



UNIVERSIDAD  
DE GRANADA

PRÁCTICA FINAL  
RECUPERACIÓN DE LA  
INFORMACIÓN

Juan Carlos González Quesada y Pedro Jiménez Alférez

*Lucene*

## Contenido

|                                                      |    |
|------------------------------------------------------|----|
| Indexar la colección o añadir un nuevo fichero ..... | 2  |
| Creación del índice y las facetas.....               | 3  |
| Creación de los archivos a indexar.....              | 3  |
| Características del Índice.....                      | 3  |
| Características de las facetas .....                 | 4  |
| Indexación .....                                     | 4  |
| Creación de las consultas .....                      | 5  |
| Búsqueda por Facetas .....                           | 7  |
| Mostrar los resultados .....                         | 8  |
| Interfaz gráfica.....                                | 9  |
| Trabajo en grupo .....                               | 10 |
| ¿Cómo se ejecuta?.....                               | 10 |

## Indexar la colección o añadir un nuevo fichero

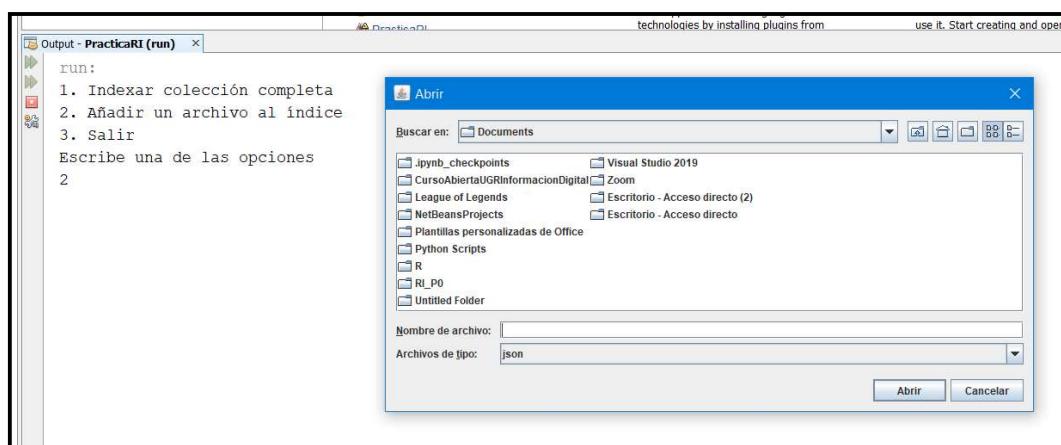
Para la construcción del Índice, el administrador del sistema tendrá dos opciones:

- **Indexación de toda la colección:** Desde la carpeta “*pdf\_json*” en donde se encontrarán todos los archivos *json*, se realizará la carga e indexación de todos los ficheros.
- **Añadir un fichero al índice:** desde cualquier parte de su ordenador, seleccionará el archivo que quiere añadir y el programa lo indexará y almacenará en la carpeta “*pdj\_json*”. De esta forma, si ocurre un error se podrán reindexar todos de nuevo, incluidos los añadidos posteriormente:



```
case 2:
    final JFrame frame = new JFrame();
    frame.setVisible(true);
    frame.setExtendedState(JFrame.ICONIFIED);
    frame.setExtendedState(JFrame.NORMAL);
    JFileChooser fc = new JFileChooser();
    fc.addChoosableFileFilter(new FileNameExtensionFilter("json", "json"));
    fc.setAcceptAllFileFilterUsed(false);
    if(JFileChooser.APPROVE_OPTION==fc.showOpenDialog(null)){
        frame.setVisible(false);
        archivoAAnadir = fc.getSelectedFile();
        byte[] jsonData = Files.readAllBytes(Paths.get(archivoAAnadir.getAbsolutePath()));
        ObjectMapper objectMapper = new ObjectMapper();
        JsonNode rootNode = objectMapper.readTree(jsonData);
        Ficovid archivo = new Ficovid(rootNode);
        Indice indiceAux = new Indice();
        indiceAux.indexarNuevoDocumento(archivo);
        archivoAAnadir.renameTo(new File("../pdf_json/" + archivoAAnadir.getName()));
    }
    else{
        System.out.println("Nada fue seleccionado. Por favor, inténtelo de nuevo");
    }
    frame.dispose();
```

