SAPIENZA
UNIVERSITÀ DI ROMA

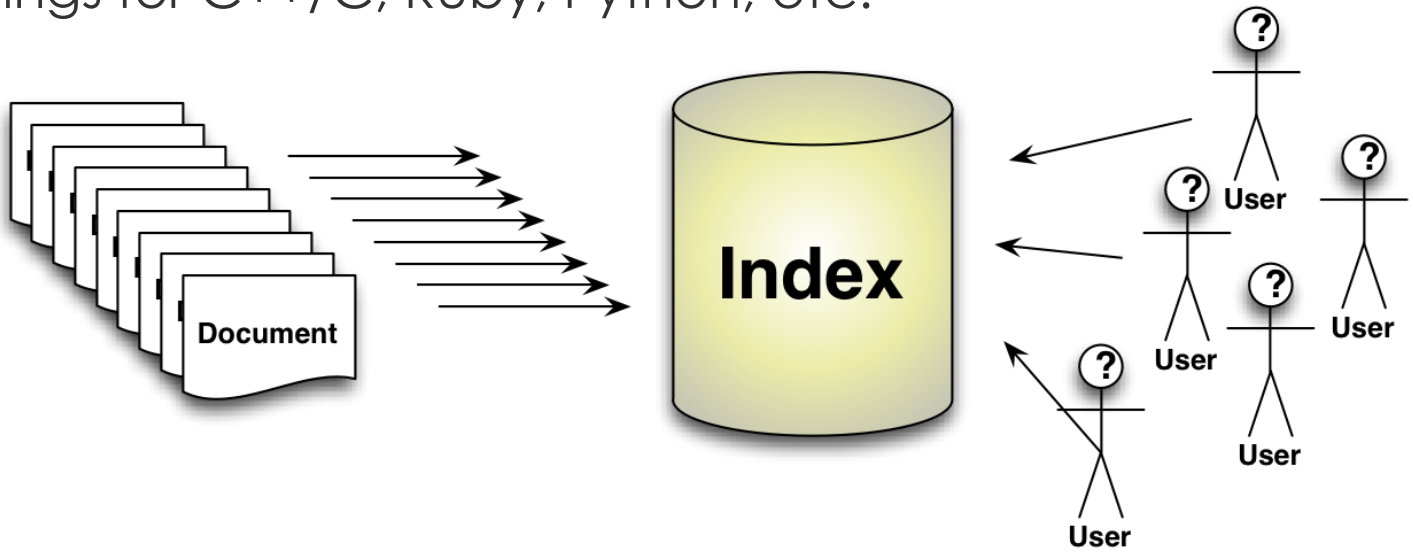
Giovanni Stilo, Ph.D.
stilo@di.uniroma1.it

