COM-35702 Design and Engineering of Intelligent Information Systems

Engineering and Error Analysis with UIMA

ITAM

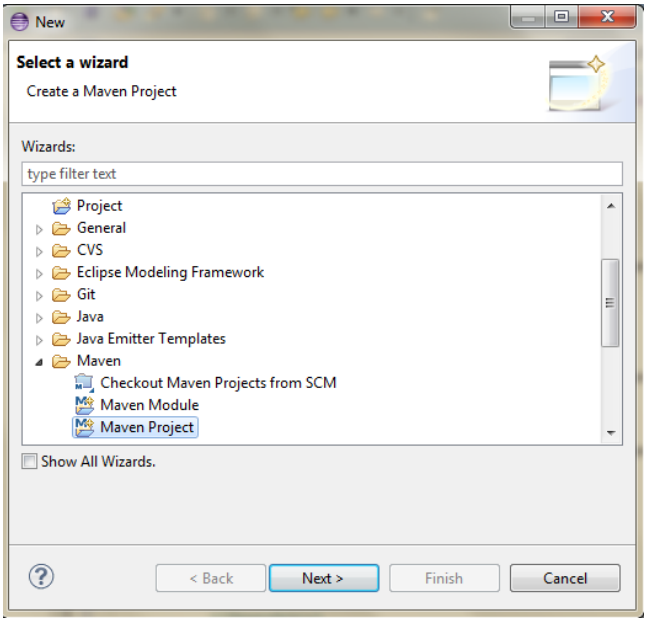
Tania Patiño

137175

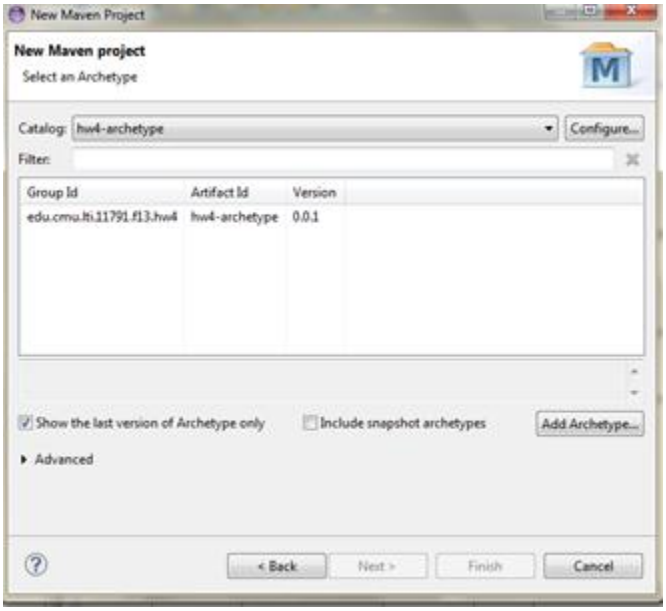
25-04-2014

Task 0.1 Creating Maven project from the archetype

Step1. Create a Maven Project



Step2. Activate the archetype for hw4, hw4-archetype

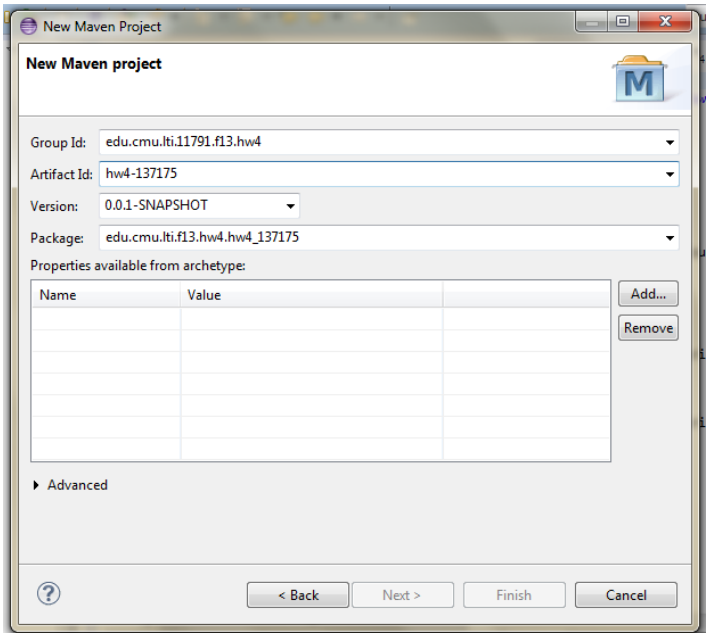


Step 3. Create the following Catalog URL

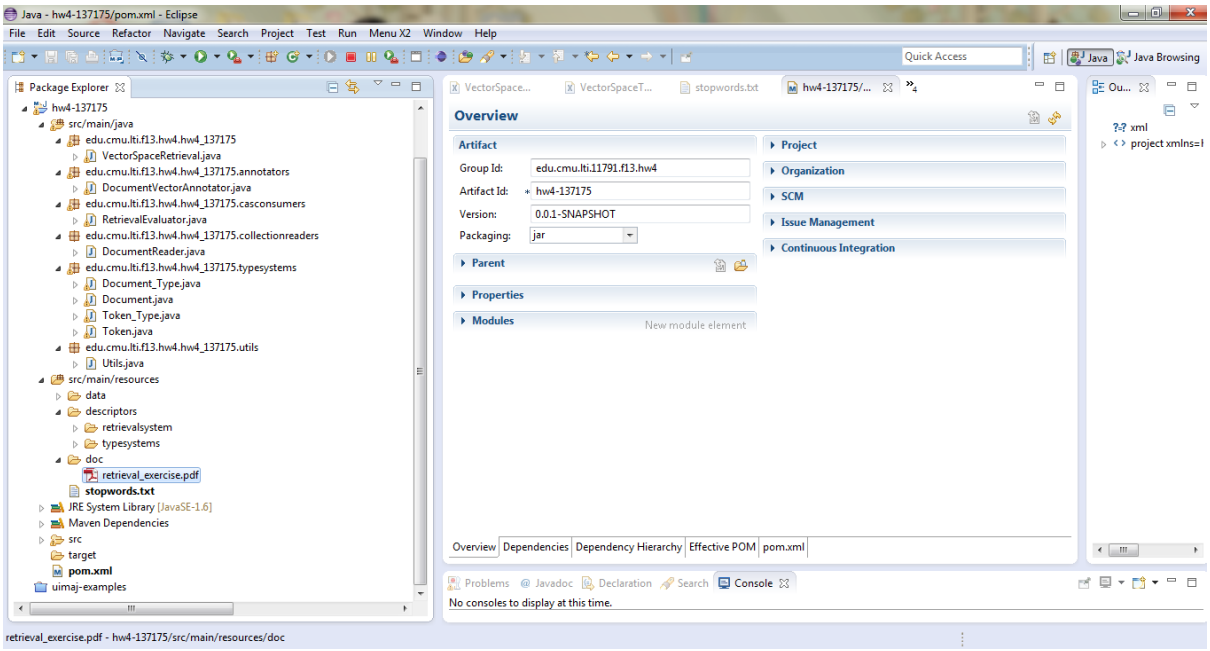
Also add the archetype for Homework: hw4-archetype

Group Id : edu.cmu.lti.11791.f13.hw4

Artifac Id: hw4-137175



Finally, this is the maven project with the archetype loaded and all the structure project created now.



There is a pom.xml file which one was edited in the SCM information. Making some comments in this file with <!- - code - ->

```
<?xmlversion="1.0"encoding="UTF-8"?>
<projectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XM
LSchema-instance"xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>edu.cmu.lti.11791.f13.hw4</groupId>
	<artifactId>hw4-137175</artifactId>
	<version>0.0.1-SNAPSHOT</version>

	<!-- <repositories>
		<repository>
			<id>oaqa</id>

	<url>http://mu.lti.cs.cmu.edu:8081/nexus/content/groups/course</url>
		</repository>
	</repositories>
	<distributionManagement>
		<repository>
			<id>deployment</id>

	<url>http://mu.lti.cs.cmu.edu:8081/nexus/content/repositories/course-
releases</url>
		</repository>
		<snapshotRepository>
			<id>snapshots</id>

	<url>http://mu.lti.cs.cmu.edu:8081/nexus/content/repositories/course-
snapshots</url>
```

```
        </snapshotRepository>
</distributionManagement> -->

<dependencies>
        <dependency>
                <groupId>org.apache.uima</groupId>
                <artifactId>uimaj-core</artifactId>
                <version>2.4.0</version>
        </dependency>
```