Para este proceso, hemos creado un *JFileChooser*, que controlará el archivo que seleccionará. Hemos restringido que sólo pueda seleccionar aquellos archivos que tengan la extensión “*.json*”, en caso de que suba uno y no contenga la estructura que nosotros tenemos en el programa, se añadirá un documento con todo desconocido. Nuestro programa, permite que el proceso de añadir los archivos sea totalmente transparente al usuario, es decir, si se está ejecutando alguna consulta y se quiere añadir algún archivo al índice, no será necesario que el usuario cierre el programa o suspender temporalmente



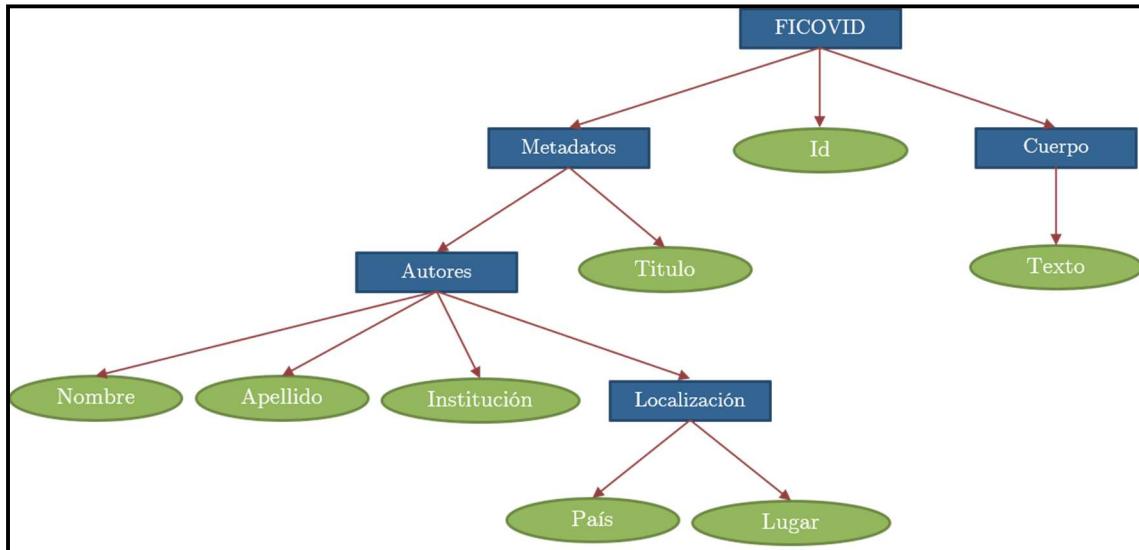
Buscador de archivos tras la selección de la opción 2

las búsquedas. Pues que cuando finalice el proceso de indexación y el usuario realice una nueva consulta, dicho documento ya podrá ser usado como resultado para la consulta.

## Creación del índice y las facetas

### Creación de los archivos a indexar

Para cada archivo de la carpeta pdf\_json, creamos un objeto del tipo Ficovid que está compuesto por:



Árbol de los objetos Ficovid

Los rectángulos representan clases y las elipses atributos.

### Características del Índice

Dentro de esta clase, lo primero es asignar a cada campo su analizador correspondiente: a Autores, países, universidad y lugar, le asignamos el SimpleAnalyzer() mientras que para los títulos, le asignamos el StandardAnalyzer() y para el texto, usamos el EnglishAnalyzer(). Una vez asignados los analizadores, establecemos los directorios donde se van a crear los distintos índices y valores mediante la función FSDirectory.open(rutaX).

```
//Asignamos a cada campo su analizador correspondiente
Analyzer analizador = new SimpleAnalyzer();
analyzerPerfield.put("autores", analizador);
analyzerPerfield.put("paises", analizador);
analyzerPerfield.put("titulo", new StandardAnalyzer());
analyzerPerfield.put("universidad", analizador);
analyzerPerfield.put("lugar", analizador);
analyzerPerfield.put("texto", new EnglishAnalyzer());

miAnalyzer = new PerFieldAnalyzerWrapper(new WhitespaceAnalyzer(), analyzerPerfield);

//Establecemos los directorios donde se crearán los distintos indices y facetas
FSDirectory dir = FSDirectory.open(Paths.get(rutaIndice));
FSDirectory taxoDir = FSDirectory.open(Paths.get(rutaFaceta));
IndexWriterConfig config = new IndexWriterConfig(miAnalyzer);
config.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
```

