last week saw the launch of the latest iPhone, next week we expect to see the first Nokia Windows Phone 7 handsets - and today Google has unveiled the latest version of its Android operating system.

At first glance, Ice Cream Sandwich looks as though it will set a new benchmark for what a clever phone should be able to do.

"Ice Cream Sandwich?" I hear non-Android users ask.

Each version of the operating system is named after a popular dessert, so you have to get used to bizarre statements like "People are at the heart of Ice Cream Sandwich", as Google's blog said this morning.

The delights of ice cream, and the Samsung Galaxy Nexus phone designed to showcase the system, were unveiled at an event in Hong Kong, and there's a YouTube video showing what it's about.



Comments

Sign in or register to comment and rate comments.

All posts are reactively-moderated and must obey the house rules.

All Comments (62)

Order by: Oldest First ▾ Highest Rated ▾ Lowest Rated

1. GrahamZ

19TH OCTOBER 2011 - 13:14

At Last !!

Can't wait for it to be rolled out across the majority of handsets (ultimately unifying them). Hopefully it won't be too long before I can update my Galaxy....

Content: TextField

Comments: TextField

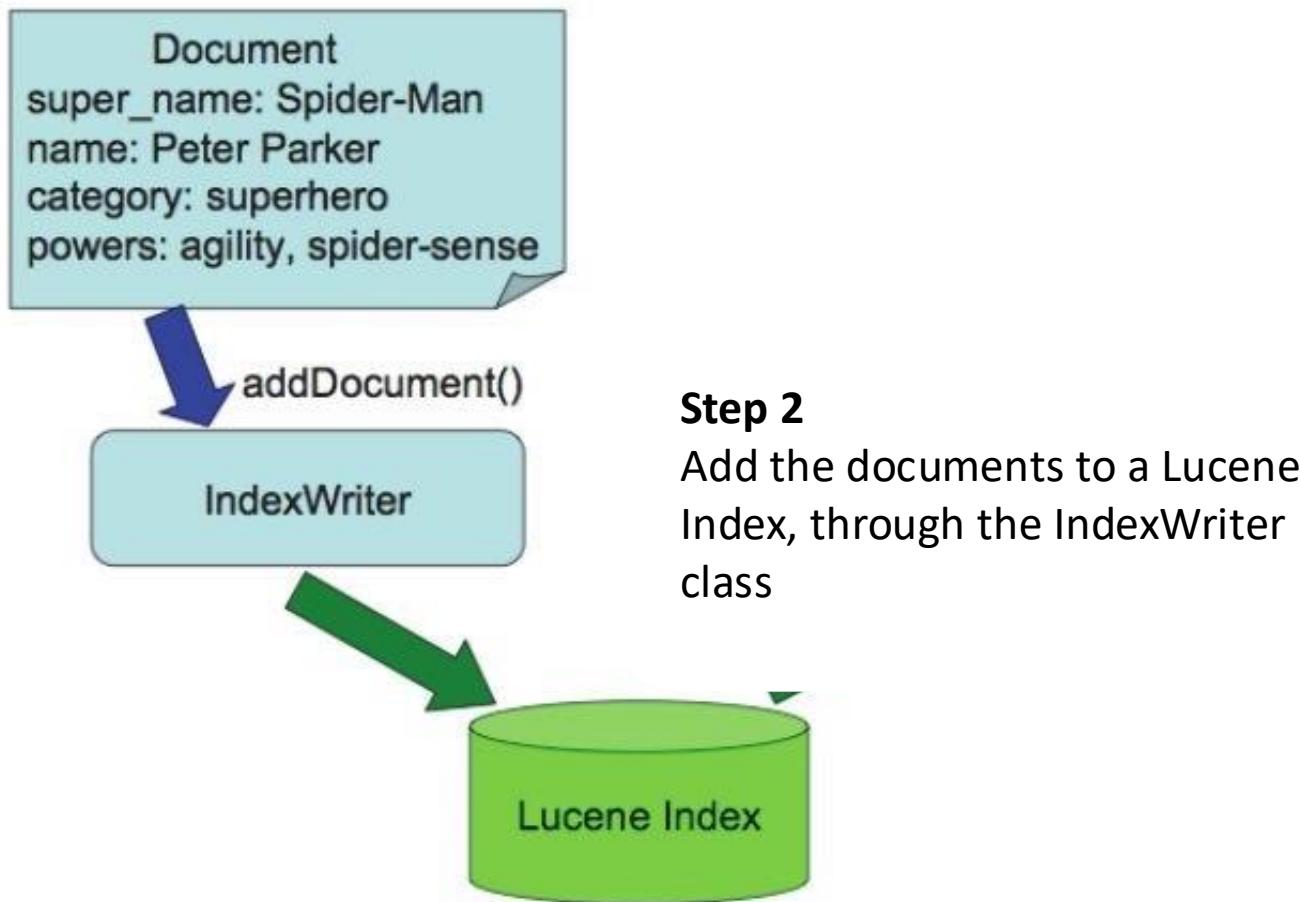
Overview of the Process

```
Document  
super_name: Spider-Man  
name: Peter Parker  
category: superhero  
powers: agility, spider-sense
```

