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

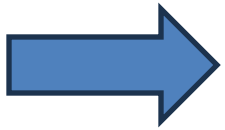
cataldo.musto@uniba.it

Credits: Marco Polignano

Code Repository & Requirements

Code repository

https://github.com/swapUniba/MRI_2024_25



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/8_11_2/index.html

Apache Lucene

Scan for the Doc

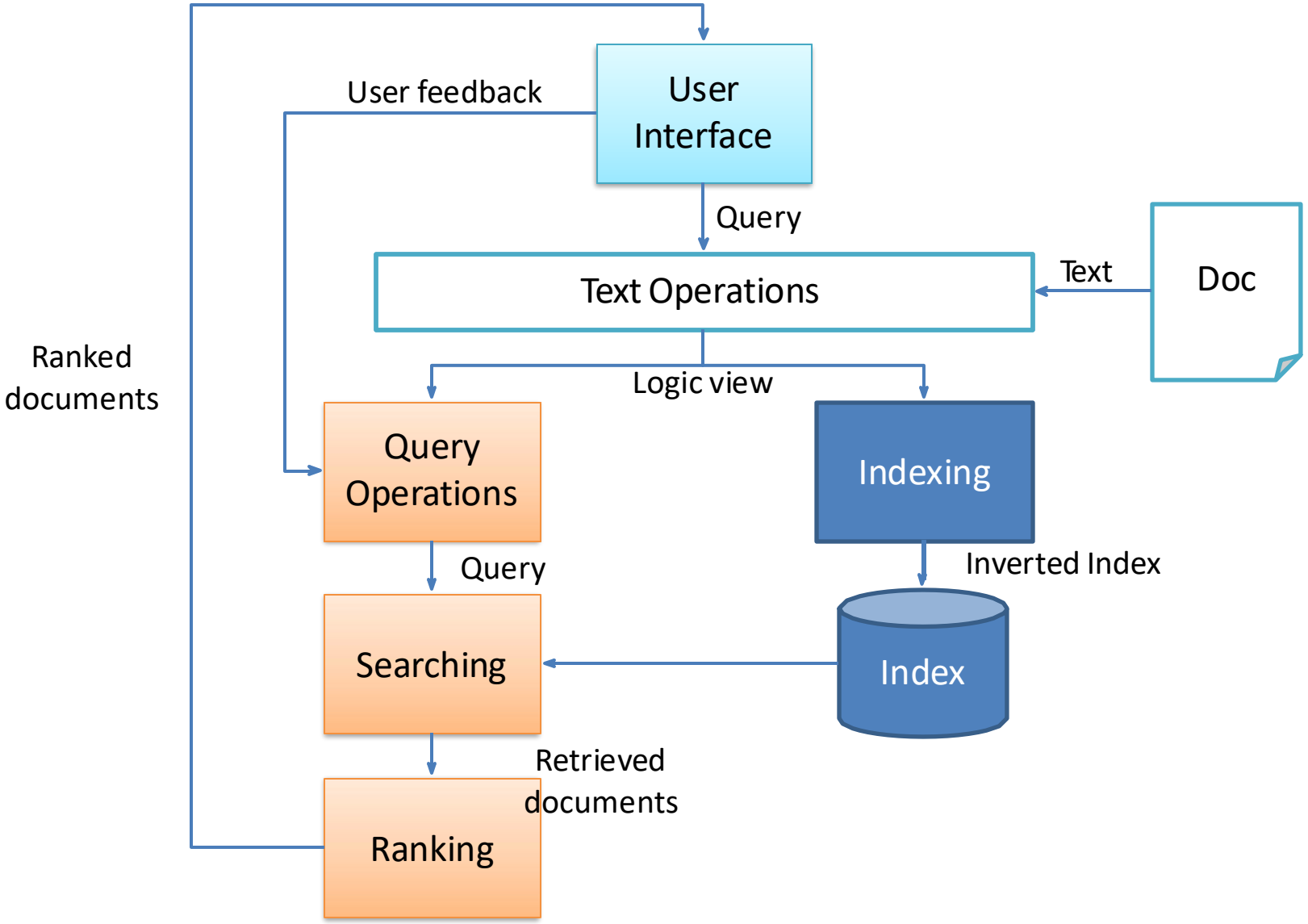
- Apache Software Foundation project
  - http://lucene.apache.org
- What Lucene is
  - Search library: indexing and searching Application Programming Interface (API)
- What Lucene is not
  - Search engine (no crawling, server, user interface, etc.)