## Características de las facetas

Para los campos que vamos a utilizar para la búsqueda de facetas, le aplicamos la función **setMultiValued(campo,true)**, para que puedan tener distintos valores para cada faceta.

```
//Configuramos el taxo para las facetas
fconfig = new FacetsConfig();
fconfig.setMultiValued("autores", true);
fconfig.setMultiValued("paises", true);
fconfig.setMultiValued("universidad", true);
fconfig.setMultiValued("lugar", true);
```

## Indexación

```
//Función para indexar documentos
public void indexarDocumentos( ArrayList<Ficovid> ficheros) throws IOException{
    int contador=0;
    for(Ficovid p : ficheros){
        contador++;
        Document doc = new Document();
        doc = new Document();
        doc.add(new StringField("id_json", p.getId(), Field.Store.YES));
        doc.add(new TextField("titulo", p.getMetadata().getTitulo(), Field.Store.YES));

        for(int i=0; i<p.getAutores().size();i++){
            doc.add(new TextField("autores", p.getAutores().get(i).toString(), Field.Store.YES));
            doc.add(new TextField("paises", p.getAutores().get(i).getLocalizacion().getPais(), Field.Store.YES));
            doc.add(new TextField("universidad", p.getAutores().get(i).getInstitucion(), Field.Store.YES));
            doc.add(new TextField("lugar", p.getAutores().get(i).getLocalizacion().getLugar(), Field.Store.YES));

            //Añadimos las facetas
            doc.add(new FacetField("autores", p.getAutores().get(i).toString()));
            doc.add(new FacetField("paises", p.getAutores().get(i).getLocalizacion().getPais()));
            doc.add(new FacetField("universidad", p.getAutores().get(i).getInstitucion()));
            doc.add(new FacetField("lugar", p.getAutores().get(i).getLocalizacion().getLugar()));
        }
        int longitud = p.getCuerpo().getTexto().length();
        doc.add(new SortedDocValuesField("sorted", new BytesRef(Integer.toString(10).getBytes())));
        doc.add(new TextField("texto", p.getCuerpo().getTexto(), Field.Store.YES));

        doc.add(new NumericDocValuesField("rangos", longitud));

        doc.add(new IntPoint("longitud", longitud));
        doc.add(new StoredField("longitud", longitud));
        doc.add(new FacetField("longitud", Integer.toString(longitud)));
        writer.addDocument(fconfig.build(taxoWriter, doc));

        System.out.println("Contador del indice: " + contador);
    }

    closeIndex();
}
```

En esta función, debemos de pasar un **ArrayList** con todos los ficheros. A continuación, los recorremos y creamos una variable local para añadir el ID\_JSON y el título del documento.

Una vez realizada esta parte y dentro de un bucle, creamos otro que contengan todos los autores, países, universidades y lugaeres. Además de los datos ya introducidos, agregamos todos los datos de las facetas. Para ello usamos la función **FacetField()** que introduce toda la información de un campo para las facetas y añadiremos las mismas facetas que campos indexados.

Después de recorrer los autores, añadimos el texto que contiene el fichero. Por último, añadimos la longitud del fichero a nuestra variable local y creamos la variable sort, que servirá para ordenar por relevancia los documentos.

## Creación de las consultas

- **Consulta por términos:** Es la búsqueda más similar al buscador de Google. La función que hemos creado recibe el campo sobre el que se va a buscar y el conjunto de términos.

```
public static String buscadorPorTérmino(Analyzer analyzer, Similarity similarity, String Campo, String termino){  
    IndexReader reader = null;  
    String solución = "";  
    try{  
        reader = DirectoryReader.open(FSDirectory.open(Paths.get(rutaIndice)));  
        IndexSearcher searcher = new IndexSearcher(reader);  
        searcher.setSimilarity(similarity);  
        QueryParser parser = new QueryParser(Campo, new SimpleAnalyzer());  
  
        Query query = null;  
        try{  
            query = parser.parse(termino);  
        }catch(org.apache.lucene.queryparser.classic.ParseException e){  
            System.out.println("Error en la cadena de consulta");  
        }  
        if(query!=null){  
            System.out.println("La consulta es " + query.toString());  
            TopDocs results = searcher.search(query, 10);  
            ScoreDoc[] hits = results.scoreDocs;  
  
            solución = mostrarResultadosBusquedas(hits, searcher, results);  
        }  
        else{  
            solución="No hay";  
        }  
    }  
}
```

El analizador que se usa es el *SimpleAnalyzer*, se obtienen solo 10 documentos y los resultados se muestran con la función *mostrarResultadosBusquedas()* que explicaremos más adelante.

