

Lucene.

Sistema de RI de código abierto

Necesidad de un S.R.I.

- En algún momento nos puede surgir la necesidad de usar un sistema de recuperación de información:
 - ♦ Investigación: para probar modelos.
 - ♦ Uso en una organización: para buscar en sus colecciones documentales.
- ¿Utilizo alguno ya existente o lo creo desde cero?
 - ♦ Según las necesidades.
 - ♦ Según el esfuerzo requerido.

Necesidad de un S.R.I.

Pasos:

- 1) Analizar los sistemas disponibles con código abierto.
- 2) Determinar si sus prestaciones satisfacen los requerimientos
- 3) En caso de que no los satisfagan,
 - 3.1) Partir de cero y crear un sistema nuevo.
 - 3.2) Modificar uno ya existente.
- 4) Aplicar el sistema.



- No es un sistema de RI propiamente dicho
- Conjunto de bibliotecas para realizar indexación y recuperación de alto rendimiento,
 - Pueden ser utilizadas por programadores para integrarlas en sus propias aplicaciones.
- Escrito enteramente en Java.
- Referencias:
 - <https://lucene.apache.org/>
 - Lucene in Action
 - <http://en.wikipedia.org/wiki/Lucene>
- Integración con otros lenguajes
 - C++, C#, PHP, Python, Ruby, Perl

Características

- Algoritmos de búsqueda potentes, fiables y eficientes
- Permite ordenar resultados por relevancia
- Lenguaje de consulta muy potente (frases, comodines, proximidad, rangos...)
- Búsqueda por campos
- Búsqueda por rango de fechas
- Ordenación por cualquier campo
- Búsqueda multi-índice y combinación de resultados
- Permite búsquedas mientras se actualiza el índice

Usuarios de Lucene:

- IBM,
- E.ON,
- LinkedIn,
- Twitter,
- Wikipedia.com
- Monster.com
- search.wikia.com
- technorati.com
- Fotocasa.com, trovit.com
- Infojobs.com.br
- Laboris.net
- Eclipse IDE
- CNET
- SourceForge.net
- Wikipedia
- TheServerSide

<https://wiki.apache.org/lucene-java/PoweredBy>

<http://lucene.apache.org/core/index.html>

Apache Lucene Core

DOWNLOAD

Apache Lucene 6.2.1

Apache Lucene™ is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Apache Lucene is an open source project available for free download. Please use the links on the right to access Lucene.

Resources

[Mailing Lists](#)

[Developer](#)

[Features](#)

[Releases](#)

[System Requirements](#)

Lucene™ Features

Lucene offers powerful features through a simple API:

Scalable, High-Performance Indexing

- over 150GB/hour on modern hardware
- small RAM requirements -- only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30% the size of text indexed

Release Docs

[6.2.1](#)

About

[License](#)

[Who We are](#)



Powerful, Accurate and Efficient Search Algorithms

- ranked searching -- best results returned first
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g. title, author, contents)
- sorting by any field
- multiple-index searching with merged results
- allows simultaneous update and searching
- flexible faceting, highlighting, joins and result grouping
- fast, memory-efficient and typo-tolerant suggesters
- pluggable ranking models, including the [Vector Space Model](#) and [Okapi BM25](#)
- configurable storage engine (codecs)

Cross-Platform Solution

- Available as Open Source software under the [Apache License](#) which lets you use Lucene in both commercial and Open Source programs
- 100%-pure Java
- Implementations [in other programming languages available](#) that are index-compatible

Lucene

- No se ofrece ningún instalador
- No ofrece aplicaciones de administración ni en línea de mandatos (sólo a modo de ejemplo).
- No incluye arañas para indexar páginas web.
- No soporta directamente ficheros en formato HTML.
- No soporta documentos de MS Word.
- No soporta XML ni consultas XML.
- No suporta el formato PDF.
- No existe pasarela hacia bases de datos.
- No existe interfaces para realizar búsquedas web.
- No existe “teléfono de consultas del departamento técnico”, pero sí una amplia comunidad de usuarios.

¿Qué no es Lucene?

- No es un producto listo para ser usado.
- Sólo ofrece bibliotecas para crear aplicaciones potentes
- Documentación
http://lucene.apache.org/core/6_2_1/

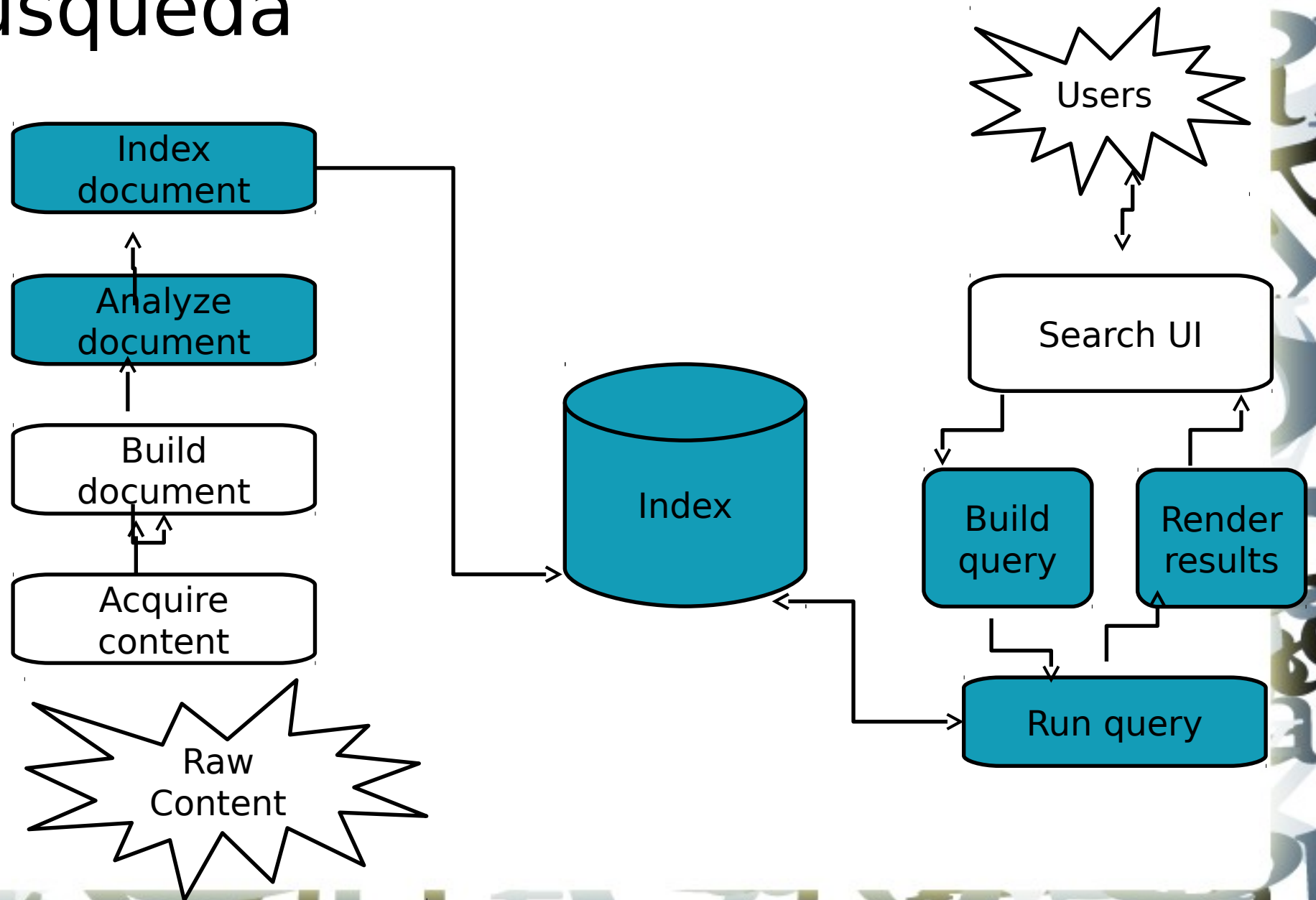
Un poco de Historia

- Desarrollado por Doug Cutting 1997-
- Fue puesta como código abierto (open source) en Marzo de 2000 a través de SourceForge.
- Se unió a Apache Software Foundation en 2001
 - ♦ Familia Jakarta de software escrito en Java