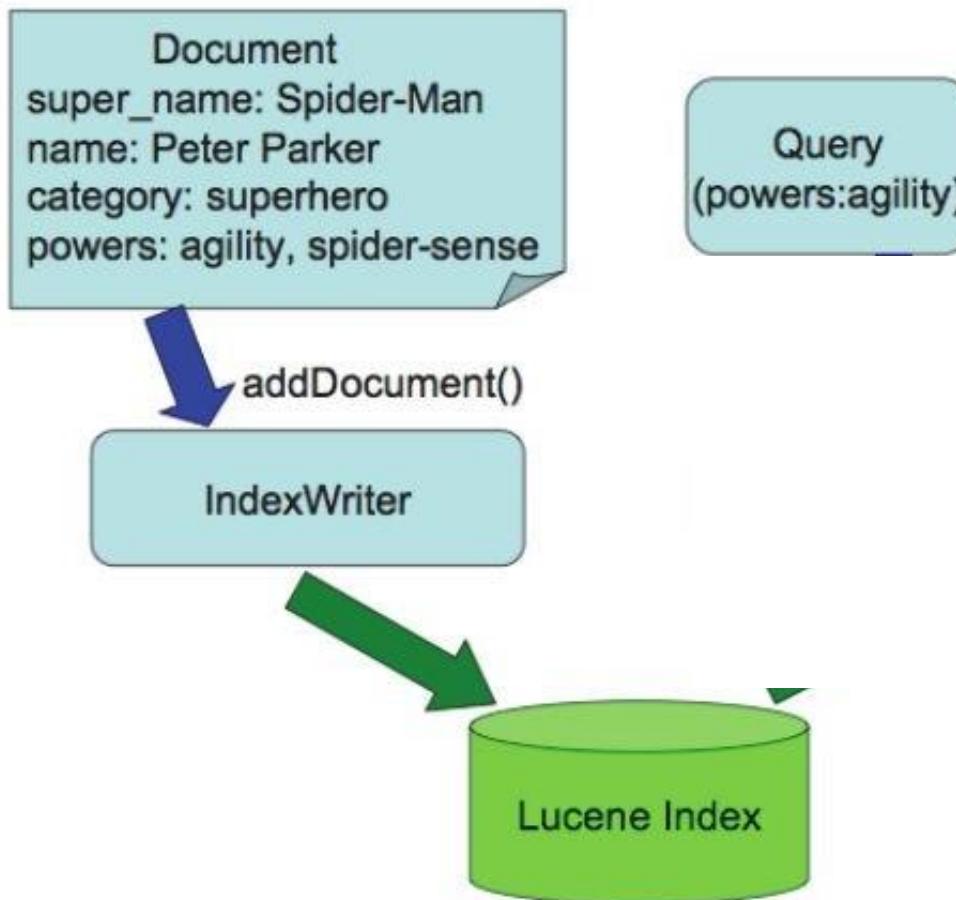
Step 1

Create a document and populate its fields

Overview of the Process



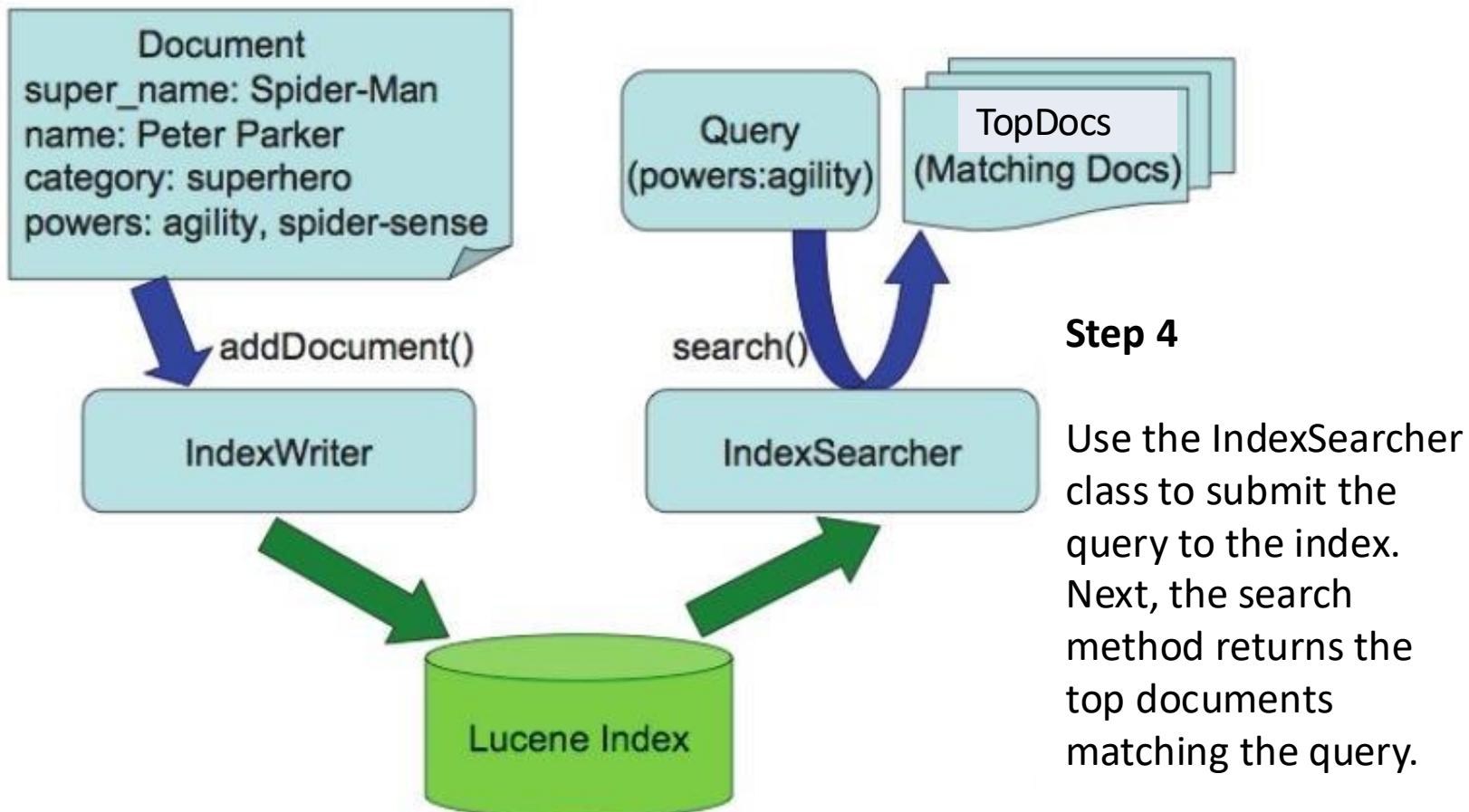
Overview of the Process



Step 3

Search the Index by defining a query and submit it to the index

Overview of the Process



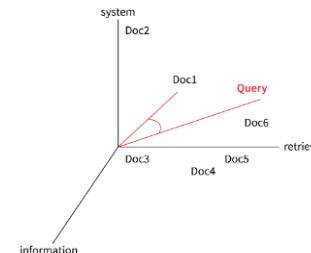
Use the **IndexSearcher** class to submit the query to the index. Next, the **search** method returns the top documents matching the query.

Lucene Model - Searcher

- Based on **Vector Space Model**

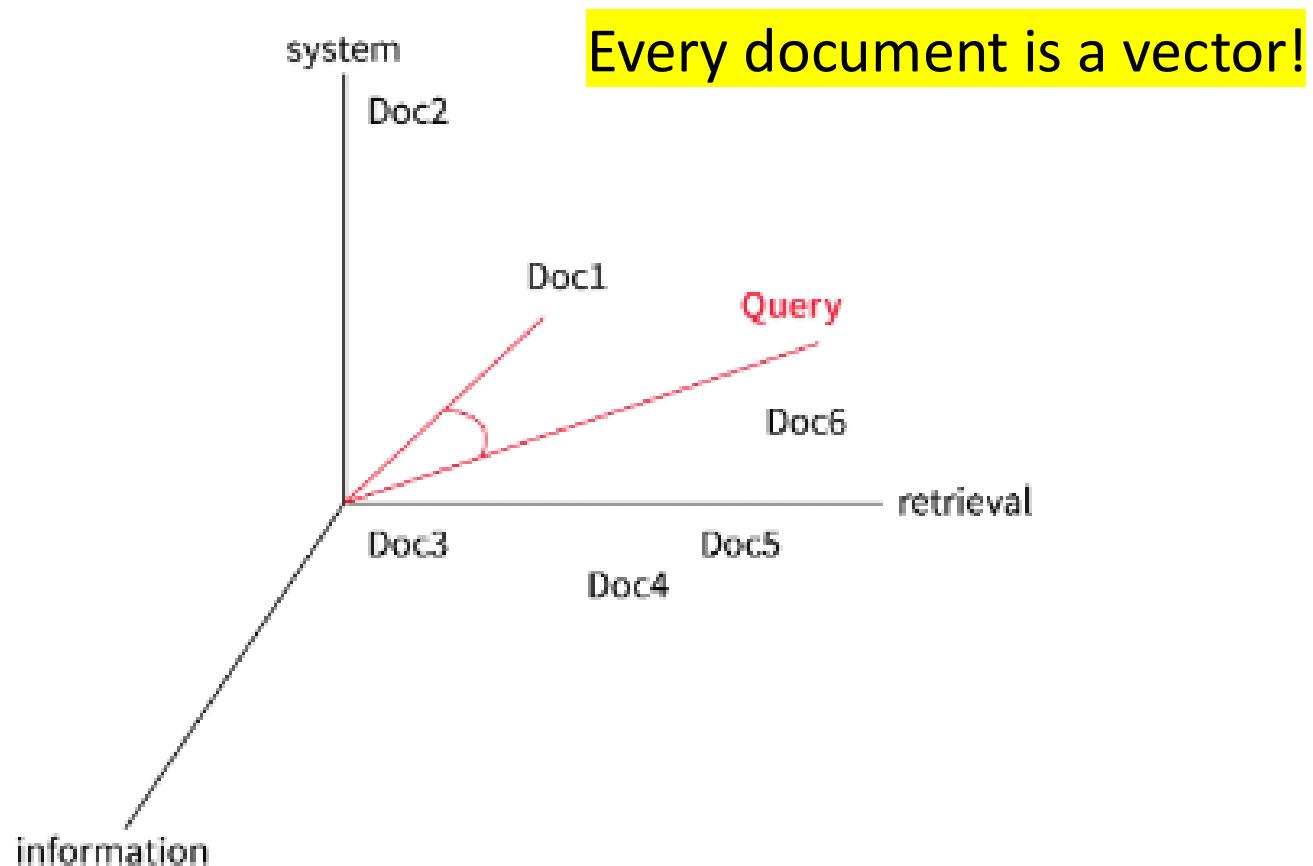
	D1	D2	D3	
Cheshire Cat	1	0	1	
Alice	2	2	2	
book	1	0	0	
King	1	1	0	
table	0	1	1	
Queen	0	1	1	
grin	0	0	2	

Every document is a vector!