Take to Wife Lucene

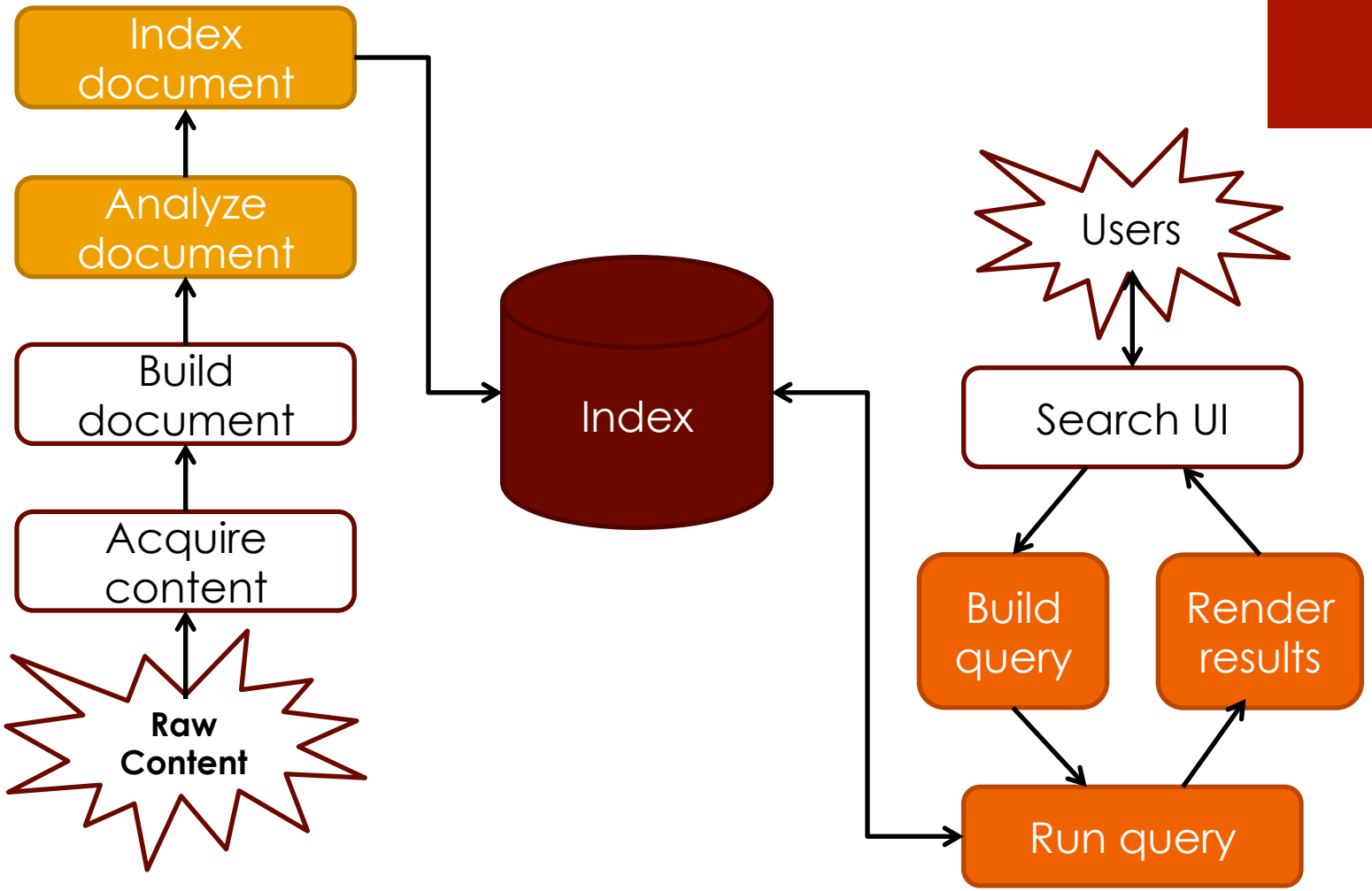
Lucene Basic Principles

Lucene ?

- Name **Lucene** came from Doug Cutting's (founder) wife's middle name.
- **Free-text** indexing library
- Implements standard **IR/search** functionality
  - Query models, ranking, indexing
- **Core API** is implemented in Java
  - Bindings for C++/C, Ruby, Python, etc.



Main Flow


Index document
Analyze document
Build document
Acquire content
Raw Content
Index
Users
Search UI
Build query
Render results
Run query

Document

- a Document is the **basic unit** for *indexing* and *searching*
  (**note**: it is different from the notion of document as file)
- each Document is a **list** of Field(s)
- each Field has a **name** and a text value
- **It is up to us to decide what to include in a Document**

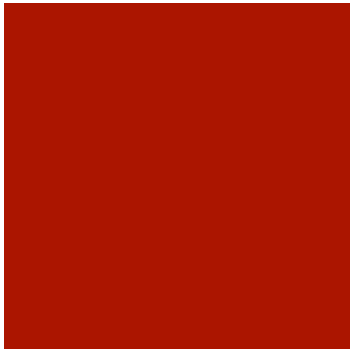
```
…
Document doc = new Document();
doc.add(new Field(…));
doc.add(new Field(…));
doc.add(new Field(…));
…
```