No es útil para buscar en la red

- ♦ No tiene araña
 - NUTCH (motor de búsqueda open source (Apache))
- ♦ No tiene analizador sintáctico (parser)
 - ➔ Solr (motor de búsqueda también basado en Lucene)

Lucene En un sistema de búsqueda



Instalación de Lucene

- Obtener la version de
 - ♦ <https://lucene.apache.org/>
 - ♦ 28 Octubre de 2016 -
 - Apache Lucene 6,2,1
 - ♦ Código fuente o fichero binario
- Encontraremos ficheros .jar
 - ♦ lucene-core-XXX.jar
 - ♦ lucene-analyzer-XXX.jar
 - ♦ Lucene-queryparser-XXX.jar
 - ♦ Es necesario introducir estos fichero en las aplicaciones que lo usan:
 - Deben estar referenciados en el classpath cuando este corriendo la máquina virtual de Java.

New Project

Steps

1. Choose Project
2. ...

Choose Project

Categories:

- Java
- NetBeans Modules
- Samples

Projects:

- Java Application
- Java Desktop Application
- Java Class Library
- Java Project with Existing Sources
- Java Free-Form Project

Description:

Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

< Back

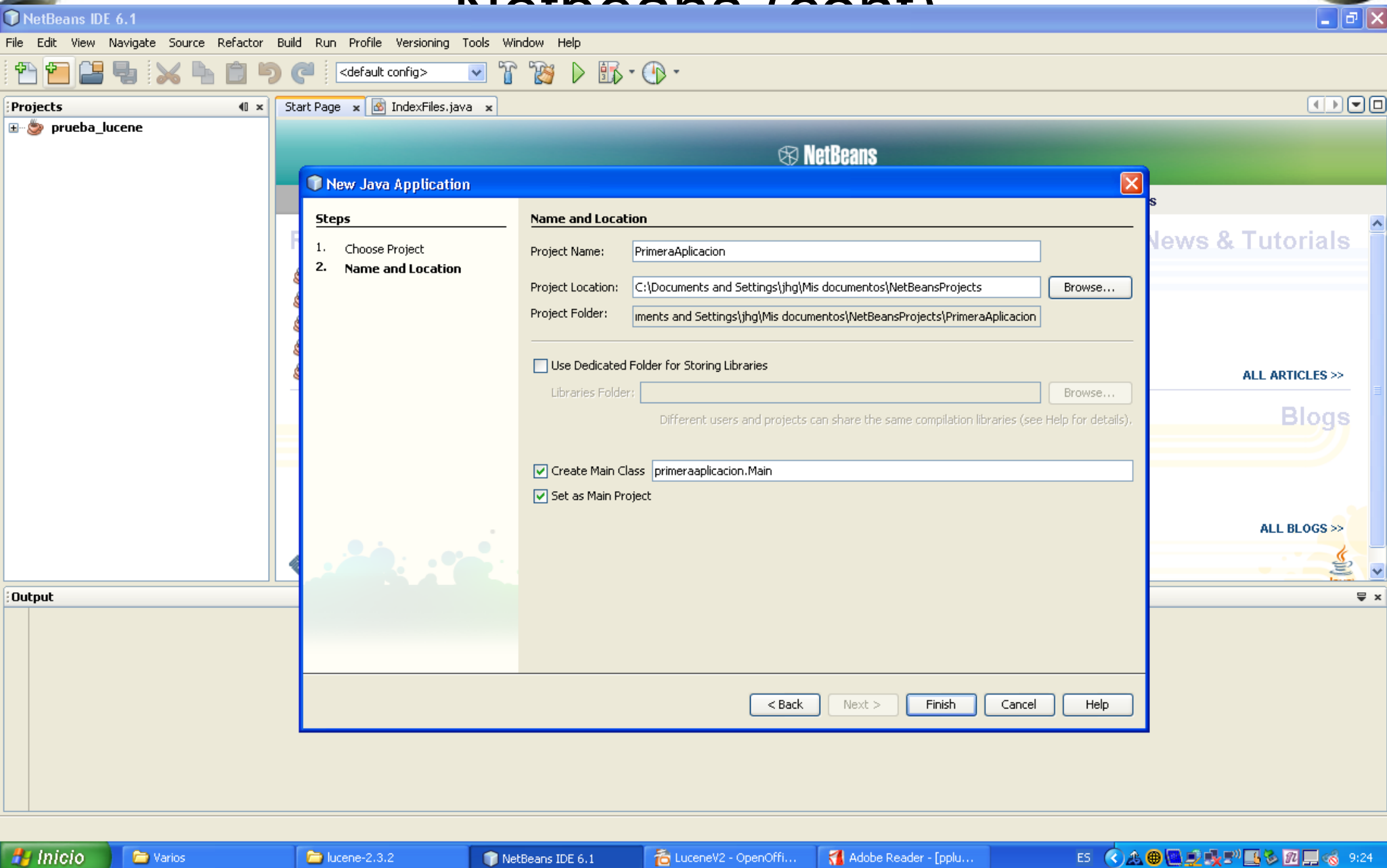
Next >

Finish

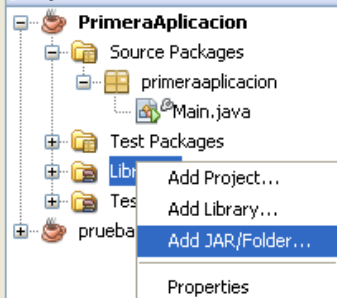
Cancel

Help

Netbeans (cont)



Projects



Start Page x IndexFiles.java x Main.java x

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package primeraaplicacion;

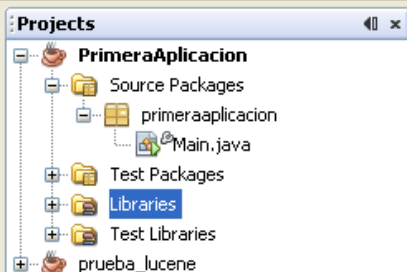
/**
 *
 * @author jhg
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

1:1 INS

Output



Start Page x IndexFiles.java x Main.java x



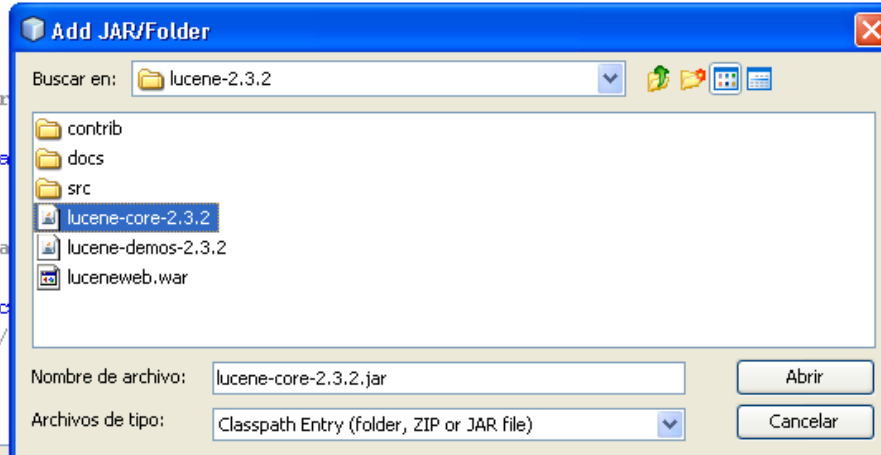
```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package primeraaplicacion;
```

```
/**
 *
 * @author
 */
public class
```

```
/**
 *
 * @pa
 */
public
//
}
```

1:1 INS



Output

Con Lucene podemos...