Lucene

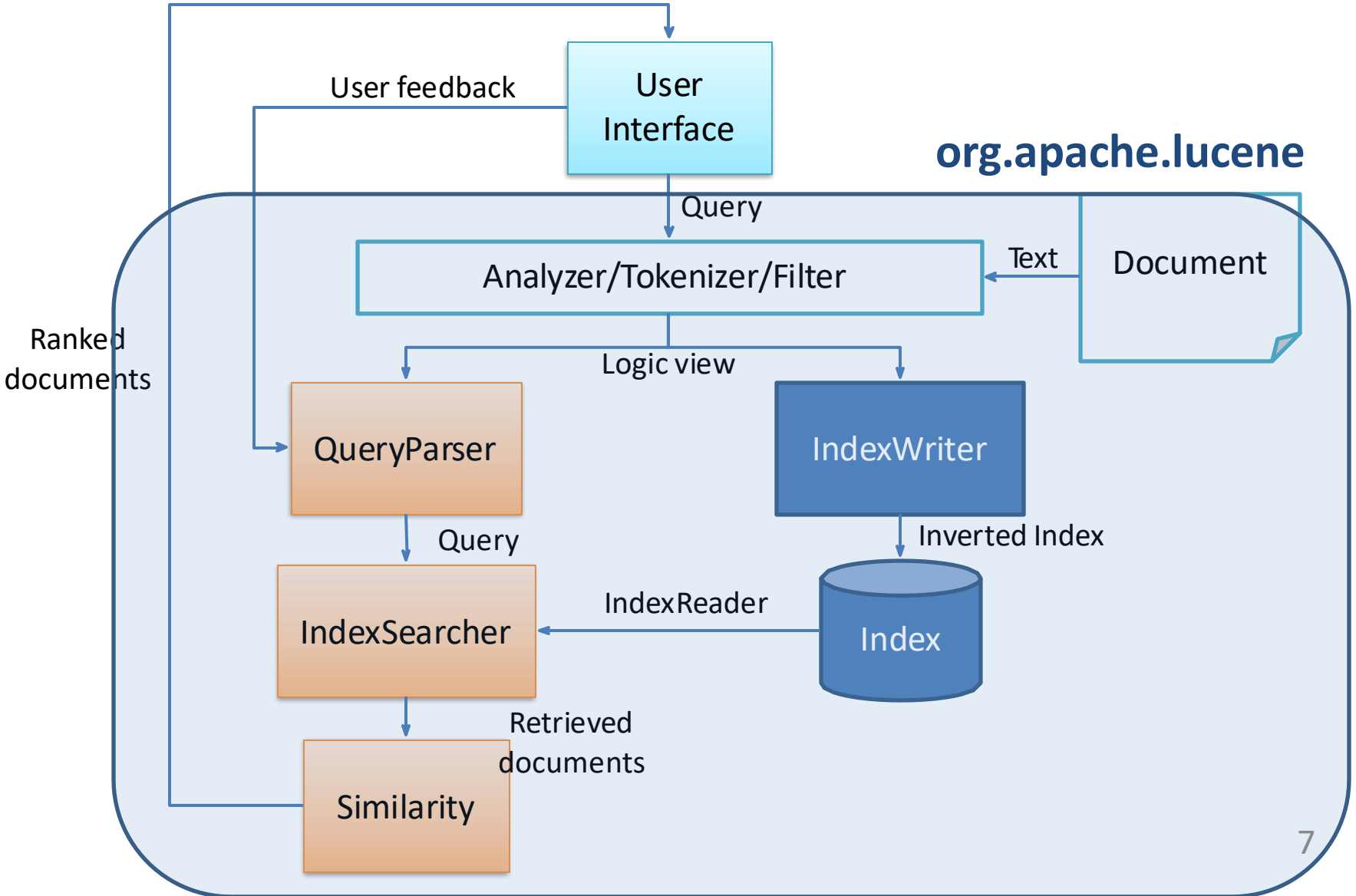
Scan for the Doc

- 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



Information Retrieval Process



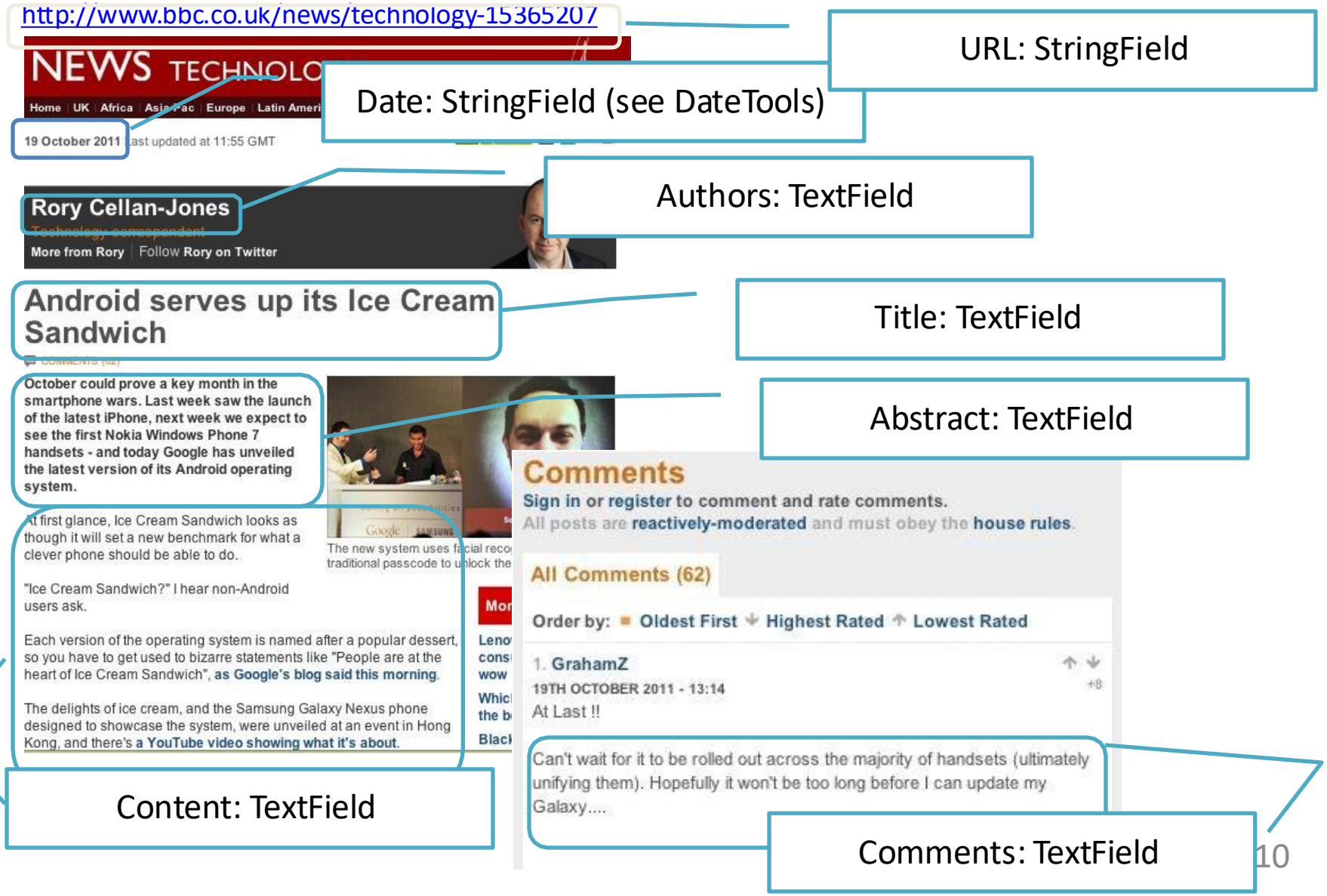
Document

- A collection of *Field*s which give structure to the document
- Each Field is added to the document
- A Field has three parts
  - Name
  - Type
  - Value
    - text (String, Reader or pre-analyzed TokenStream)
    - binary (byte[])
    - numeric (a Number)
- A Field may be stored in the index (TYPE_STORED), so it may be returned with hits on the document.
- Use FieldType for personalized types