Field

- a field is the **basic unit** composing *Documents* - each Document is a list of Field(s).
- for each field, you need to **specify**
  - Name
  - Value
  - Options (almost storing)

```
…
Document doc = new Document();
Field f1 = new Field("fieldName1", "fieldContent1", FieldOptions1);
Field f2 = new Field("fieldName2", "fieldContent2", FieldOptions1);
doc.add(new Field(…));
…
```

Field Types



- **Numeric types:**
  - FloatField
  - DoubleField
  - IntField
  - LongField
- **Text types:**
  - StringField
  - TextField
- **Expert types:**
  - Field

Directory

- At its core, a list of files.
- RAMDirectory
  - in-memory volatile dir
  - useful for “on-the-fly” indexes

```
…
Directory dir= new RAMDirectory();
…
```

- RAMDirectory
  - file-based, persistent dir

```
…
Directory dir= new SimpleFSDirectory(new File("index"));
…
```

Indexing Documents

- Is the process of Writing the Index
- We need an IndexWriter

```
…
Directory dir= new RAMDirectory();
Analyzer analyzer = new StandardAnalyzer(LUCENE_41);
IndexWriterConfig cfg= new     IndexWriterConfig(LUCENE_41,analyzer);
IndexWriter writer = new IndexWriter(dir, cfg);
…
Document doc = new Document();
…
for(all documents) {
  field.setLongValue(value of doc);
  writer.addDocument(document);
}
writer.commit();
writer.close();
…
```

LongField*

- Field that indexes long values for **efficient** range *filtering* and *sorting*:

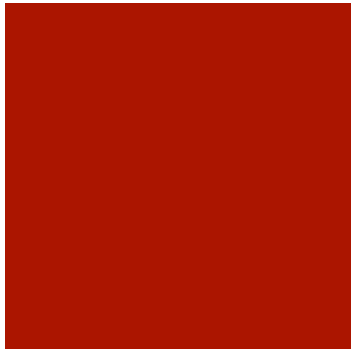
```
…
document.add(new LongField(name, 6L, Field.Store.NO));
…
```

- For optimal performance, **re-use** the **LongField** and **Document** instance for more than one document:

```
LongField field = new LongField("fieldName", 0L, Field.Store.NO);
Document document = new Document();
document.add(field);
for(all documents) {
  field.setLongValue(value of doc);
  writer.addDocument(document);
}
```

* other fields of numeric types works in the same way

Textual Fields



- **StringField**: a field that is indexed but *not tokenized*; the entire String value is indexed as a single token.
  - For example this might be used for a 'country' field or an 'id' field.

```
…
doc.add(new StringField("fieldName", content,Field.Store.YES));
…
```

- **TextField**: A field that is indexed and *tokenized*, without term vectors.
  - For example this would be used on a 'body' field, that contains the bulk of a document's text.

```
…
doc.add(new TextField ("fieldName", content,Field.Store.YES));
…
```

Field

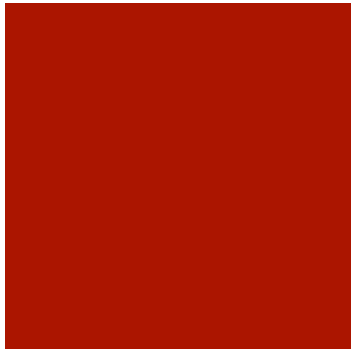
- **Expert**: *directly* create a field for a document. Most users should use one of the sugar subclasses:
  - IntField, LongField, FloatField, DoubleField, StringField, TextField.

```
FieldType MY_FLD_TYPE = new FieldType();
MY_FLD_TYPE.setIndexed(true);
MY_FLD_TYPE.setTokenized(true);
MY_FLD_TYPE.setStored(true);
MY_FLD_TYPE.setStoreTermVectors(true);
MY_FLD_TYPE.setStoreTermVectorPositions(true);
MY_FLD_TYPE.freeze();
…
doc.add(new Field("fieldName", content,MY_FLD_TYPE ));
…
```