- Consulta por Entero:

```
public static String buscadorPorEntero(Analyzer analyzer, Similarity similarity, String Campo, int termino){
IndexReader reader = null;
String solucion = "";
int entero1, entero2;
try{
    if(termino>100){
        entero1=termino-100;
        entero2=termino+100;
    }
    else{
        entero1=termino/2;
        entero2=termino+100;
    }
    reader = DirectoryReader.open(FSDirectory.open(Paths.get(rutaIndice)));
    IndexSearcher searcher = new IndexSearcher(reader);
    searcher.setSimilarity(similarity);
    BufferedReader in = null;
    in = new BufferedReader(new InputStreamReader(System.in, StandardCharsets.UTF_8));
    Query query = null;
    //Busqueda por rango
    query = IntPoint.newRangeQuery("longitud", entero1,entero2);
    TopDocs results = searcher.search(query, 10);
    ScoreDoc[] hits = results.scoreDocs;
    solucion = mostrarResultadosBusquedas(hits, searcher, results);
}
}
```

Aquí pensamos que lo mejor es dar un resultado de longitud aproximado. Para evitar que por un número no se encuentren documentos similares, es decir, si el usuario introduce 456 y hay uno por 457, nosotros se lo mostramos. Para ello, la búsqueda se realiza por rango.

- Consulta Booleana:

```
campo1 = campo1.toLowerCase();
campo2 = campo2.toLowerCase();
consultal = consultal.toLowerCase();
consulta2 = consulta2.toLowerCase();

Query query1, query2;
Term termino = new Term(campo1, consultal);
query1 = new TermQuery(termino);
termino = new Term(campo2, consulta2);
query2 = new TermQuery(termino);
BooleanClause b1=null, b2=null;
```

Utilizamos dos términos en la consulta que pueden ser de diferente campo. Para construir la cláusula booleana, tendremos en cuenta ocho factores con respecto a lo que solicite el usuario.

- Que estén los dos términos: se construirá usando en ambas MUST.
- Que este uno de los dos términos: se usará SHOULD.
- Que no esté alguno de los dos términos: su usará la función de búsqueda booleana especial, en la que se dan los documentos en los que aparece solo uno de los términos y se unen los resultados.
- Que no esté ninguno de los dos: esta cláusula no se puede realizar porque Lucene no permite la doble negación.

- **Consulta por frases:** se introduce una frase y se busca en el documento.

```

reader = DirectoryReader.open(FSDirectory.open(Paths.get(rutaIndex)));
IndexSearcher searcher = new IndexSearcher(reader);
searcher.setSimilarity(similarity);
BufferedReader in = null;
in = new BufferedReader(new InputStreamReader(System.in, StandardCharsets.UTF_8));
Analyzer ana = new WhitespaceAnalyzer();
PhraseQuery.Builder builder = new PhraseQuery.Builder();
//Sirve para asignar el analizador que se usará en la consulta
if(campo.contentEquals("autores") || campo.contentEquals("titulo") || campo.contentEquals("paises") || campo.contentEquals("universidad")){
    ana = new SimpleAnalyzer();
}
else{
    if(campo.contentEquals("texto")){
        ana = new EnglishAnalyzer();
    }
}
//Sirve para coger los términos de la frase y convertirlos en un N-Grama (De String a Stream)
TokenStream stream = ana.tokenStream(null, consulta);
stream.reset();
while(stream.incrementToken()){
    String palabra = "";
    palabra+= stream.getAttribute(CharTermAttribute.class);
    Term termino = new Term(campo, palabra);
    builder.add(termino);
}

```

Primero tenemos que convertir el string que se pasa como parámetro en un conjunto de tokens que se buscarán. Para realizar la búsqueda, determinaremos su analizador según el campo que se esté buscando.