Lucene Model - Searcher

- Based on **Vector Space Model**

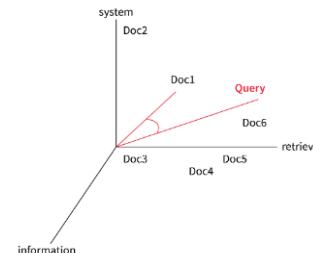


Lucene Model - Searcher

- Based on **Vector Space Model**

	D1	D2	D3	Q
Cheshire Cat	1	0	1	0
Alice	2	2	2	1
book	1	0	0	0
King	1	1	0	0
table	0	1	1	0
Queen	0	1	1	1
grin	0	0	2	0

The query is a vector as well



Lucene Model - Searcher

- Based on **Vector Space Model**
- **How is the relevance calculated in Lucene?**

Lucene Model - Searcher

- Based on **Vector Space Model**
- How is the relevance calculated in Lucene?
- Relevance is based on Okapi BM25 Scoring Function
 - https://en.wikipedia.org/wiki/Okapi_BM25
 - BM25 improves the classical similarity score based on TF-IDF by introducing two new parameters
 - One to handle the problem of **document length**
 - One to handle the «**saturation**» **problem** of TF

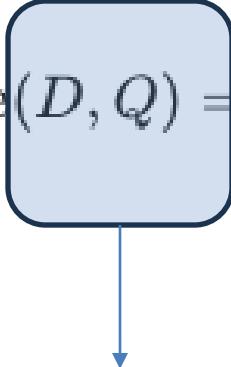
More info https://lucene.apache.org/core/10_3_1/core/org/apache/lucene/search/similarities/package-summary.html

Lucene Model - Searcher

$$\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

More info https://lucene.apache.org/core/10_3_1/core/org/apache/lucene/search/similarities/package-summary.html

Lucene Model - Searcher

$$\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$


(document, query)

More info https://lucene.apache.org/core/10_3_1/core/org/apache/lucene/search/similarities/package-summary.html

Lucene Model - Searcher

$$\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$


IDF score

More info https://lucene.apache.org/core/10_3_1/core/org/apache/lucene/search/similarities/package-summary.html

Lucene Model - Searcher

$$\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

The diagram illustrates the Lucene scoring formula. It shows the formula $\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$. The term $\text{IDF}(q_i)$ is highlighted with a blue rounded rectangle and has a blue arrow pointing down to the text "IDF score". The term $\text{tf}(q_i, D)$ is also highlighted with a blue rounded rectangle and has a blue arrow pointing up to the text "TF score".

More info https://lucene.apache.org/core/10_3_1/core/org/apache/lucene/search/similarities/package-summary.html

Lucene Model - Searcher

$$\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Saturation parameter Length normalization parameter

More info https://lucene.apache.org/core/10_3_1/core/org/apache/lucene/search/similarities/package-summary.html

Lucene Model - Searcher

$$\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Saturation parameter (k) reduces the importance of tf

- Low values = quick saturation (i.e., after a few occurrences, the importance of each new occurrence decreases)

Lucene Model - Searcher

$$\text{Score}(D, Q) = \sum_{i \in Q} \text{IDF}(q_i) \cdot \frac{\text{tf}(q_i, D) \cdot (k_1 + 1)}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b - b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

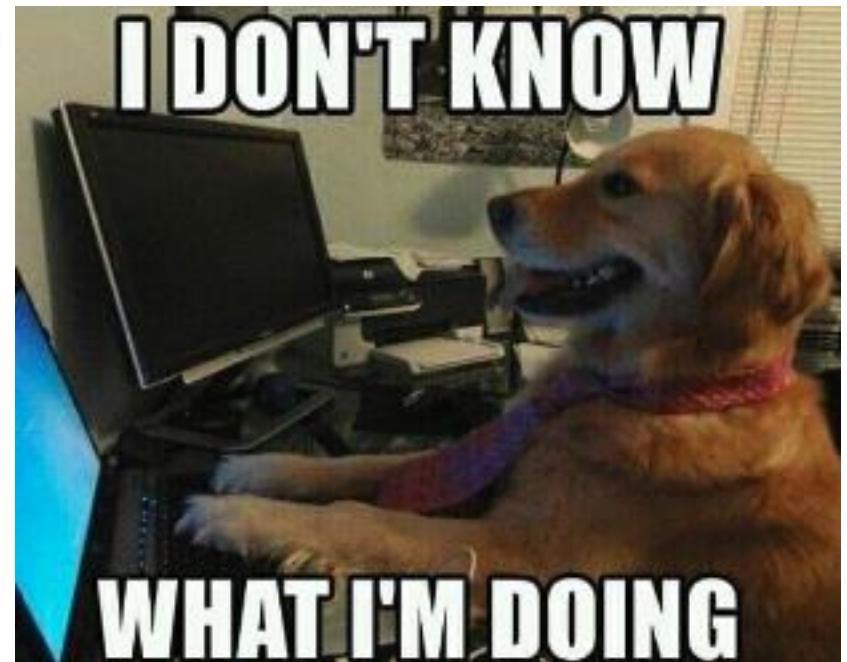
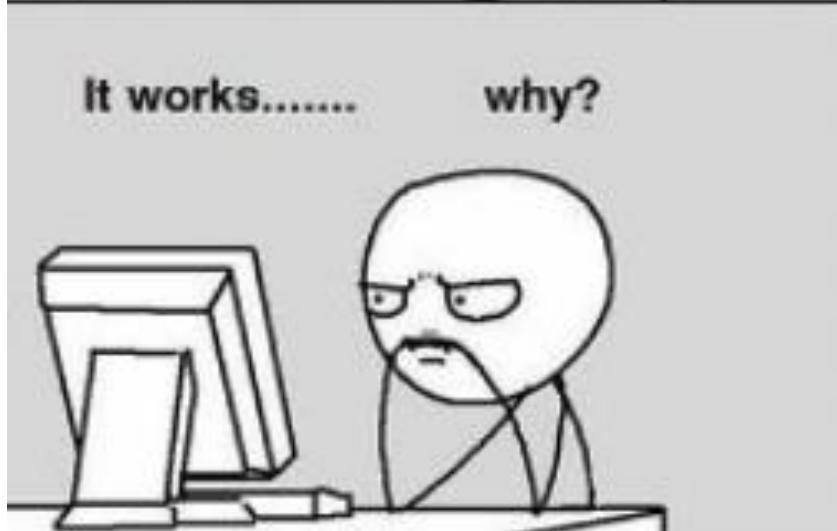
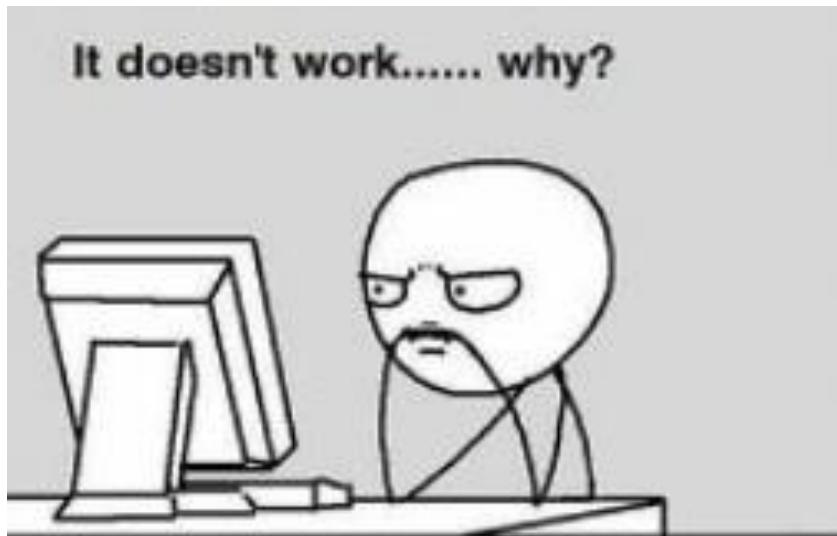
Saturation parameter (k) reduces the importance of tf

- Low values = quick saturation (i.e., after a few occurrences, the importance of each new occurrence decreases)

Normalization length parameter (b) weighs the importance of document length in the retrieval process

- High Values (max = 1) = the document length is highly considered to calculate the relevance

Let's Start



Let's start

1) Download the repository (you can use a GUI as well)

`git clone https://github.com/swapUnibaAcademy/MRI_Lab`

2) Create a new project

File → Open →

Import content from `./code/LuceneLab`

(`pom.xml` is there)

Let's start

Alternatively, you can directly clone the project

From IntelliJ IDEA:

File → New → Project from Version Control

Paste the GitHub repository URL.