- **Indexar**
 - crear estructuras de datos que permitan un acceso rápido y aleatoria a la información que almacenan.
- **Búsquedas**
 - El proceso de buscar y recuperar información almacenadas en los índices con el propósito de acceder a los documentos concretos.
- Lucene no está diseñado para trabajar con datos altamente volátiles. Cuando los documentos cambian, los indexeds deben ser actualizados para poder reflejar los nuevos términos de forma correcta.
 - Actualizar implica “borrar”+”añadir”
 - Los datos añadidos serán visibles para las nuevas sesiones, lo que puede implicar problemas de sincronización.

Muy, Muy básica aplicación con lucene

- Para usar Lucene, una aplicación debería:
 - ♦ Crear Documents añadiéndole Fields;
 - ♦ Crea un IndexWriter y añadir documentos con addDocument();
 - ♦ Llamar a QueryParser.parse() para construir una consulta desde un string
 - ♦ Crear un IndexSearcher y pasarle la consulta con el método search().

Apache Lucene: Indexación

```
Analyzer analyzer = new StandardAnalyzer();

// Store the index in memory:
Directory directory = new RAMDirectory();
// en disco ... //Directory directory = FSDirectory.open("/tmp/testindex");
IndexWriterConfig config = new
    IndexWriterConfig(analyzer);
IndexWriter iwriter = new IndexWriter(directory, config);
Document doc = new Document();
String texto = "En un lugar de la Mancha de cuyo ... .";
TextField cuerpo = new TextField("nCampo",texto, Field.Store.NO);
TextField autor = new TextField("nAutor", "Miguel de Cervantes",
Cervantes",Field.Store.YES);
doc.add(cuerpo);
doc.add(autor);
iwriter.addDocument(doc);
iwriter.close();
```


Apache Lucene: Búsqueda

.....

```
IndexReader ireader = DirectoryReader.open(directory);
IndexSearcher isearcher = new IndexSearcher(ireader);
// Parse a simple query that searches for "text":
QueryParser parser = new
    QueryParser("nCampo", analyzer);
Query query = parser.parse("lugar mancha");
ScoreDoc[] hits = isearcher.search(query, null, 1000).scoreDocs;
// Iterate through the results:
for (int i = 0; i < hits.length; i++) {
    Document hitDoc = isearcher.doc(hits[i].doc);
    System.out.println("salida "+hitDoc.get("nAutor").toString());
    System.out.println("salida "+hitDoc.toString());
}
ireader.close();
directory.close();
```

5 Uso de Lucene

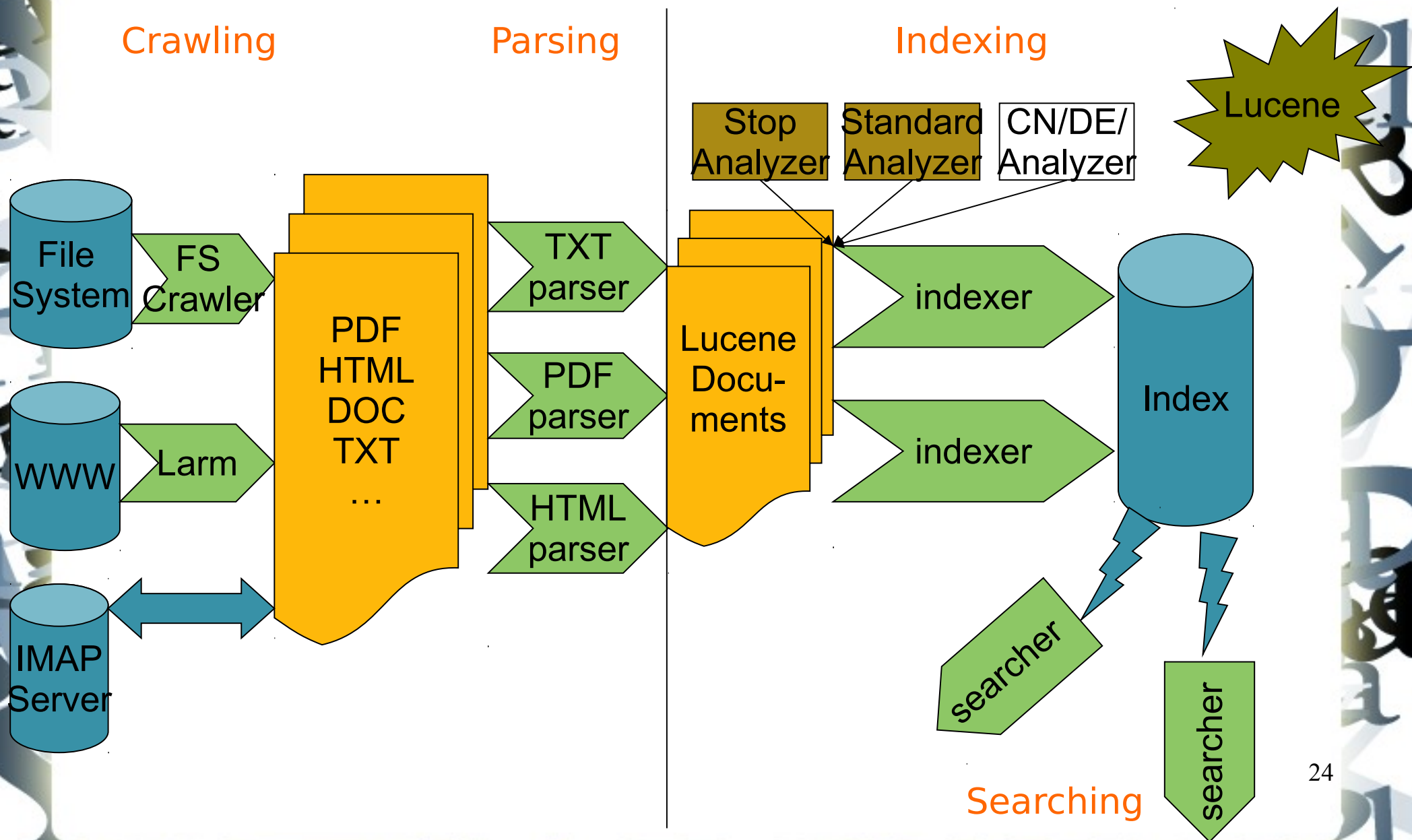
- Incluir en el classpath o ejecutar
 - `javac -classpath lucene.jar <my .java>`
 - `java -cp lucene.jar:myapp.jar <myClass>`

- Importar las clases a utilizar

```
import org.apache.lucene.analysis.Analyzer;  
import org.apache.lucene.analysis.standard.StandardAnalyzer;  
import org.apache.lucene.store.Directory;  
import org.apache.lucene.store.RAMDirectory;  
import org.apache.lucene.index.DirectoryReader;  
import org.apache.lucene.search.IndexSearcher;  
import org.apache.lucene.queryparser.classic.QueryParser;  
import org.apache.lucene.queryparser.classic.ParseException;  
import org.apache.lucene.search.Query;  
import org.apache.lucene.document.Document;  
import org.apache.lucene.document.Field;  
import org.apache.lucene.search.ScoreDoc;  
import org.apache.lucene.document.TextField;  
import org.apache.lucene.index.IndexWriter;  
import org.apache.lucene.index.IndexWriterConfig;  
import org.apache.lucene.store.FSDirectory;  
import org.apache.lucene.util.Version;
```

Indexación con Lucene

Arquitectura de Lucene



Indexación

- **Proceso de preparar y añadir texto a Lucene**
 - Optimizado para búsquedas
 - ♦ Antes de poder realizar cualquier operación debemos indexar un conjunto de documentos de texto y crear el fichero invertido índice de términos
- **Clave: Lucene sólo indexa Strings**
 - Qué significa esto?
 - Lucene no trabaja directamente sobre XML, Word, PDF, etc.
 - Podemos encontrar muchas herramientas de código abierto que nos facilitan esta labor:
 - Es nuestra labor convertir cualquier formato en algo que Lucene pueda utilizar
 - Extraer String de los ficheros fuente

¿Qué se podría indexar?