## Búsqueda por Facetas

Para este procedimiento y junto a la interfaz, ofrecemos varias visualizaciones del conjunto de las facetas.

- Ver todas las facetas de todos los documentos.
- Ver las facetas de un solo campo de todos los documentos.
- Ver todas las facetas aplicadas a la consulta realizada por el usuario.
- Ver las facetas de un solo campo aplicadas a la consulta realizada por el usuario.

```

if(!consulta){ //Se muestran todas las facetas disponibles
    query = new MatchAllDocsQuery();
    TopDocs tdc = FacetsCollector.search(searcher, query, 10, fc);
    Facets facetas = new FastTaxonomyFacetCounts(taxoReader, fconfig, fc);
    List<FacetResult> TodasDims = facetas.getAllDims(10);
    System.out.println("Categorías totales " + TodasDims.size());
    salida="";
    LongRange[] rangos = new LongRange[6];
    rangos[0] = new LongRange("1-1000", 1L, true, 1000L, false);
    rangos[1] = new LongRange("1000-5000", 1000L, true, 5000L, false);
    rangos[2] = new LongRange("5000-10000", 5000L, true, 10000L, false);
    rangos[3] = new LongRange("10000-25000", 10000L, true, 25000L, false);
    rangos[4] = new LongRange("25000-50000", 25000L, true, 50000L, false);
    rangos[5] = new LongRange("50000-100000", 50000L, true, 100000L, true);
    FacetsCollector fcRango = new FacetsCollector();
    FacetsCollector.search(searcher, new MatchAllDocsQuery(), 10, fcRango);
    LongRangeFacetCounts facetillas = new LongRangeFacetCounts("rangos", fcRango, rangos);
    List<FacetResult> TODAS = facetillas.getAllDims(10);
    if(todosCampos){
        for(FacetResult fr: TodasDims){
            salida += "Categoria " + fr.dim + "\n";
            for(LabelAndValue lv: fr.labelValues){
                salida += "    Etiqu: " + lv.label + " valor (#n->" + lv.value + "\n";
            }
        }
        for(FacetResult fr: TODAS){
            salida += "Categoria " + fr.dim + "\n";
            for(LabelAndValue lv: fr.labelValues){
                salida += "    Etiqu: " + lv.label + " valor (#n->" + lv.value + "\n";
            }
        }
    }
}

```

Se establecen los rangos para la búsqueda por facetas. Si quiere ver las de un campo concreto se le añade en el for con un if que solo mostrará las que cumplen la condición.