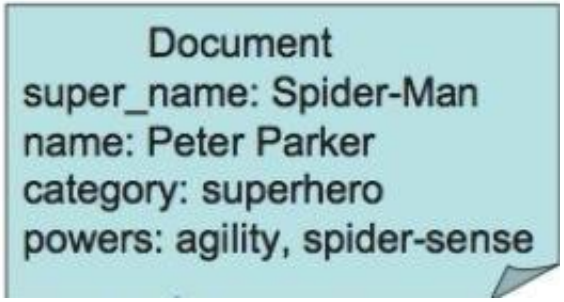
Document: Fields Representation

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

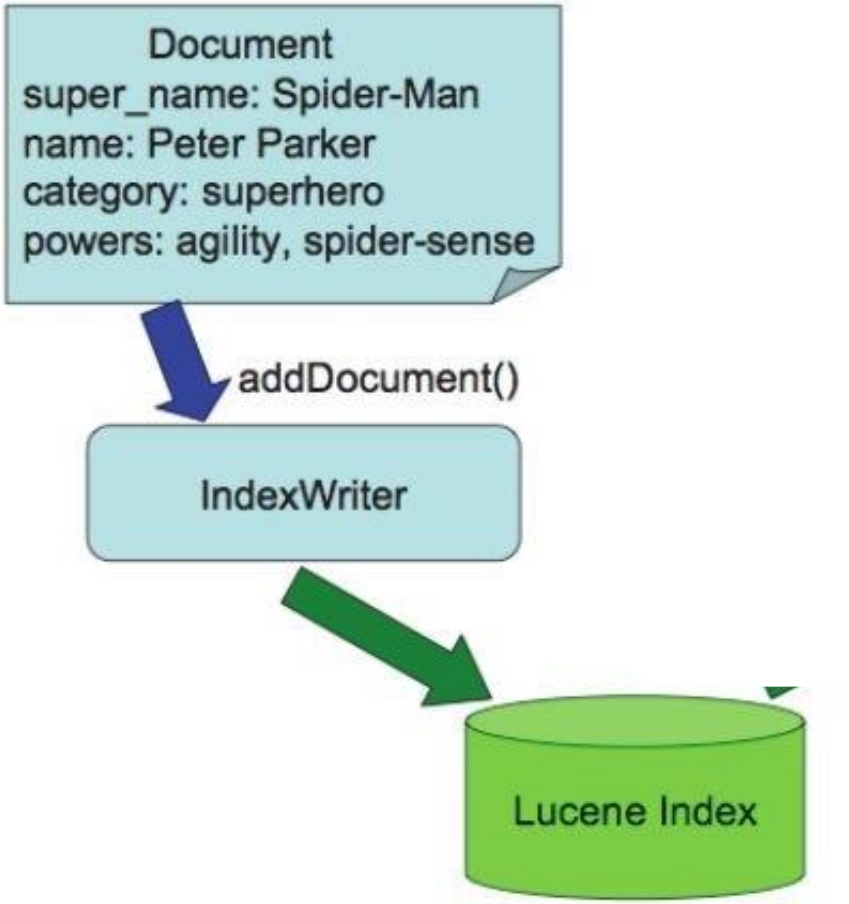
Document: Fields Representation



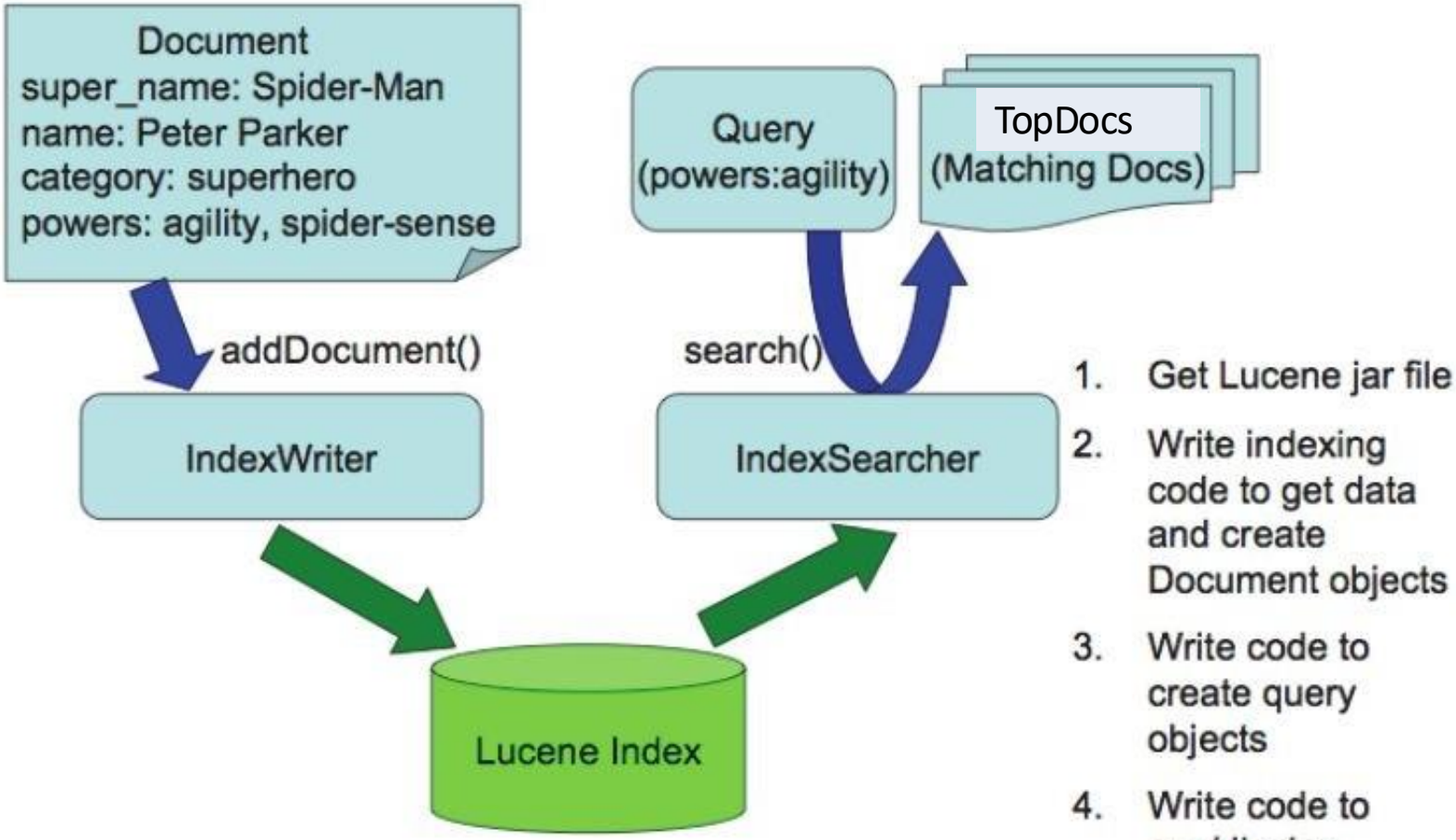
Overview of the Process



Overview of the Process

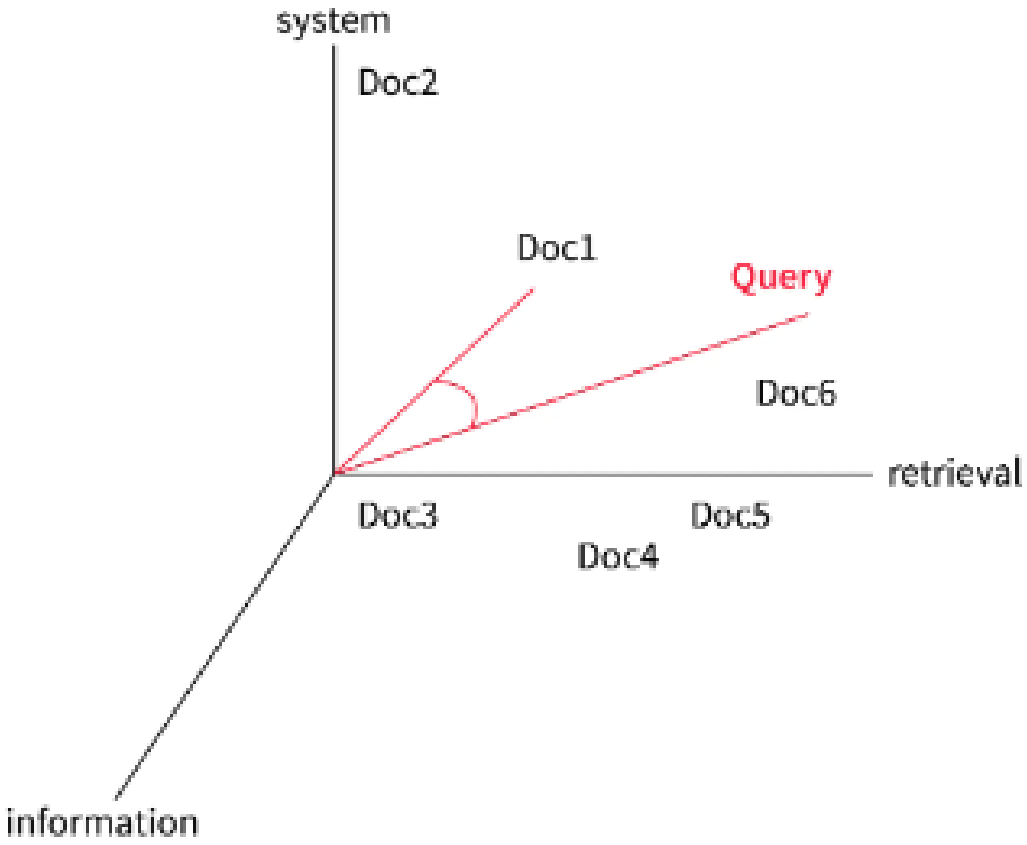


Overview of the Process



Lucene Model - Searcher

- Based on **Vector Space Model**

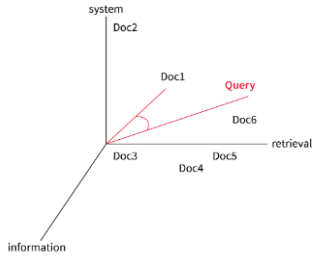


Lucene Model - Searcher

- 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

Every document is a vector!



Lucene Model - Searcher

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

Lucene Model - Searcher

$$score(q,d) = coord(q,d) \cdot queryNorm(q,d) \cdot \sum_{t \in q} (tf \cdot idf^2 \cdot boost(t) \cdot norm(t,d))$$

$$tf(t) = (\#occur(t))^{1/2}$$

$$idf(t) = 1 + \log\left(\frac{numDocs}{docFreq + 1}\right)$$

Lucene Model - Searcher

$$score(q,d) = coord(q,d) \cdot queryNorm(q,d) \cdot \sum_{t \in q} (tf \cdot idf^2 \cdot boost(t) \cdot norm(t,d))$$

$$tf(t) = (\#occur(t))^{1/2}$$

$$idf(t) = 1 + \log\left(\frac{numDocs}{docFreq+1}\right)$$

- **`coord(q,d)`** score factor based on how many query terms are found in the specified document
- **`queryNorm(q,d)`** is a normalizing factor used to make scores between queries comparable (i.e., it penalizes longer queries)
- **`boost(t)`** term boost
- **`norm(t,d)`** encapsulates a few (indexing time) boost and length factors (i.e., it penalizes longer documents)

More info https://lucene.apache.org/core/8_6_2/core/index.html

Let’s Start

Let’s start

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

git clone https://github.com/swapUniba/MRI2425

2) Create a new Java project

Import content from **./ApacheLucene/MRI2425**

Let’s start