Choose a destination folder.

Click **Clone**.

IntelliJ will clone the project and automatically try to detect Maven.

Hello World

See the class `uniba.mri.lucene.HelloWorld.java`

This class creates a **Document** object and populates its fields

Hello World

See the class `uniba.mri.lucene.HelloWorld.java`

```
//Open a directory from the file system (index directory)
FSDirectory fsdir = FSDirectory.open(new
File("./resources/documenti_news").toPath());  
  
//IndexWriter configuration
IndexWriterConfig iwc = new IndexWriterConfig(new
StandardAnalyzer());  
  
//Index directory is created if not exists or
overwritten
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE)  
  
//Create IndexWriter
IndexWriter writer = new IndexWriter(fsdir, iwc);
```

Hello World

See the class `uniba.mri.lucene.HelloWorld.java`

```
//Open a directory from the file system (index directory)
FSDirectory fsdir = FSDirectory.open(new
File("./resources/documenti_news").toPath());  
  
//IndexWriter configuration
IndexWriterConfig iwc = new IndexWriterConfig(new
StandardAnalyzer());  
  
//Index directory is created if not exists or
overwritten
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE)  
  
//Create IndexWriter
IndexWriter writer = new IndexWriter(fsdir, iwc);
```

Si può creare in CREATE o
APPEND

Hello World

See the class `uniba.mri.lucene.HelloWorld.java`

```
//Create document and add fields
Document doc = new Document();

doc.add(new TextField("super_name", "Spider-Man", Field.Store.NO));
doc.add(new TextField("name", "Peter Parker", Field.Store.NO));
doc.add(new TextField("category", "superhero", Field.Store.NO));
doc.add(new TextField("powers", "agility, spider-sense",
                     Field.Store.NO));

//add document to index
writer.addDocument(doc);

//close IndexWriter
writer.close();
```

Hello World

See the class **uniba.mri.lucene.HelloWorld.java**

```
//Create document and add fields
Document doc = new Document();
doc.add(new TextField("super_name", "Spider-Man",
Field.Store.NO));
doc.add(new TextField("name", "Peter Parker", Field.Store.NO));
doc.add(new TextField("category", "superhero", Field.Store.NO));
doc.add(new TextField("powers", "agility, spider-sense",
Field.Store.NO));
//add document to index
writer.addDocument(doc);
//close IndexWriter
writer.close();
```

Può essere YES oppure NO.

Indica se vogliamo memorizzare il campo o meno. **Attenzione:** non significa che non possiamo fare ricerche su quel campo, significa che non possiamo più risalirne al contenuto.

Hello World

See the class `uniba.mri.lucene.HelloWorld.java`

```
//Create document and add fields
Document doc = new Document();
doc.add(new TextField("super_name", "Spider-Man",
Field.Store.NO));
doc.add(new TextField("name", "Peter Parker", Field.Store.NO));
doc.add(new TextField("category", "superhero", Field.Store.NO));
doc.add(new TextField("powers", "agility, spider-sense",
Field.Store.NO));

//add document to index
writer.addDocument(doc);
//close IndexWriter
writer.close();

// Crea un IndexReaderIndexReader reader
= DirectoryReader.open(sdir);
// Conta il numero di documenti attivi (non cancellati)
int numDocs = reader.numDocs();
System.out.println("Numero di documenti nell'indice: " + numDocs);36
```

Field Constructors

- `TextField(String name, String value, Field.Store store)`
 - **Descrizione:** Crea un campo di testo. Il valore del campo viene analizzato (tokenizzato) utilizzando l'analizzatore predefinito.
- `StringField(String name, String value, Field.Store store)`
 - **Descrizione:** Crea un campo di tipo stringa. Il valore non viene analizzato, ma viene indicizzato così com'è.
- `NumericDocValuesField(String name, long value)`
 - **Descrizione:** Crea un campo numerico (float o double). Il valore viene indicizzato, e puoi scegliere di memorizzarlo.
- `BinaryField(String name, byte[] value, Field.Store store)`
 - **Descrizione:** Crea un campo binario che può contenere dati in formato binario. Questo è utile per memorizzare file o dati non testuali.
- `Field(String name, String value, FieldType fieldType)`
 - **Descrizione:** Crea un campo generico che può utilizzare un tipo di campo personalizzato definito da `FieldType`.

Field Constructors (ex.)

```
// Aggiungi un campo testo
doc.add(new TextField("title", "Il Signore degli Anelli",
Field.Store.YES));
```



```
// Aggiungi un campo di stringa (non memorizzato)
doc.add(new StringField("author", "J.R.R. Tolkien",
Field.Store.NO));
```



```
// Aggiungi un campo numerico
doc.add(new NumericField("publicationYear", 1954,
Field.Store.YES));
```

Adding documents to an Index

- `addDocument(Iterable<? extends IndexableField> doc)`
Adds a document to this index.
- `addDocuments(Iterable<? extends Iterable<? extends IndexableField>> docs)`
Atomically adds a block of documents with sequentially assigned document IDs, such that an external reader will see all or none of the documents.

Add Multiple Docs

```
List<Document> docs = new ArrayList<>();  
  
Document doc1 = new Document();  
doc1.add(new TextField("title", "Alice in Wonderland", Field.Store.YES));  
doc1.add(new TextField("content", "A girl falls into a rabbit hole.", Field.Store.YES));  
docs.add(doc1);  
  
Document doc2 = new Document();  
doc2.add(new TextField("title", "Through the Looking-Glass", Field.Store.YES));  
doc2.add(new TextField("content", "Alice steps through a mirror.", Field.Store.YES));  
docs.add(doc2);  
  
Document doc3 = new Document();  
doc3.add(new TextField("title", "Alice in Quantumland", Field.Store.YES));  
doc3.add(new TextField("content", "A journey through the quantum world.",  
Field.Store.YES));  
docs.add(doc3);  
  
// aggiunge tutti i documenti in un colpo solo  
writer.addDocuments(docs);  
  
// forza il commit dei dati su disco  
writer.commit();  
  
System.out.println(docs.size() + " documenti aggiunti all'indice in " + indexPath); 40
```

Add Multiple Docs

```
List<Document> docs = new ArrayList<>();
```

We store the documents in a ArrayList

```
Document doc1 = new Document();
doc1.add(new TextField("title", "Alice in Wonderland", Field.Store.YES));
doc1.add(new TextField("content", "A girl falls into a rabbit hole.", Field.Store.YES));
docs.add(doc1);
```

```
Document doc2 = new Document();
doc2.add(new TextField("title", "Through the Looking-Glass", Field.Store.YES));
doc2.add(new TextField("content", "Alice steps through a mirror.", Field.Store.YES));
docs.add(doc2);
```

```
Document doc3 = new Document();
doc3.add(new TextField("title", "Alice in Quantumland", Field.Store.YES));
doc3.add(new TextField("content", "A journey through the quantum world.",
Field.Store.YES));
docs.add(doc3);
```

```
// aggiunge tutti i documenti in un colpo solo
writer.addDocuments(docs);
```