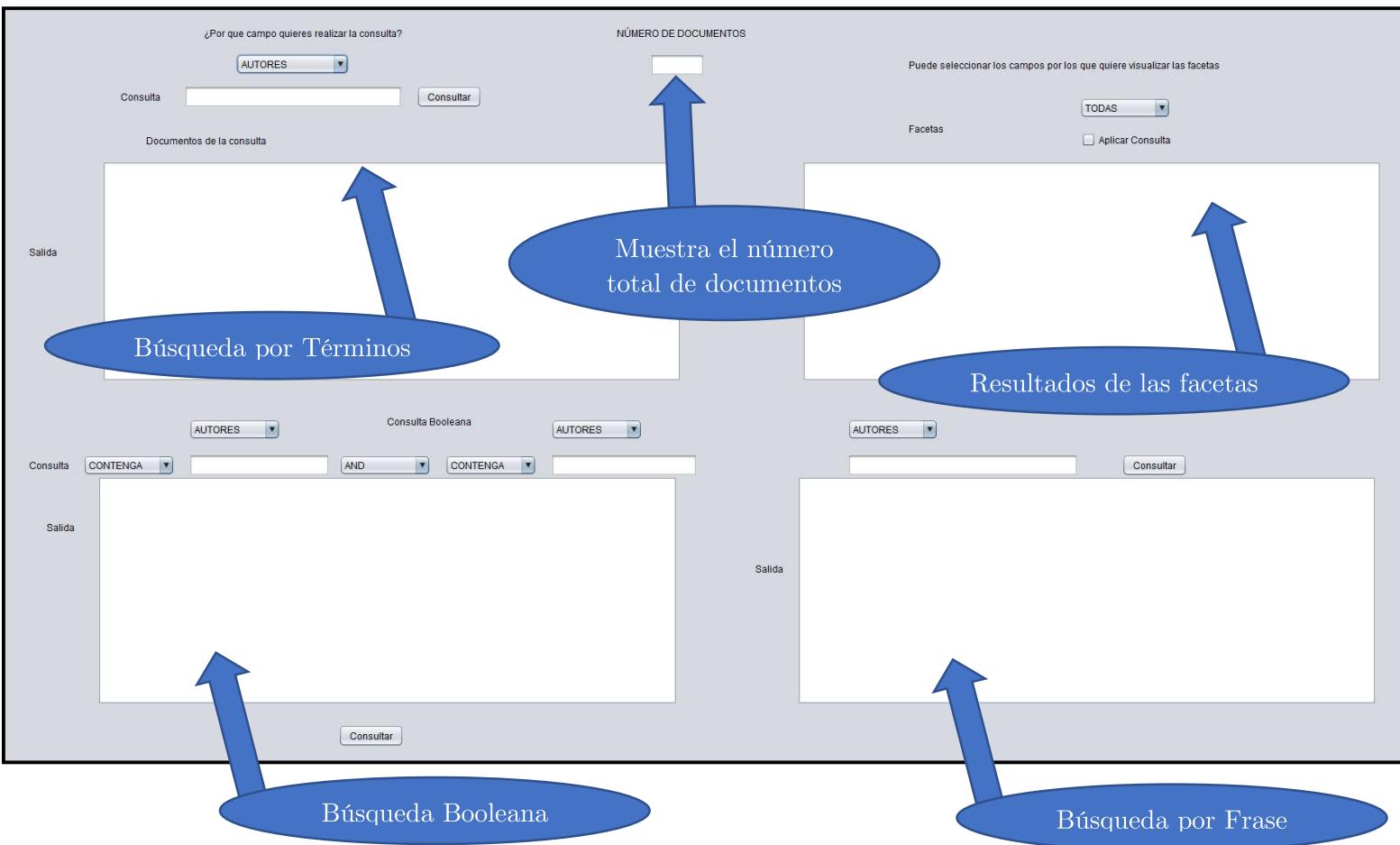
## Mostrar los resultados

Para visualizar los resultados, usaremos la función mostrarResultados() que devuelve un string con los diez documentos más relevantes para la consulta.

```
public static String mostrarResultadosBusquedas(ScoreDoc[] hits, IndexSearcher searcher, TopDocs results) throws IOException{
    TotalHits numTotalHits = results.totalHits;
    System.out.println(numTotalHits.value + " documentos encontrados");
    nDocs = (int) numTotalHits.value;
    String resultado="";
    for(int j = 0; j < hits.length; j++){
        Document doc = searcher.doc(hits[j].doc);
        String cuerpo = doc.get("titulo");
        String texto = doc.get("texto");
        if(texto.length()>200){
            texto= texto.substring(0,200);
        }
        String[] autores = doc.getValues("autores");
        String[] paises = doc.getValues("paises");
        String lon = doc.get("longitud");
        String[] uni = doc.getValues("universidad");
        String[] lugar = doc.getValues("lugar");
        String univer="";
        String lugares="";
        String autor = "", pais="";
        int aux = j+1;
        resultado += "Documento: " + aux +"\n";
        resultado += "Titulo: " + cuerpo + "\n";
        resultado += "Longitud: " + lon + "\n";
        for(int k = 0; k < autores.length; k++){
            if(k==autores.length-1)
                autor += autores[k];
            else
                autor += autores[k] + ", ";
        }
        paises = sinRepetir(paises);
        for(int k = 0; k < paises.length; k++){
            pais += paises[k] + ", ";
        }
        uni = sinRepetir(uni);
        for(int k = 0; k < uni.length; k++){
            univer += uni[k] + ", ";
        }
        lugar = sinRepetir(lugar);
        for(int k = 0; k < lugar.length; k++){
            lugares += lugar[k] + ", ";
        }
        resultado += "Universidad: " + univer + "\n";
        resultado += "Lugar: " + lugares + "\n";
        resultado += "Autores: " + autor + "\n";
        resultado += "Paises: " + pais + "\n";
        resultado+= "Texto: " + texto + "... \n";
        resultado += "\n";
    }
}
```