Alternatively, setup a Java Maven project in your IDE

```
<dependencies>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-core</artifactId>
            <version>8.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-analyzers-common</artifactId>
            <version>8.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-queryparser</artifactId>
            <version>8.11.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-queries</artifactId>
            <version>8.11.2</version>
        </dependency>
</dependencies>
```

Hello World

See the class `di.uniba.it.mri2324.lucene.HelloWorld`

```
//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 `di.uniba.it.mri2324.lucene.HelloWorld`

```
//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 `di.uniba.it.mri2324.lucene.HelloWorld`

```
//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 `di.uniba.it.mri2324.lucene.HelloWorld`

```
//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

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

//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).");
```

Hello World

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


//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

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));
```

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);
```

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.

Preliminary Exercise

- Modify **`helloworld.java`**
- Add five documents (define the **fields** and the **content** you prefer)
- Check the number of documents in the index
- Try some queries, and check if the number of documents matching the query is correct

Main Text Operations

- **Tokenization**
  - split text in token
- **Stop word elimination**
  - remove common words and closed word class (e.g. function words)
- **Stemming**
  - reducing inflected (or sometimes derived) words to their stem

More info
https://lucene.apache.org/core/8_6_2/analyzers-common/index.html

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

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 a TokenStream
  - 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

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)

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 `di.uniba.it.mri2324.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 `di.uniba.it.mri2324.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 `di.uniba.it.mri2324.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()));
}
```

Custom Analyzer

See the class `di.uniba.it.mri2223.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 filter = new LowerCaseFilter(source); //minuscole
    filter = 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.

- 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

Exercise 1

- Index the famous novel **Alice In Wonderland** in a Lucene searchable index.

- 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

3. **Open the text file** with the content

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 has started?
   b) When do you understand that a new paragraph has started?
   etc.

5. Store each paragraph (together with the other fields) **as a new document.**

Exercise 1 - walkthrough

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

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

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

3. **Open the text file** with the content

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

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

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?
etc.

Suggestion: Check empty lines and specific words :-)