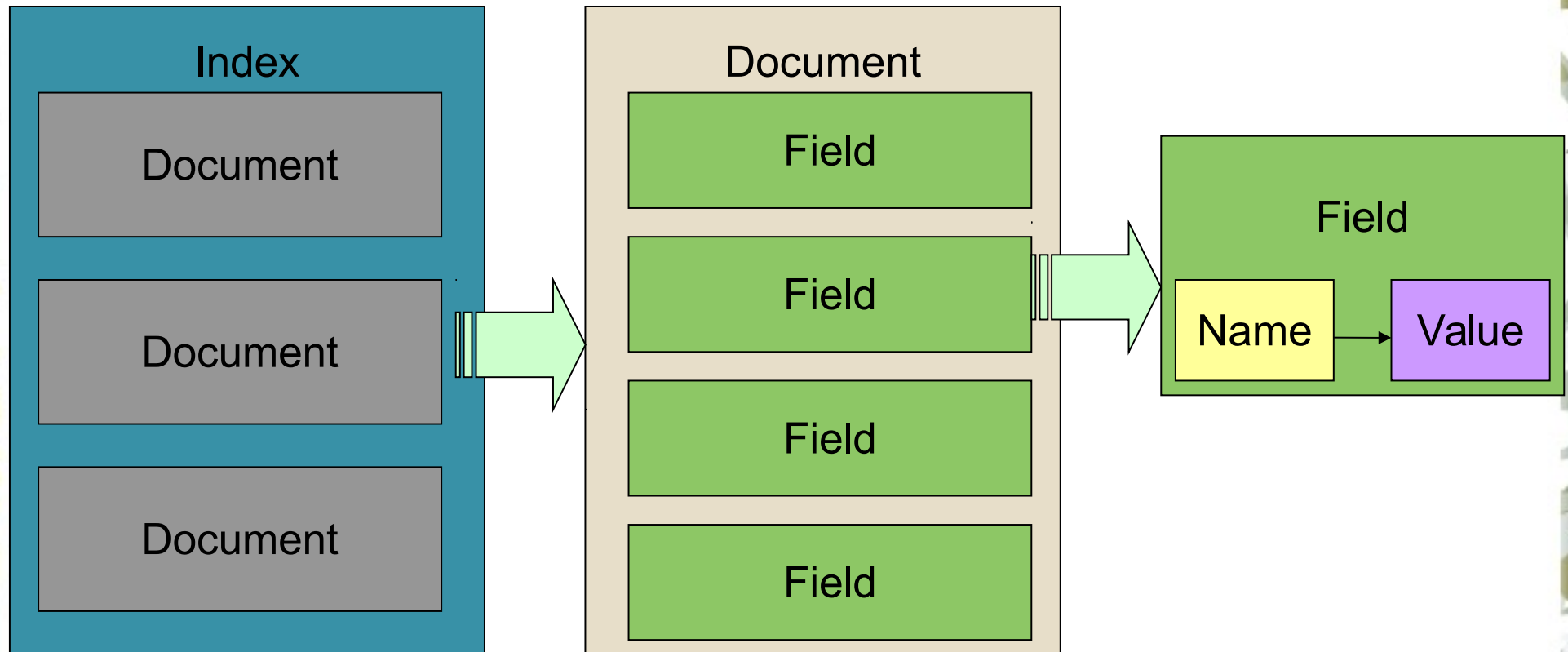
- Texto plano (plugin: parse-text)
- HTML (parse-html)
- JavaScript (for extracting links only?) (parse-js)
- Microsoft Power Point (.ppt) (parse-mspowerpoint)
- Microsoft Word (.doc) (parse-msword)
- Adobe PDF (parse-pdf)
- RSS (parse-rss)
- RTF (parse-rtf)
- MP3: las marcas ID3v1 ó ID3v2 que contienen información sobre el título, artista, álbum, etc (la información útil para buscar mp3s).
- ZIP: Expandir el fichero zipo y devolver texto concatenado (parse-zip)

Indexación

Utilizaremos los siguientes recursos que nos ofrece Lucene

- **Document:** La clase que representa los documentos. Indexamos objetos document y recuperamos objetos document
- **Field:** Esta clase representa campos de un documento
- **Analyzer:** Es una clase abstracta que representa el conjunto de analizadores de términos que debemos utilizar para analizar los documentos y obtener los términos que hay en ellos
- **IndexWriter:** Crea y mantiene el índice de terminos (abre un índice existente, añade, borra o documentos)
- **Directory:** Clase que representa la localización de un índice₂₇

Indexación en Lucen (conceptual)



Clases en el proceso de Indexación

- `IndexWriter`
 - Es el componente central del proceso de indexación.(Responsable de convertir texto en el formato interno de Lucene).
 - Esta clase crea un nuevo índice y añade documentos al índice existente.
 - Es un objeto que da acceso de escritura al índice pero no de lectura o búsqueda.

`Import org.apache.lucene.index.IndexWriter;`

Constructor:

`IndexWriter(Directory d, IndexWriterConfig iwc)`

Construir un indice en el directorio d

InderWriter

```
public IndexWriter(Directory d, IndexWriterConfig conf)
```

throws IOException

Construye un IndexWriter con los parámetros dados en conf

Parámetros

d -el directorio donde se aloja el índice

conf – los parámetros de configuración:

- ♦ Versión de Lucene
- ♦ Analizador utilizado
- ♦ (.... Crear, Añadir, Crear_o_Añadir)

Clases en el proceso de Indexación

- Analyzer
- (veremos detalladamente después)
 - Se especifica en el constructor de la clase IndexWriterConfig. Extrae los términos índice del texto que se va a indexar. Si el contenido a ser indexado no está en formato texto, debe convertirse primero.
 - Representa el tipo de proceso que se realiza sobre los términos
 - Reducción de una palabra a su raíz (stemming).
 - Eliminación de palabras vacías.
 - Identificación de frases.

Import [org.apache.lucene.analyzer;](#)

Constructor:

- [Analyzer\(\)](#)

Clases en el proceso de Indexación

- `Directory`
 - Es la localización del índice de Lucene. Un índice se representa como un conjunto de ficheros. Cuando se crea se puede abrir para lectura, o borrado.
 - En las aplicaciones es necesario almacenar el índice de Lucene, para lo cual se tienen las siguientes clases:
 - a) `FSDirectory`: Mantiene la lista real de archivos en un directorio.
 - b) `RAMDirectory` Mantiene los datos en memoria. Para mayor rapidez de búsqueda,
- `Import org.apache.lucene.store.Directory;`

Como Indexar?