Maven Artifacts

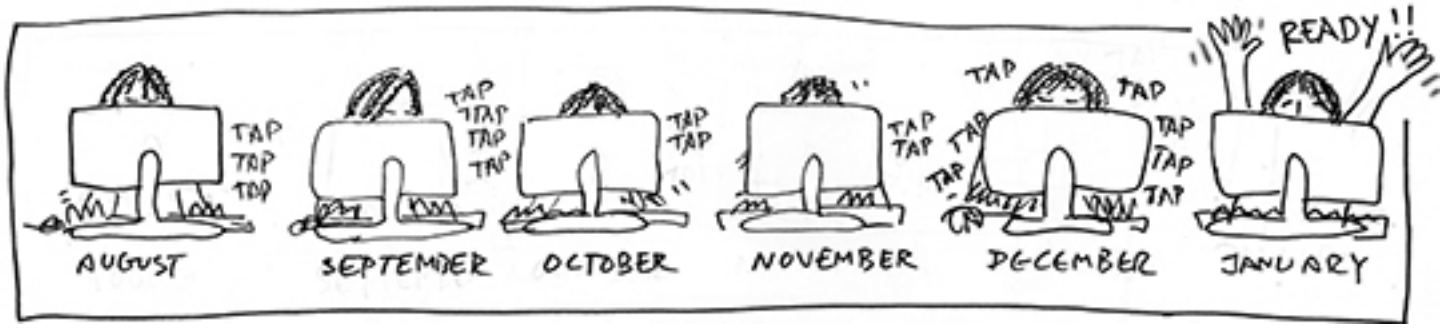
```
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-core</artifactId>
        <version>4.1.0</version>
</dependency>
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-queryparser</artifactId>
        <version>4.1.0</version>
</dependency>
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-queries</artifactId>
        <version>4.1.0</version>
</dependency>
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-analyzers-common</artifactId>
        <version>4.1.0</version>
</dependency>
```

Excercise



- Create a class that allow to index many Identity Card Documents.
- Each documents have the following fields (choosing the best type of field):
  - IC Number;
  - Name;
  - Surname;
  - Birth Date
  - State
  - City
  - Height (expressed in meter for example 1.75)

Let's Try?!?!


TAP
TAP
TAP
AUGUST
TAP
TAP
TAP
TAP
SEPTEMBER
TAP
TAP
OCTOBER
TAP
TAP
NOVEMBER
TAP
TAP
TAP
TAP
TAP
TAP
TAP
DECEMBER
READY!!
JANUARY

Searching

- We already have the index but we want to perform queries on it.
- We need to read it; so we need a IndexReader.

```
…
Directory dir= new RAMDirectory();
IndexReader ir= DirectoryReader.open(dir);
…
```

- We need to use an IndexSearcher that use IndexReader to perform a Query.

```
…
IndexSearcher searcher = new IndexSearcher(ir);
…
```

Querying

- At this point we need a Query q.
- We can manually build a query.

```
…
Query q = new TermQuery( new Term("fieldName","termToSearch");
…
```

- Or it is possible to build the query using a query parser and submit to it a string that satisfy the Query Language[1] QL.