## Interfaz gráfica

La interfaz gráfica mostrará en la misma pantalla todas las búsquedas que se pueden realizar:



- **Búsqueda por términos.** Para este tipo de búsqueda, lo primero que tenemos que hacer es seleccionar el campo por el que queremos buscar. Una vez seleccionado, introducimos los términos que nos interesan y, a continuación, si pulsamos en el botón Consulta, nos muestra los 10 documentos con mejor score que contengan los términos introducidos. Además, mostramos el número de documentos que contienen esos términos en el centro de la interfaz. Los campos que podemos utilizar en esta búsqueda son: Texto, Lugar, Universidad, País, Autores, Longitud Aproximada o Título. Paralelamente, se mostrarán las facetas que están asociadas a los documentos. Nosotros podremos ver las facetas concretas seleccionando el campo y ver las facetas referentes a la consulta, clicando sobre “Aplicar consulta”.
- **Búsqueda por facetas.** Para esta búsqueda, podemos obtener distintos resultados, podemos ver todas las facetas de todos los documentos disponibles o podemos ver las de un solo campo. Además, al marcar la opción relacionada con la consulta, podemos observar todas las facetas aplicadas a la consulta realizada por el usuario o las facetas o solo las facetas de un campo aplicado a la consulta.

introducida. Los campos que podemos utilizar en esta búsqueda son: Texto, Lugar, Universidad, País, Rangos, Autores, Longitud Aproximada o Título.

- **Búsqueda booleana.** En esta opción, obtenemos los documentos dependiendo de cómo tratemos los campos. Para ello usamos las expresiones lógicas AND y OR para mostrar los documentos que contengan si o si ambos términos o que solo contengan uno de ellos. Además, a cada término puedes indicar que esté o no esté en la consulta mediante las palabras CONTENGA o NO\_CONTENGA. Los campos que podemos utilizar en esta búsqueda son: Texto, Lugar, Universidad, País, Autores.
- **Búsqueda por frases.** Esta búsqueda es parecida a la antes mencionada de términos, sin embargo, para este tipo de investigación, requiere que las palabras introducidas, deben de ser continuas en el texto, es decir, requiere de una frase y no de palabras sueltas. Autores, Título, Texto, Universidad

## Trabajo en grupo

Esta práctica junto a la memoria la hemos realizado conjuntamente quedando para su realización usando la plataforma de Google Meet.

## ¿Cómo se ejecuta?

Para indexar la colección completa, será necesario tener los archivos en la carpeta pdf\_json. Se ejecutará con la opción ‘Run File’ la clase indexar y se seleccionará la opción1. Tardará sobre 11 minutos en realizar el proceso entero.

Para probar el programa basta con darle a la flecha verde y empezar a realizar consultas.

## Librerías utilizadas (JAR)

<https://mvnrepository.com/artifact/commons-beanutils/commons-beanutils/1.9.3>

<https://mvnrepository.com/artifact/commons-collections/commons-collections/3.2.2>

<https://mvnrepository.com/artifact/org.apache.commons/commons-lang3/3.6>

<https://mvnrepository.com/artifact/commons-logging/commons-logging/1.2>

<https://mvnrepository.com/artifact/org.apache.commons/commons-text/1.1>

<https://mvnrepository.com/artifact/com.carrotsearch/hppc/0.8.1>

<https://mvnrepository.com/artifact/org.apache.lucene/lucene-core/8.0.0>

<https://mvnrepository.com/artifact/org.apache.lucene/lucene-core/8.2.0>

<https://mvnrepository.com/artifact/org.apache.lucene/lucene-facet/8.2.0>

[https://lucene.apache.org/core/8\\_2\\_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html](https://lucene.apache.org/core/8_2_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html)

<https://mvnrepository.com/artifact/org.apache.lucene/lucene-sandbox/8.2.0>

[https://lucene.apache.org/core/8\\_2\\_0/queries/org/apache/lucene/queries/function/package-summary.html](https://lucene.apache.org/core/8_2_0/queries/org/apache/lucene/queries/function/package-summary.html)

<http://openjdk.java.net/jeps/263>

<https://mvnrepository.com/artifact/com.opencsv/opencsv/4.1>

<https://tika.apache.org/1.23/index.html>