- Crear `IndexWriter`
- Para cada entrada
 - Crear un `Document`
 - Añadir los `Fields` al `Document`
 - Añadir el `Document` al `IndexWriter`
- Cerrar el `IndexWriter`
- Optimizar (opcional)

Creando un IndexWriter

```
import org.apache.lucene.index.IndexWriter;  
Import org.apache.lucene.util;  
import org.apache.lucene.store.Directory;  
import org.apache.lucene.index.IndexWriterConfig;  
import org.apache.lucene.analysis.standard.StandardAnalyzer;  
...  
private IndexWriter writer;  
...  
public Indexer(String indexDir) throws IOException {  
    Version version = Version.LUCENE_XY; // Sustituir  
    Directory dir = FSDirectory.open(new File(indexDir));  
    Analyzer analizador = new StandardAnalyzer();  
    IndexWriterConfig iwc = new IndexWriterConfig(analizador);  
    Iwc.setOpenMode(OpenMode.CREATE);  
    writer = new IndexWriter(dir, iwc);  
}
```

Document

- § Son las unidades de indexación y búsqueda.
- § Esta computesto por un conjunto de campos (**Field**: pares nombre y valor).
 - Además de la indexación, un campo se puede almacenar opcionalmente en el índice, en cuyo caso lucene lo puede devolver a la hora de recuperar. Así un documento tiene usualmente uno o más campos almacenados que lo identifican unívocamente.
 - Los campos que no se almacenan no estan disponibles en la lista de hits.

Field: componentes

- Nombre, cadena que identifica al campo
- Valores concretos
 - Texto: String, Reader o un TokenStream (pre-analizado),
 - binary (byte[]),
 - Numérico.
- Store (Se almacena)
 - `Field.Store.YES` Almacena los valores en el índice (util para textos cortos como nombre del documento, que puede ser mostrado con los resultados). El valor se almacena en su forma original (no se analiza)
 - `Field.Store.NO` No se almacena el valor en el índice
- Mejor utilizar las subclases

Subclases de Field

- TextField: Reader or String indexed for full-text search
- StringField: String indexed verbatim as a single token
- IntPoint: int indexed for exact/range queries.
- LongPoint: long indexed for exact/range queries.
- FloatPoint: float indexed for exact/range queries.
- DoublePoint: double indexed for exact/range queries.
- SortedDocValuesField: byte[] indexed column-wise for sorting/faceting
- SortedSetDocValuesField: SortedSet<byte[]> indexed column-wise for sorting/faceting
- NumericDocValuesField: long indexed column-wise for sorting/faceting
- SortedNumericDocValuesField: SortedSet<long> indexed column-wise for sorting/faceting
- StoredField: Stored-only value for retrieving in summary results

Ej: IntField/StringField/TextField

- **IntPoint** indexa valores enteros. Permite hacer consultas con filtrado y ordenación por sus valores. NO sirve para mostrar, necesitamos añadir un campo extra
 - `IntPoint field = new IntPoint("edad", 6, Field.Store.NO);`
 - `field.setIntValue(32);`
- **StringField** se indexa pero no tokeniza: La cadena entera se almacena como un único token. Útil para un campo país o Id.
 - `... new StringField("nombre", "Nueva Guinea", Field.Store.YES);`
- **TextField**. Se indexa y tokeniza. Es útil para el 'cuerpo' de un texto, que contiene el grueso del documento.
 - `... aux = new TextField("cuerpo", texto, Field.Store.NO);`
 - `TextField(String name, Reader reader)`
 - `TextField(String name, String value, Field.Store store)`
 - `TextField(String name, TokenStream stream)`

```
Document doc1 = new Document();

doc1.add(new StringField("asin", "B005XSS8VC", Field.Store.YES));

doc1.add(new SortedSetDocValuesField("format", new BytesRef("kindle")));

Field titleField1 = new Field("title", "What's New in Java 7", titleType); //Tipos definidos por el usuario (sig. transp)
titleField1.setBoost(3.0f);

doc1.add(titleField1);

doc1.add(new SortedDocValuesField("publisher", new BytesRef("O'Reilly Media")));

doc1.add(new Field("author", "Madhusudhan Konda", authorType)); //Tipos definidos por el usuario (sig. transp)

doc1.add(new Field("summary", "Java 7 has a number of features that will please developers. Madhusudhan Konda provides an overview of these, including strings in switch statements, multi-catch exception handling, try-with-resource statements, the new File System API, extensions of the JVM, support for dynamically-typed languages, and the fork and join framework for task parallelism.", summaryType)); //Tipos definidos por el usuario (sig. transp)

doc1.add(new NumericDocValuesField("page", 19));

doc1.add(new LegacyIntField("size", 148, Field.Store.YES));

doc1.add(new SortedNumericDocValuesField("price", 0));

doc1.add(new IntPoint("rating", 1));

doc1.add(new StringField("rating_display", "1", Field.Store.YES));
```

```
FieldType titleType = new FieldType();
```

```
titleType.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS);
```

```
titleType.setStored(true);
```

```
titleType.setTokenized(true);
```

```
FieldType authorType = new FieldType();
```

```
authorType.setIndexOptions(IndexOptions.DOCS_AND_FREQS);
```

```
authorType.setStored(true);
```

```
authorType.setOmitNorms(true);
```

```
authorType.setTokenized(true);
```

```
authorType.setStoreTermVectors(true);
```


Document como crearlos

```
public static Document crearDocumento(String nombre, String
descri,    int habitaciones)
Document doc = new Document(); //doc vacío, sin campos.
    doc.add(new StringField("hotel", nombre,
                                Field.Store.YES));
    doc.add(new TextField("desc", descr,
                                Field.Store.NO));
    doc.add(new IntPoint("habit", habitaciones);
return doc;
}
...
Documet doc;
doc = crearDocumento("Los Patos", "Hotel con encanto",75);
doc = crearDocuemnto("AC Santa Paula", "Centrico, con encanto,
lujo", 250);
```

Si quisiéramos mostrar en la salida el número de habitaciones tendremos que crear un nuevo campo, `Field.Store.YES`. Para ordenar la salida por este campo deberemos incluir un campo adicional de tipo `SortedNumericDocValuesField`

Indexando un Document con IndexWriter

```
private IndexWriter writer;  
...  
private void indexFile(File f) throws  
    Exception {  
    Document doc = getDocument(f);  
    writer.addDocument(doc);  
}
```

Indexar un directorio y cerrar el índice

```
private IndexWriter writer;
...
public int index(String dataDir, FileFilter filter)
                                throws Exception {
    File[] files = new File(dataDir).listFiles();
    for (File f: files) {
        if (... &&
            (filter == null || filter.accept(f))) {
            indexFile(f);
        }
    }
    return writer.numDocs();
    writer.close();
}
```

IndexWriter ... algunos métodos

- `addDocument (Document)`;
`addDocument(Document, Analyzer)`;
- `deleteDocuments(Term)`
`deleteDocuments(Query)`.
Term (par campo,valor), Query consulta...
- `updateDocument(Term, Document)`;
borra los documentos con el atributo Term y luego
añade el nuevo documento
- `Close()`; Los cambios se hacen en memoria, y
cuando se cierran el índice se pasa a disco.

Sobre el Analizador

Analyzer

Analyzer no realiza Parser

- Parsing (análisis sintáctico del texto), .
 - ♦ Las aplicaciones que usen Lucene pueden indexar documentos en varios formatos – HTML, XML, PDF, Word, ... –
 - ♦ **Lucene sólo trabaja sobre texto.**
 - ♦ NO realiza el Parser de los documentos, y es responsabilidad de las aplicaciones utilizar el parser adecuado para convertir los documentos en texto:

Analyzer y Tokenizer

- Estrechamente relacionados....
- El **Analyzer** es responsable de toda la tarea de gestión de tokens del texto de entrada, mientras que el **Tokenizer** sólo es responsable de descomponer el texto de entrada en tokens.
- Muy probablemente, los tokens creados por el **Tokenizer** serán modificadas o incluso omitirse por el analizador (a través de uno o más TokenFilters) antes de ser devuelto.