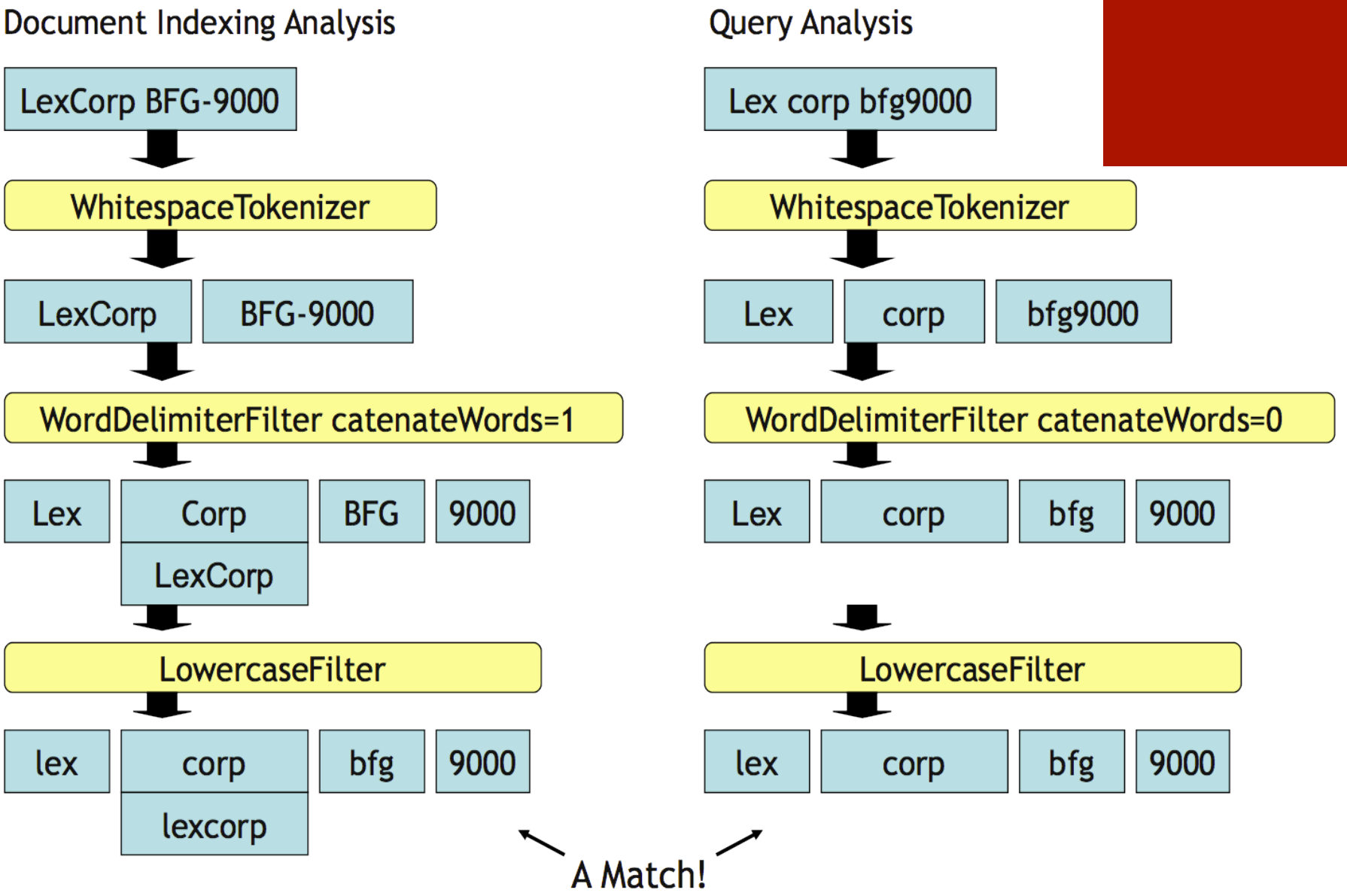
```
…
QueryParser parser = new QueryParser(USED_VERSION,
                                     "defaultFieldName", analyzer);
Query q = parser.parse(query);
…
```

1- http://lucene.apache.org/core/4_0_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description