We add all the documents at once

```
// forza il commit dei dati su disco
writer.commit();
```

```
System.out.println(docs.size() + " documenti aggiunti all'indice in " + indexPath); 41
```

Preliminary Exercise

- Extend **helloworld.java**
- Add more documents to the index (with different content)
- Print the number of documents in the index, to check that everything is ok

Search the Index

See the class `uniba.mri.lucene.SearchIndex.java`

In this example, we search a Lucene Index

Classes to be used:

- **IndexSearcher**
 - Indicates the index to be searched
- **QueryParser**
 - Indicates the characteristics of the query (i.e., on which field)
- **Query**
 - Indicates the content of the query

Search the Index

See the class **uniba.mri.lucene.SearchIndex.java**

```
//Create the IndexSearcher
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));
//Create the query parser with the default field and analyzer
QueryParser qp = new QueryParser("name", new StandardAnalyzer());
//Parse the query
Query q = qp.parse("parker");

//Search
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

Search the Index

See the class `uniba.mri.lucene.SearchIndex.java`

```
//Create the IndexSearcher  
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));  
//Create the query parser with the default field and analyzer  
QueryParser qp = new QueryParser("name", new StandardAnalyzer());  
//Parse the query  
Query q = qp.parse("parker");  
//Search  
TopDocs topdocs = searcher.search(q, 10);  
System.out.println("Found " + topdocs.totalHits.value + " document(s).")
```

Indichiamo il campo su cui effettuare la ricerca e il contenuto della query

Query Parser (ex.)

Utilizzando **MultiFieldQueryParser()** si possono interrogare più campi in parallelo.

```
// Inizializza i nomi dei campi che vuoi interrogare
String[] fields = {"name", "category", "powers"}; ←

// Creazione di MultiFieldQueryParser
MultiFieldQueryParser parser = new
    MultiFieldQueryParser(fields, analyzer); ←

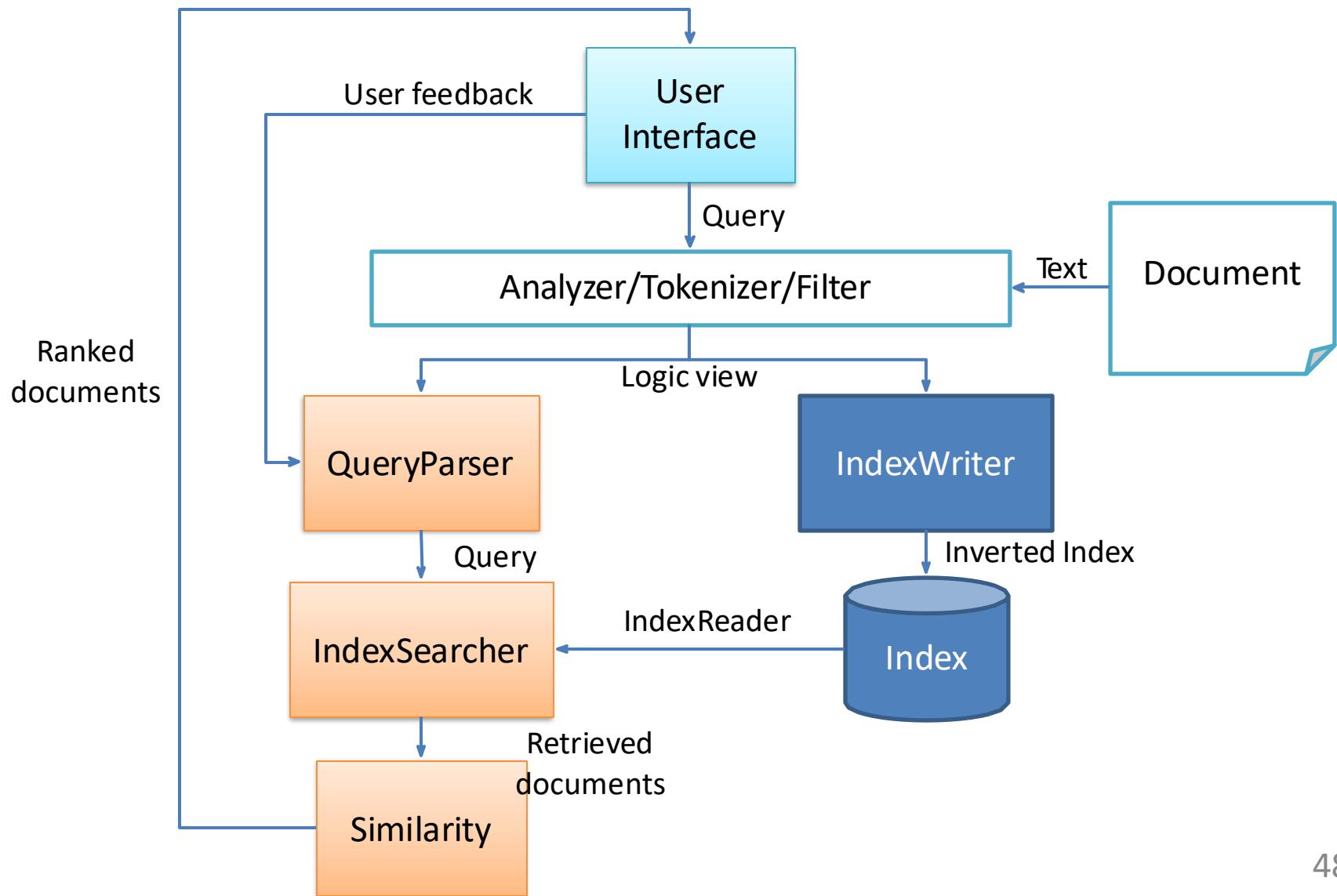
// Definisci la query
String queryString = "parker";
Query query = parser.parse(queryString);