Analyzer

- Importante: Las aplicaciones generalmente no invocan análisis - Lucene lo hace por ellos:
 - ♦ En indexación, como consecuencia de `addDocument` (doc), se llama al analizador elegido para la indexación.
 - ♦ En la búsqueda, un `QueryParser` podrá invocar el analizador durante el análisis (en algunas consultas el análisis no se realiza, por ejemplo, consultas comodín).
- Sin embargo, una aplicación podría invocar Análisis de cualquier texto para las pruebas o para cualquier otro propósito:

Tokenizacion

- Tokenizar: Descomponer el texto en un conjunto de pequeños componentes, aunque a veces es necesario realizar un análisis en profundidad.
 - ♦ Pre-Tokenizacion: Salto de marcas HTML, salto de cadenas arbitrarias, etc.
 - ♦ Post-Tokenización:
 - ➔ Stemming – Reemplazar las palabras por sus raíces. p.e. "bikes" -> "bike"
 - ➔ Eliminación de palabras vacías – "the", "and" y "a"
 - ➔ Normalización – quitar acentos y similares
 - ➔ Añadir sinónimos en la posición actual

Tokenizar

- Disponible en el paquete [Analysis](#) que proporciona el mecanismo para convertir cadenas [Strings](#) y lectores ([Readers](#)) en tokens que pueden ser indexados por Lucene.

Analyzer más comunes

- (1) `WhitespaceAnalyzer`

Divide la entrada considerando los espacios en blanco.

- (2) `SimpleAnalyzer`

Divide considerando cualquier caracter que no sea letra, y transforma en minúscula.

- (3) `StopAnalyzer`

Igual que `SimpleAnalyzer`, pero también elimina las palabras vacías

- (4) `KeywordAnalyzer`, considera el stream como un único token (pe. Nombres productos, ids, ..)

- Paquetes org.apache.lucene.analysis.core

Analyzer

Analyzer más comunes

- (5) StandardAnalyzer

Más sofisticado, utiliza gramáticas. , URLs y direcciones de email se dividen en tokens:

ej:

→ `http://decsai.ugr.es` -> `http` `decsai.ugr.es`

→ `jhg@decsai.ugr.es` -> `jhg` `decsai.ugr.es`

- (6) UAX29URLEmailAnalyzer

Diseñado para trabajar con url y emails

- (7) SpanishAnalyzer

Para el castellano

- Paquetes

- (5,6) `org.apache.lucene.analysis.standard`

- (7) `org.apache.lucene.analysis.es`

Ejemplos

- “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]

Múltiples Analyzers

- Normalmente, sólo se invoca a un tipo de Analyzer para todo el documento ... Sin embargo, nos puede interesar utilizar un analizador para cada campo

PerFieldAnalyzerWrapper varios =

```
    new PerFieldAnalyzerWrapper(new StandardAnalyzer());  
varios.addAnalyzer("nombre", new KeywordAnalyzer());  
varios.addAnalyzer("email",  
                    new UAX29URLEmailAnalyzer());
```

- En este caso, StandardAnalyzer es utilizado por defecto, excepto en "nombre" and "email",
- PerFieldAnalyzerWrapper puede utilizarse como cualquier analizador para indexado y consulta.

Dentro de Analyzer

- Hay cuatro clases principales
- **Analyzer** - Un analizador es responsable de la construcción de un `TokenStream` (secuencia de tokens) que puede ser usado en los procesos de indexación y búsqueda.
- **CharFilter** - `CharFilter` extiende `Reader` para realizar sustituciones de pre-tokenización, borrados y / o inserciones en el texto de un `Reader` de entrada
- **Tokenizer** - es un `TokenStream` y descompone el texto entrante en tokens. En la mayoría de los casos, un analizador usará un `Tokenizer` como el primer paso en el proceso de análisis.
- **TokenFilter** - es responsable de la modificación de los tokens que se han creado por el `Tokenizer`. Modificaciones más comunes realizadas por un `TokenFilter` son: eliminación, derivados, inclusión de sinónimos. No todos los analizadores requieren `TokenFilters`.

TokenStream

TokenStream enumera una secuencia de tokens, bien sobre los campos (**Fields**) de un documento o sobre el texto de una consulta.

Es una clase abstracta; las subclases concretas son:

- ♦ **Tokenizer**, un **TokenStream** cuya entrada es **Reader**
- ♦ **TokenFilter**, un **TokenStream** con entrada otro **TokenStream**.

Tokenizer y TokenFilter

- Tokenizer

- WhitespaceTokenizer
- KeywordTokenizer
- LetterTokenizer
- StandardTokenizer
- ...

TokenFilter

- LowerCaseFilter
- StopFilter
- PorterStemFilter
- ASCIIFoldingFilter
- StandardFilter
- ...

Workflow

1. Instanciar un objeto `TokenStream/TokenFilter` que añade/lee atributos a/de `AttributeSource`.
2. Hacer una llamada a `reset()`.
3. El objeto toma los atributos del stream y almacena referencias a todos los atributos que quiere acceder.
4. Se hace una llamada a `incrementToken()` hasta que devuelve false, analizando cada uno de los atributos.
5. Se hace la llamada a `end()`
6. Se cierra `close()` para liberar recursos cuando se termina de utilizar el `TokenStream`.

Llamando al Analyzer

```
Version matchVersion = Version.LUCENE_XY; //
Analyzer analyzer = new StandardAnalyzer(matchVersion);
TokenStream ts = analyzer.tokenStream(null, new
    StringReader("some text goes here"));
OffsetAttribute offsetAtt =
    ts.addAttribute(OffsetAttribute.class);
try {
    ts.reset(); // stream to the beginning. (Required)
    while (ts.incrementToken()) {
        // Use AttributeSource.reflectAsString(boolean)
        // for token stream debugging.
        System.out.println("token: " +
            ts.reflectAsString(true));
        System.out.println("att " +
            ts.getAttribute(CharTermAttribute.class));
        System.out.println("token start offset: " +
            offsetAtt.startOffset());
        System.out.println(" token end offset: " +
            offsetAtt.endOffset());
    }
    ts.end(); // Perform end-of-stream operations,
} finally {
    ts.close(); // Release resources associated
```

Código de Ejemplo para indexación

[IndexarFichero.java](#)

Lo podéis probar con Obras Completas de Cervantes

[AnalyzerUtils.java](#)

Casos Especiales

- Fechas y Numeros necesitan un tratamiento especial para poder ser utilizadas en la busqueda
 - [org.apache.lucene.document.DateTools](#)
 -
- Para tratar números Lucene los considera como cadenas, salvo que se almacene como campo NUMERICO, por tanto debemos asegurarnos que el analizador no eliminen los números

[WhitespaceAnalyzer](#), [StandarAnalyzer](#), [UAX..](#), [Spanish](#)

Si se indexan como String Lucene usa orden lexicografico

20 < 7 < 709 < 71
007 < 020 < 071 < 709 !!

- Ver AplicacionDate



LUKE

Luke - Lucene Index Toolkit

- Podemos utilizar la herramienta Luke para acceder a un índice existente
- No esta totalmente actualizada, pero podemos encontrar distintas versiones por la web
 - ♦ <https://code.google.com/p/luke/>
 - ♦ <https://java.net/projects/opengrok/downloads>
 - ♦ Decsai.ugr.es
- Se ejecuta utilizando
 - ♦ `Java -jar lukeall-xxx.jar`
 - ♦ Donde xxx representa la versión.

Luke

Luke accede a un indice Lucene existente.

- ♦ Podemos
 - Navegar los documentos por numero o termino
 - Ver documentos
 - Recuperar una lista ordenada de los terminos mas frecuentes
 - Ejecutar búsquedas
 - Analizar los resultados
 - Etc.