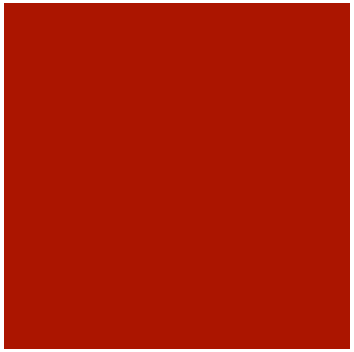
Query Language Examples

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*

Text Analysis



Text Analysers



- WhitespaceAnalyzer
  - splits on whitespace
- SimpleAnalyzer
  - splits on whitespace and special characters; applies lowercase
- StopAnalyzer
  - SimpleAnalyzer + stopword
- StandardAnalyzer
  the most complete (Whitespace+Stop+misc)
- SnowballAnalyzer performs also stemming

Perform Query

- Finally submit the query to the searcher and read results.

```
…
TopDocs top = searcher.search(q, 100); // perform a query and limit results number
ScoreDoc[] hits = top.scoreDocs; // get only the scored documents (ScoreDoc is a tuple)

Document doc=null;

for(ScoreDoc entry:hits){
    // load document in memory (only the stored filed are available)
    doc=searcher.doc(entry.doc); /* the same as ir.document(entry.doc); */

    System.out.println("field1: "+doc.get("field1"));
    System.out.println("field2: "+doc.get("field2"));
    System.out.println("field3: "+doc.get("field3"));
    System.out.println("field4: "+doc.get("field4"));
}
ir.close();
…
```

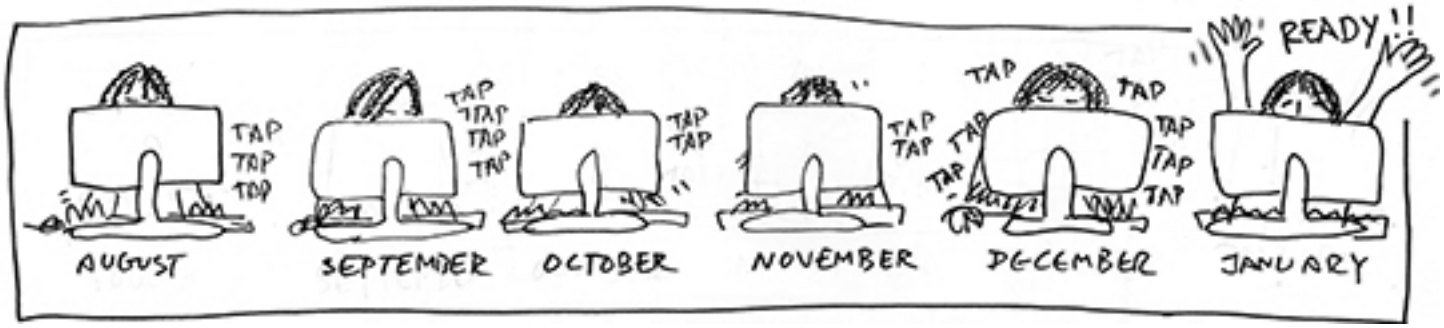
Maven Artifacts

```
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-core</artifactId>
        <version>4.1.0</version>
</dependency>
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-queryparser</artifactId>
        <version>4.1.0</version>
</dependency>
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-queries</artifactId>
        <version>4.1.0</version>
</dependency>
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-analyzers-common</artifactId>
        <version>4.1.0</version>
</dependency>
```

Excercise

- Create a class that allow to index many Identity Card Documents.
- Each documents have the following fields:
  - IC Number;
  - Name;
  - Surname;
  - Birth Date
  - State
  - City
  - Height (expressed in meter for example 1.75)
- Try to retrieve ID cards using different fields at the same time.

Let's Try?!?!


TAP
TAP
TAP
AUGUST
TAP
TAP
TAP
TAP
SEPTEMBER
TAP
TAP
OCTOBER
TAP
TAP
NOVEMBER
TAP
TAP
TAP
TAP
TAP
TAP
TAP
DECEMBER
READY!!
JANUARY