// Esegue la query
IndexSearcher searcher = new
    IndexSearcher(DirectoryReader.open(directory));
TopDocs results = searcher.search(query, 10);
```

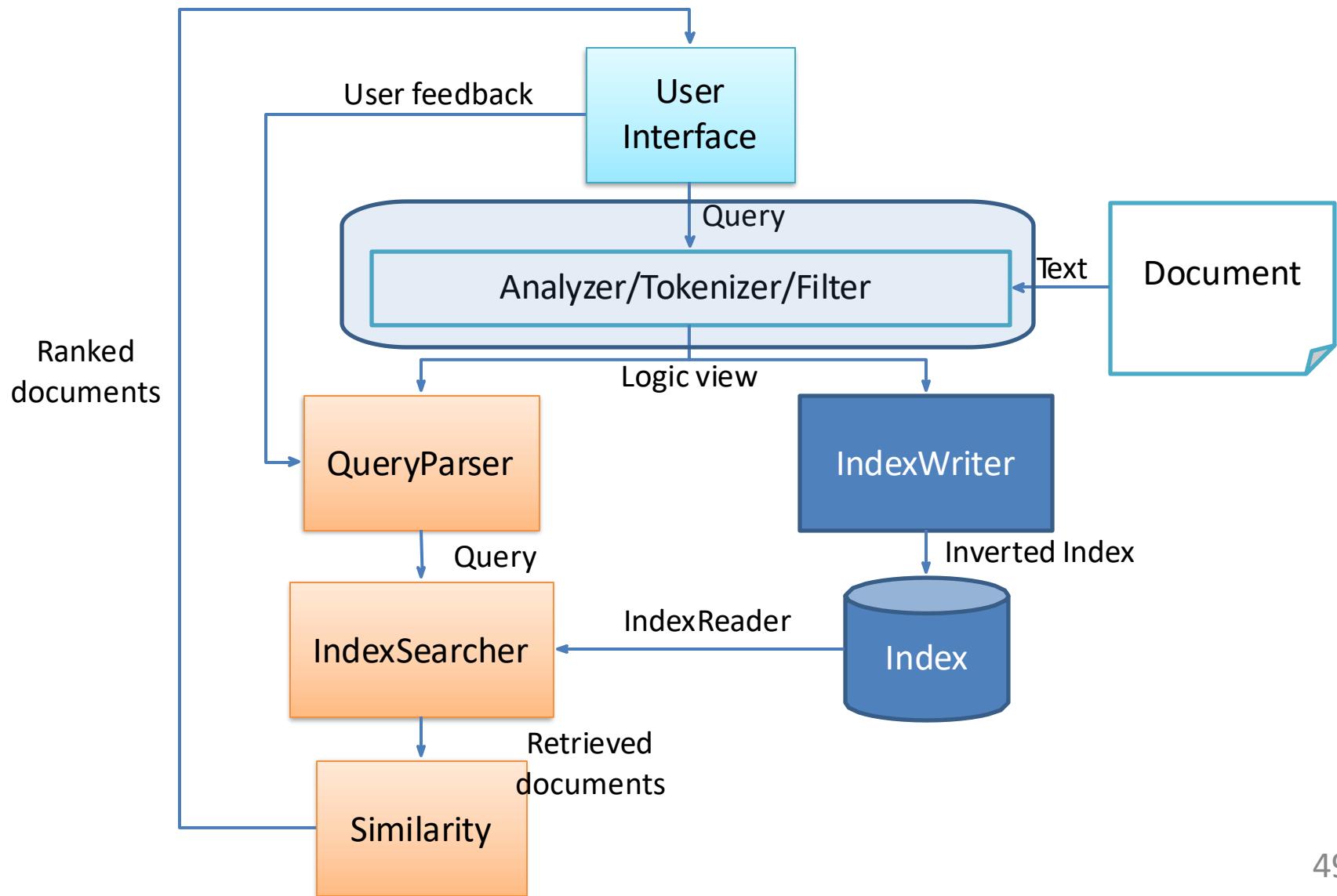
Exercise 0

- Modify **searchIndex.java**
- Create a new index (define the **fields** and the **content** you prefer)
 - Be realistic, figure out a real use case (i.e., sport news)
- Check the number of documents in the index
- Try some queries, and check if the number of documents matching the query is correct

Recap: Lucene Architecture



Let's focus on Analyzers



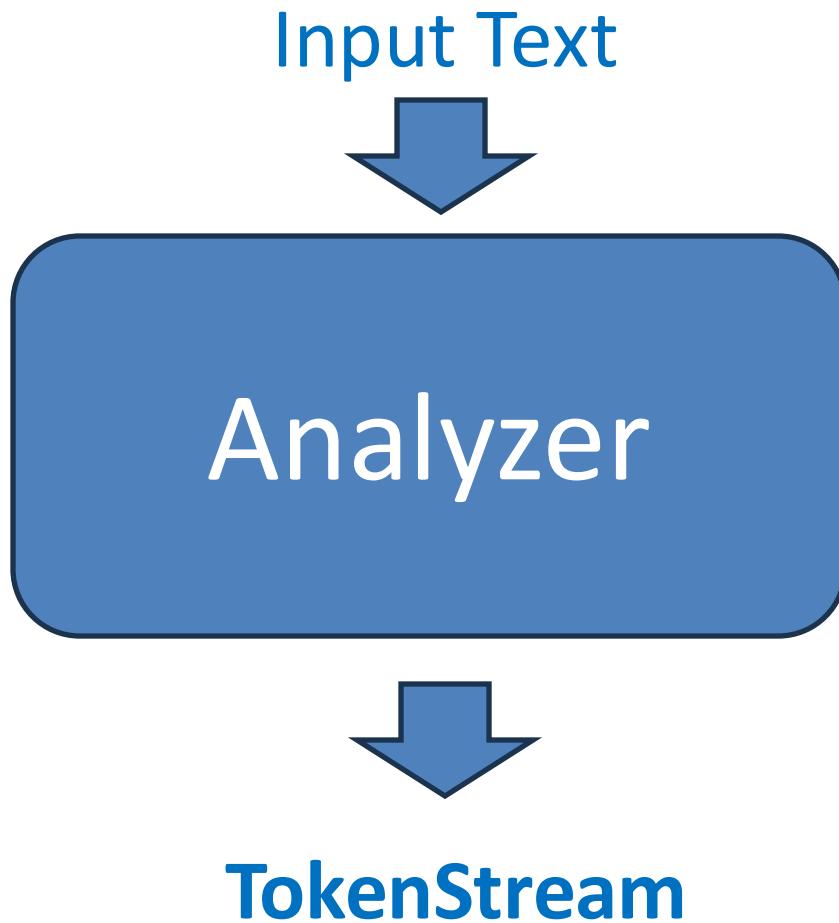
Text Analysis

- **Main Operations**
 - **Tokenization**
 - split text in token
 - **Stop word elimination**
 - remove common words
 - **Stemming**
 - reducing inflected (or sometimes derived) words to their stem

More info

https://lucene.apache.org/core/10_3_1/analysis/common/index.html

Text Analysis



Text Analysis

- **Analyzer:** encapsulates the analysis process
 - **Tokenize a text** by performing any number of operations on it
 - Responsible for **building the TokenStream** consumed by the indexing and searching processes

We have already found it in the previous exercise

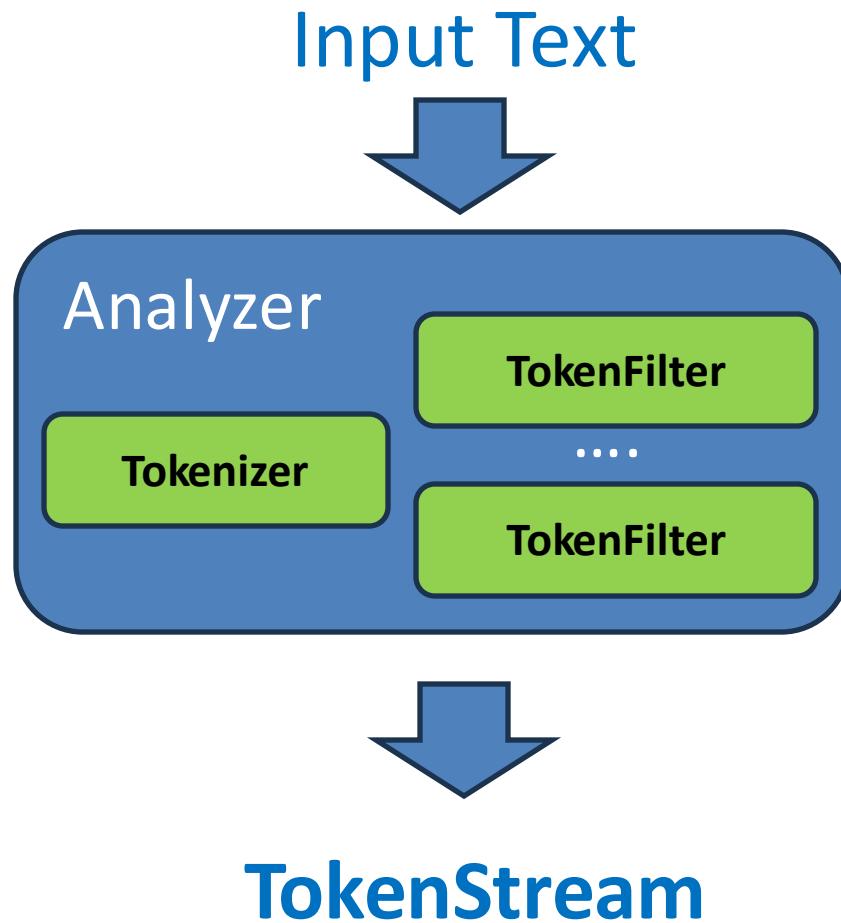
```
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
```