Luke

Overview Documents Search Files Plugins

Index name: ?

Number of fields: ?

Number of documents: ?

Number of terms: ?

Has deletions?: ?

Index version: ?

Last modified: ?

Directory implementation: ?

Re-open

Close

Select fields from the list below, and press button to view top terms in these fields. No selection means all fields.

Available Fields:

Top ranking terms. (Right-click for more options)

No Rank Field Text

Show top terms >>

Number of top terms:

50

Hint: use Shift-Click to select ranges, or Ctrl-Click to select multiple fields (or unselect all).

Path to Index directory:

Path: C:\Documents and Settings\jhg\Mis documentos\Ne... Browse...

☐ Open in Read-Only mode

☐ Force unlock, if locked

☐ (Expert) Use MMapDirectory


☐ (Expert) Load into RAMDirectory

OK Cancel

Index name: ?

Luke - Lucene Index Toolbox, v 0.8.1 (2008-02-13)

File Tools Settings Help

 Overview
  Documents
  Search
  Files
  Plugins

Index name: **C:\Documents and Settings\jhg\Mis documentos\NetBeansProjects\PrimeraAplicacion\Dir_Index**

Number of fields: **4**

Number of documents: **2**

Number of terms: **10**

Has deletions?: **No**

Index version: **1210937412022**

Last modified: **Fri May 16 13:48:41 CEST 2008**

Directory implementation: **org.apache.lucene.store.FSDirectory**

Select fields from the list below, and press button to view top terms in these fields. No selection means all fields.

Available Fields:

<empresa>

<nombre>

<telefono>

<web>

Show top terms >>

Number of top terms:

50

Hint: use Shift-Click to select ranges, or Ctrl-Click to select multiple fields (or unselect all).

Top ranking terms. (Right-click for more options)

No	Rank	Field	Text
1	2	<nombre>	juan
2	1	<nombre>	huate
3	1	<telefono>	945334532
4	1	<nombre>	luis
5	1	<nombre>	perez
6	1	<telefono>	54+945334532
7	1	<empresa>	ugr
8	1	<web>	http://www.klm.es
9	1	<web>	http://www.uno.es/tres.html
10	1	<empresa>	klm

Index name: **C:\Documents and Settings\jhg\Mis documentos\NetBeansProjects\PrimeraAplicacion\Dir_Index**

Enter search expression here:

juan

Analyzer to use for query parsing:

NOTE: use fully-qualified class name here.

org.apache.lucene.analysis.standard.StandardAnalyzer

Default field is: nombre SnowballAnalyzer name:

Query details: Update Explain structure

nombre:juan

Results: (Hint: Double-click on results to display all fields)

#	Score	Doc. Id	empresa	nombre	telefono
0	0,3716	0			
1	0,2973	1			

Query Structure

Structure of the query:

TermQuery: boost=1,0000
Term: field='nombre' text='juan'

OK

Ficheros Invertidos de Lucene

Doc 1:
Penn State
Football ...
football

Doc 2:
Football
players ...
State

Posting id	word	doc	offset
1	football	Doc 1	3
		Doc 1	67
		Doc 2	1
2	penn	Doc 1	1
3	players	Doc 2	2
4	state	Doc 1	2
		Doc 2	13

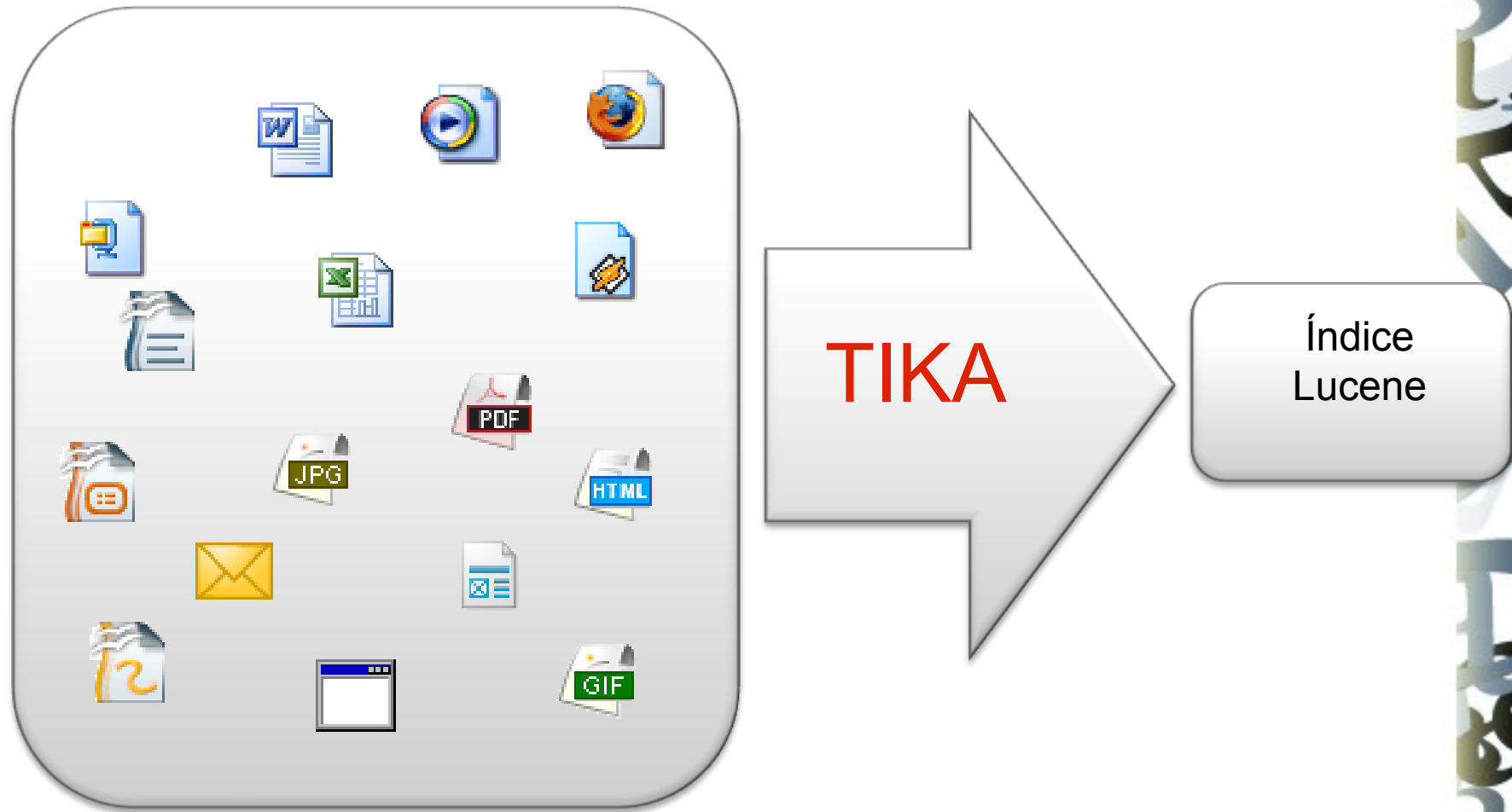


Posting
Table

Tabla de ocurrencias

- Tabla de ocurrencias permite un mecanismo rápido de búsqueda
 - Clave: word
 - Valor: posting id, satellite data (#df, offset, ...)
- Lucene implementa la tabla de ocurrencia con una tablas hash de Java
 - Función Hash depende de JVM
 - $hc2 = hc1 * 31 + nextChar$
- Se utiliza en
 - Indexación: insercion (terminos nuevosw), update (terminos existentes)
 - Búsqueda

Gestión de distintos formatos de texto:



Tambien directamente desde Lucene HTMLParser

File f

```
Document doc = new Document();  
doc.add(new Field("path", f.getPath().replace(dirSep, '/').....);  
doc.add(new Field("modified",  
    DateTools.timeToString(f.lastModified(),.....);  
doc.add(new Field("uid", uid(f),....);
```

```
FileInputStream fis = new FileInputStream(f);  
HTMLParser parser = new HTMLParser(fis);
```

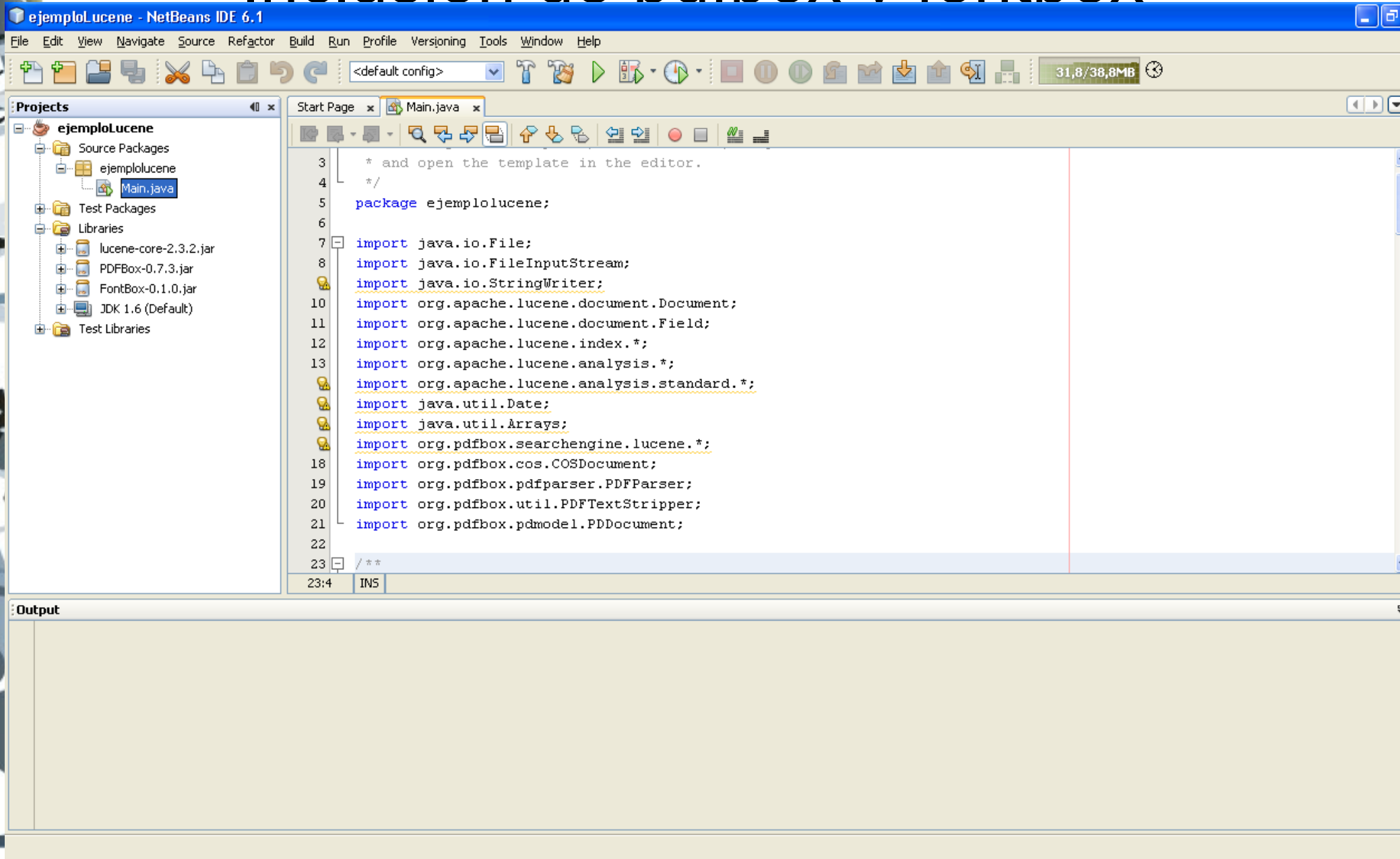
```
doc.add(new Field("contents", parser.getReader()));  
doc.add(new Field("summary", parser.getSummary(),  
    Field.Store.YES, Field.Index.NO));  
doc.add(new Field("title", parser.getTitle(),  
    Field.Store.YES, Field.Index.TOKENIZED));  
return doc;
```

- La mayoría de las páginas webs están en HTML
- Podemos encontrar distintos parses
 - ♦ Jtidy
 - ♦ NekoHTML
 - ♦

Indexando un PDF

- Portable Document Format (PDF) está ampliamente distribuido
- Permite trabajar con figuras, colores, hiperenlaces, ...
- ¿Cómo podemos trabajar con documentos PDF?
 - ♦ Utilizaremos PDFBox
 - ♦ Lo debemos de incluir en las librerías
 - ♦ También hará falta incluir FontBox

Inclusion de pdfbox v fontbox





PDFBox

Home**About**

- [Index](#)
- [Download](#)
- [Nightly Build](#)
- [Forums](#)
- [Issues](#)
- [SourceForge](#)
- [References](#)
- [Donations](#)
- [License](#)
- [Release Notes](#)

Command Line Utilities**Developers Guide**

PDFBox - Java PDF Library

- [Introduction](#)
- [Features](#)



PDF

Introduction

PDFBox is an open source Java PDF library for working with PDF documents. This project allows creation of new PDF documents, manipulation of existing documents and the ability to extract content from documents. PDFBox also includes several command line utilities.

Features

- PDF to text extraction
- Merge PDF Documents
- PDF Document Encryption/Decryption
- Lucene Search Engine Integration
- Fill in form data FDF and XFDF
- Create a PDF from a text file
- Create images from PDF pages

Clases de PDFBox

- **PDFParser**

Es la clase que contiene el parser para poder analizar documentos pdf.

Dado un documento PDF, el método **parse()** sera el encargado de parsearlo

```
Import org.pdfbox.pdfparser.PDFParser;
```

- **COSDocument**

Es la representacion en memoria del documento pdf

Es necesario cerrar (close()) el objeto cuando hayamos finalizado⁷⁷ de utilizarlo.

```
Import org.pdfbox.cos.COSDocument;
```


- **PDDocument** Clases de PDFBox

una representación en memoria (a mas alto nivel) del documento pdf.

Para ser valido, un documento debe tener al menos una pagina.

```
import org.pdfbox.pdmodel.PDDocument;
```

- **PDFTextStripper**

Esta clase toma un documento PDF y extrae todo el texto, pasando por alto su formato.

Es responsabilidad de los usuarios de la clase verificar que un determinado usuario tiene los permisos correctos para extraer el texto de documento PDF.

Clases de PDFBox

```
PDFParser parser = new PDFParser(fis);
parser.parse();
COSDocument cos = parser.getDocument();
PDDocument pdd = new PDDocument(cos);
if (pdd.isEncrypted()) {
    System.out.println("Error");
} else {
    System.out.println("No encriptado");
}
String cont_texto = null;
PDFTextStripper stripper = new PDFTextStripper();
String texto = stripper.getText(pdd);
System.out.println("Texto:"+texto);
Document lucenedoc= new Document();
lucenedoc.add(new
Field("contenido",texto,Field.Store.NO,Field.Index.TOKENIZED));
```

Para XML ...

- Podemos utilizar la librería JDOM, donde los documentos se representan mediante la clase **`org.jdom.Document`**
- Se pueden construir directamente, o bien a partir de un fichero, stream o url

Y AHORA

La búsqueda