```
QueryParser qp = new QueryParser("name", new StandardAnalyzer());
```

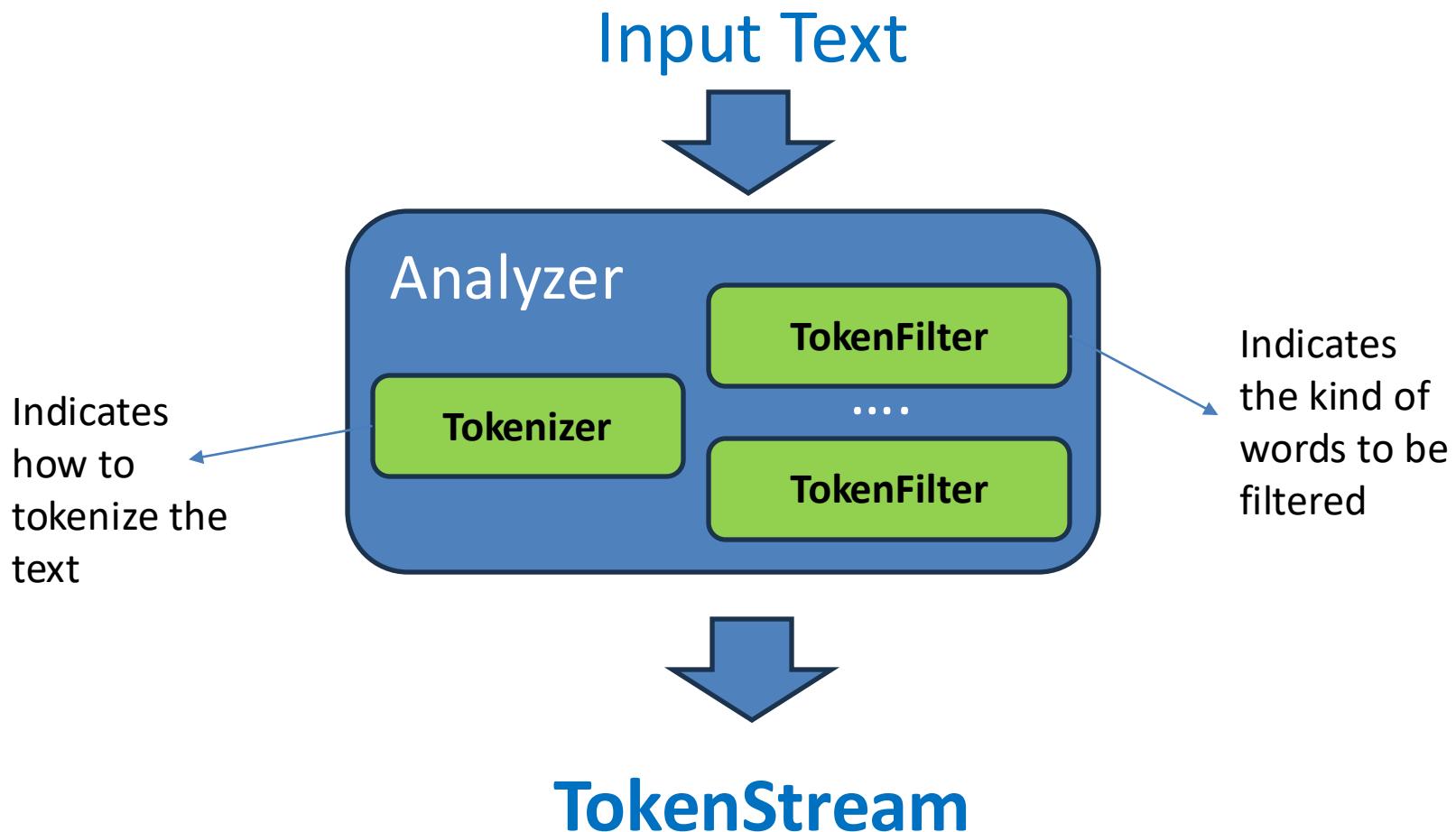
Text Analysis

- **Analyzer:** encapsulates the analysis process
 - **Tokenize a text** by performing any number of operations on it
 - Responsible for **building the TokenStream** consumed by the indexing and searching processes
- **Tokenizer:** is part of the TokenStream process
 - Responsible for breaking up incoming text into tokens
- **TokenFilter:** is also a TokenStream and is responsible for modifying tokens that have been created by the Tokenizer

Text Analysis



Text Analysis



Tokenizers

- **Tokenization** is the process of breaking input text into small indexing elements, e.g. tokens
- A Tokenizer is a TokenStream and is responsible for breaking up incoming text into tokens. Usually Analyzer will use a Tokenizer as the first step in the analysis process

source string: “full-text lucene.apache.org”

- **StandardTokenizer**
 - “full” “text” “lucene.apache.org”
- **WhitespaceTokenizer**
 - “full-text” “lucene.apache.org”
- **LetterTokenizer**
 - “full” “text” “lucene” “apache” “org”

TokenFilters

- A **TokenFilter** is also a TokenStream and is responsible for modifying tokens that have been created by the Tokenizer.
- **LowerCaseFilter**
- **StopFilter**
- **LengthFilter**
- **PorterStemFilter**
 - stemming: reducing words to root form
 - rides, ride, riding => ride
 - country, countries => countri

Analyzers

- **KeywordAnalyzer**
 - Returns a single token
- **WhitespaceAnalyzer**
 - Splits tokens at whitespace
- **SimpleAnalyzer**
 - Divides text at non letter characters and lowercases
- **StopAnalyzer**
 - Divides text at non letter characters, lowercases, and removes stop words
- **StandardAnalyzer**
 - Tokenizes based on sophisticated grammar that recognizes e-mail addresses, acronyms, etc.; lowercases and removes stop words (optional)

We do not need to precisely know how to implement **tokenizers** and **tokenfilters**. We need them if we want to customize our pipeline, otherwise standard analyzers are fine!

Analyzers

Input Text (ex.)

The quick brown fox jumped over the lazy dogs

- WhitespaceAnalyzer:
[The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]
- SimpleAnalyzer:
[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]
- StopAnalyzer:
[quick] [brown] [fox] [jumped] [lazy] [dogs]
- StandardAnalyzer:
[quick] [brown] [fox] [jumped] [over] [lazy] [dogs]

Analyzers

Input Text (ex.)

"XY&Z Corporation - xyz@example.com"

- WhitespaceAnalyzer:
[XY&Z] [Corporation] [-] [xyz@example.com]
- SimpleAnalyzer:
[xy] [z] [corporation] [xyz] [example] [com]
- StandardAnalyzer:
[xy&z] [corporation] [xyz@example.com]

Test Analyzer

See the class `uniba.mri.lucene.TestAnalyzer`

```
public static void main(String[] args) throws IOException {  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new WhitespaceAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new StandardAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new StandardAnalyzer(new  
FileReader("resources/en_stopword"))));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new EnglishAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new MyAnalyzer()));  
}
```

Test Analyzer

See the class **uniba.mri.lucene.TestAnalyzer**

Function that prints the token obtained by processing the string

```
public static void main(String[] args) throws IOException {  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new WhitespaceAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new StandardAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new StandardAnalyzer(new  
FileReader("resources/en_stopword"))));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new EnglishAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new MyAnalyzer()));  
}
```

Test Analyzer

See the class `uniba.mri.lucene.TestAnalyzer`

```
//Implementazione di getTokens()
public static List<String> getTokens(Reader reader, Analyzer analyzer) throws
IOException {
    List<String> tokens = new ArrayList<>();
    //Creiamo un flusso di token e lo inizializziamo
    TokenStream tokenStream = analyzer.tokenStream("text", reader);
    tokenStream.reset();
    //Istruzione per accedere al token corrente
    CharTermAttribute cattr = tokenStream.addAttribute(CharTermAttribute.class);
    while (tokenStream.incrementToken()) {
        String token = cattr.toString();
        tokens.add(token);
    }
    tokenStream.end();
    return tokens;
}
```

Test Analyzer

See the class `uniba.mri.lucene.TestAnalyzer`

```
//Implementazione di getTokens()
public static List<String> getTokens(Reader reader, Analyzer analyzer) throws
    IOException {
    ...
    return tokens;
}
```

La particolare
tipologia di **analyzer**
che utilizziamo è
responsabile di
determinare il flusso
di token

Test Analyzer

See the class `uniba.mri.lucene.TestAnalyzer`

```
//Implementazione di getTokens()
public static List<String> getTokens(Reader reader, Analyzer analyzer) throws
IOException {
    List<String> tokens = new ArrayList<>();
    //Creiamo un flusso di token e lo inizializziamo
    TokenStream tokenStream = analyzer.tokenStream("text", reader);
    tokenStream.reset();
    //Istruzione per accedere al token corrente
    CharTermAttribute cattr = tokenStream.addAttribute(CharTermAttribute.class);
    while (tokenStream.incrementToken()) {
        String token = cattr.toString();
        tokens.add(token);
    }
    tokenStream.end();
    return tokens;
}
```

La particolare tipologia di `analyzer` che utilizziamo è responsabile di determinare il flusso di token

Test Analyzer

See the class `uniba.mri.lucene.TestAnalyzer`

```
//Implementazione di getTokens()
public static List<String> getTokens(Reader reader, Analyzer analyzer) throws
IOException {
    List<String> tokens = new ArrayList<>();
    //Creiamo un flusso di token e lo inizializziamo
    TokenStream tokenStream = analyzer.tokenStream("text", reader);
    tokenStream.reset();
    //Istruzione per accedere al token corrente
    CharTermAttribute cattr = tokenStream.addAttribute(CharTermAttribute.class);
    while (tokenStream.incrementToken()) {
        String token = cattr.toString();
        tokens.add(token);
    }
    tokenStream.end();
    return tokens;
}
```

Ogni token non è una semplice stringa, è un insieme di attributi (viene memorizzata la posizione, l'offset, etc.)

Test Analyzer

See the class `uniba.mri.lucene.TestAnalyzer`

```
public static void main(String[] args) throws IOException {  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new WhitespaceAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new StandardAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new StandardAnalyzer(new  
FileReader("resources/en_stopword"))));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new EnglishAnalyzer()));  
  
    System.out.println(getTokens(new StringReader("the full-text indexing and  
search API: lucene.apache.org"), new MyAnalyzer()));  
}
```

This is a custom analyzer

Custom Analyzer

See the class `uniba.mri.lucene.MyAnalyzer`

`@Override`

```
protected TokenStreamComponents createComponents(String fieldName) {  
    Tokenizer source = new LetterTokenizer(); //divide parola per parola  
    TokenStream filter = new LowerCaseFilter(source); //minuscole  
    filter = new StopFilter(filter, STOP_WORDS); //nuovo filtro  
    return new TokenStreamComponents(source, filter);  
}
```

Custom Analyzer

See the class **uniba.mri.lucene.MyAnalyzer**

```
// Creiamo una variabile statica STOP_WORDS
public static final CharArraySet STOP_WORDS;

// e la inizializziamo
static { final List<String> stopWords = Arrays.asList("a", "an", "and",
"are", "the", "is", "but", "by");
    //strutture dati ottimizzate per memorizzare stringhe
    final CharArraySet stopSet = new CharArraySet(stopWords, false);
    STOP_WORDS = CharArraySet.unmodifiableSet(stopSet);
}

@Override
protected TokenStreamComponents createComponents(String fieldName) {
    Tokenizer source = new LetterTokenizer(); //divide parola per parola
    TokenStream f = new LowerCaseFilter(source); //minuscole
    f = new StopFilter(filter, STOP_WORDS); // nuovo filtro
    return new TokenStreamComponents(source, filter);
}
```

Exercise 1

- Index the famous novel **Alice In Wonderland** in a Lucene searchable index.
 - Find the content in:
`./resources/Alice_Adv_Lewis_Carroll_utf8.text`
- You should index each **Chapter** as a separate Lucene Document.
- Each Document should contain four fields:
 1. The **Title** of the Book
 2. The **Author** of the Book
 3. The **title** of the Chapter
 4. The **text** of the Chapter

Of course, **title and author remain the same** for all the chapters/documents, while the **title and the text change**

Exercise 1 - walkthrough

1. Create a **new index** (indicate the directory as a parameter)
2. Create a **new Analyzer** and a **new IndexWriter**, with their configurations

Exercise 1 - walkthrough

1. Create a **new index** (indicate the directory as a parameter)

```
FSDirectory fsdir = FSDirectory.open(new  
    File("./resources/alice").toPath());
```

2. Create a **new Analyzer** and a **new IndexWriter**, with their configurations

```
StandardAnalyzer analyzer = new StandardAnalyzer();
```

```
IndexWriterConfig config = new IndexWriterConfig(analyzer);  
config.setOpenMode(IndexWriterConfig.OpenMode.CREATE_OR_APPEND);  
IndexWriter writer = new IndexWriter(fsdir, config);
```

Exercise 1 - walkthrough

1. Create a **new index** (indicate the directory as a parameter) 
2. Create a **new Analyzer** and a **new IndexWriter**, with their configurations 
3. **Open the text file** with the content. Create a variable for each element to be stored (title, author, chapter, text)
4. Go through the content and identify a strategy to **isolate the different pieces of text**
 - a) When do you understand that a new chapter started?
 - b) When do you understand that a new paragraph started?
5. Store each paragraph (together with the other fields) **as a new document.**

Exercise 1 - walkthrough

3. Open the text file with the content. Create a variable for each element to be stored (title, author, chapter, text)

Exercise 1 - walkthrough

3. Open the text file with the content. Create a variable for each element to be stored (title, author, chapter, text)

Suggestion:

Write the instructions to open a text file (i.e.,with **BufferedReader** and **readline()**) and go through the content.

If you need it, print the content in the console and/or open the source document in a text editor

Exercise 1 - walkthrough

3. Open the text file with the content. Create a variable for each element to be stored (title, author, chapter, text)

Suggestion:

Write the instructions to open a text file (i.e.,with **BufferedReader** and **readline()**) and go through the content.
If you need it, print the content in the console and/or open the source document in a text editor

String title = b.readLine(); Rigo 1
b.readLine(); Rigo 2 (vuoto)

String author = b.readLine(); Rigo 3
b.readLine(); Rigo 4 (vuoto)

... poi iniziano i capitoli :-)

Exercise 1 - walkthrough

4. Isolate the different pieces of text

- a) When do you understand that a new chapter has started?
- b) When do you understand that a new paragraph has started?

Suggestion: Check empty lines and specific words :-)

Exercise 1 - walkthrough

4. Isolate the different pieces of text

- a) When do you understand that a new chapter has started?
- b) When do you understand that a new paragraph has started?

Suggestion: Check empty lines and specific words :-)

Solution

- Each chapter start with the word CHAPTER
- Each new paragraph, start the next line

Exercise 1 - walkthrough

4. Isolate the different pieces of text

- a) When do you understand that a new chapter has started?
- b) When do you understand that a new paragraph has started?

Suggestion: Check empty lines and specific words :-)

Solution

- Each chapter start with the word CHAPTER
 - If (readLine.contains("CHAPTER"))
- Each new paragraph, start the next line

How to check the solution?

Luke is a GUI tool for inspecting, searching, and analyzing Apache Lucene indexes, allowing users to explore documents, terms, and analyzers interactively.

How to check the solution?

Luke is a GUI tool for inspecting, searching, and analyzing Apache Lucene indexes, allowing users to explore documents, terms, and analyzers interactively.



How to obtain it?

It is included in the binary release of Lucene

<https://www.apache.org/dyn/closer.lua/lucene/java/10.3.1/lucene-10.3.1.tgz>

How to check the solution?

- The binary is included in the `./bin/` folder
- Select the folder where the index is stored (the folder!)
- Open it and analyze the index that is stored