Task 1 Building Vector space Retrieval Model using UIMA

Task 1.1 Input

This is the file with the text collection:document.txt, that is located on: src/main/resources/data/document.txt

That contains the next questions:

qid=1 rel=99 Classical music is dying
qid=1 rel=0 Pop music has absorbed influences from most other genres of popular music
qid=1 rel=1 Classical music may never be the most popular music
qid=1 rel=0 Everybody knows classical music when they hear it
qid=2 rel=99 Energy plays an important role in climate change
qid=2 rel=0 Old wine and friends improve with age
qid=2 rel=0 With clothes the new are the best, with friends the old are the best
qid=2 rel=1 Climate change and energy use are two sides of the same coin.qid=3 rel=99 One's best friend is oneself
qid=3 rel=1 The best mirror is an old friend
qid=3 rel=0 My best friend is the one who brings out the best in me
qid=3 rel=0 The best antiques are old friends

Explanation about each section:

qid is a query ID,

relevant values: 1 indicates correct retrieval and 0 indicates wrong retrieval results.

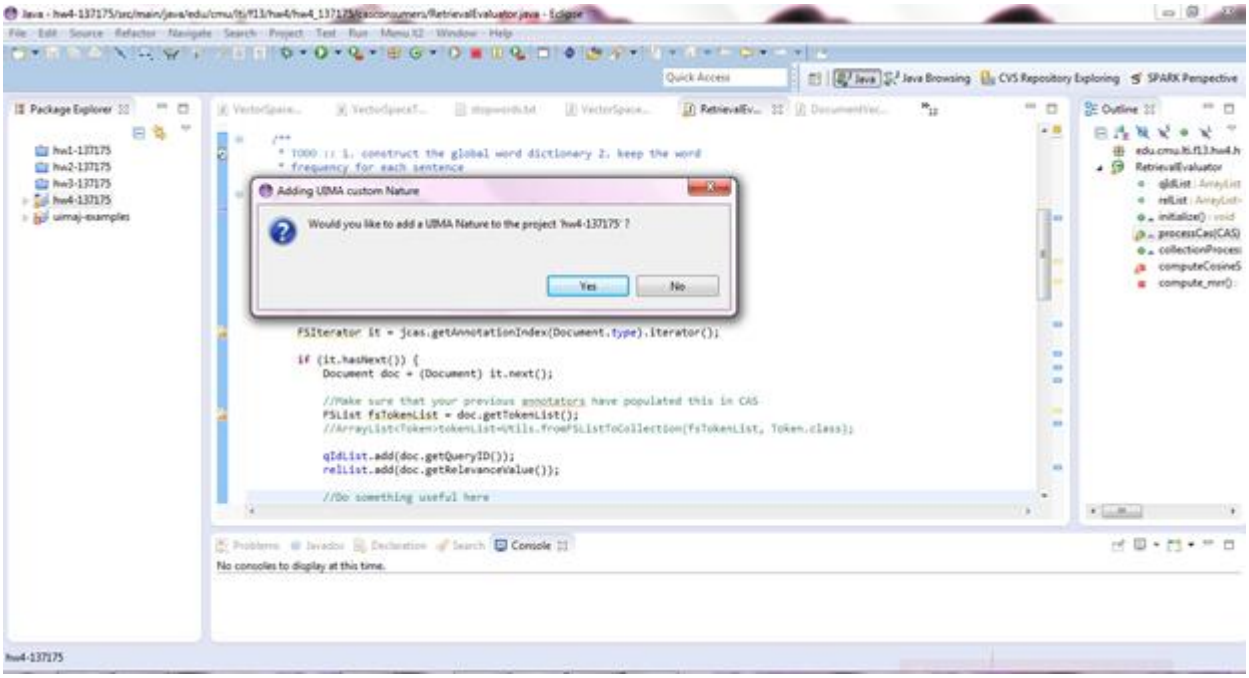
99 denotes the query itself example: qid=2 rel=99 Energy plays an important role in climate change
Means that query with id 2 has a text string : *Energy plays an important role in climate change.*

Task 1.2 Type System

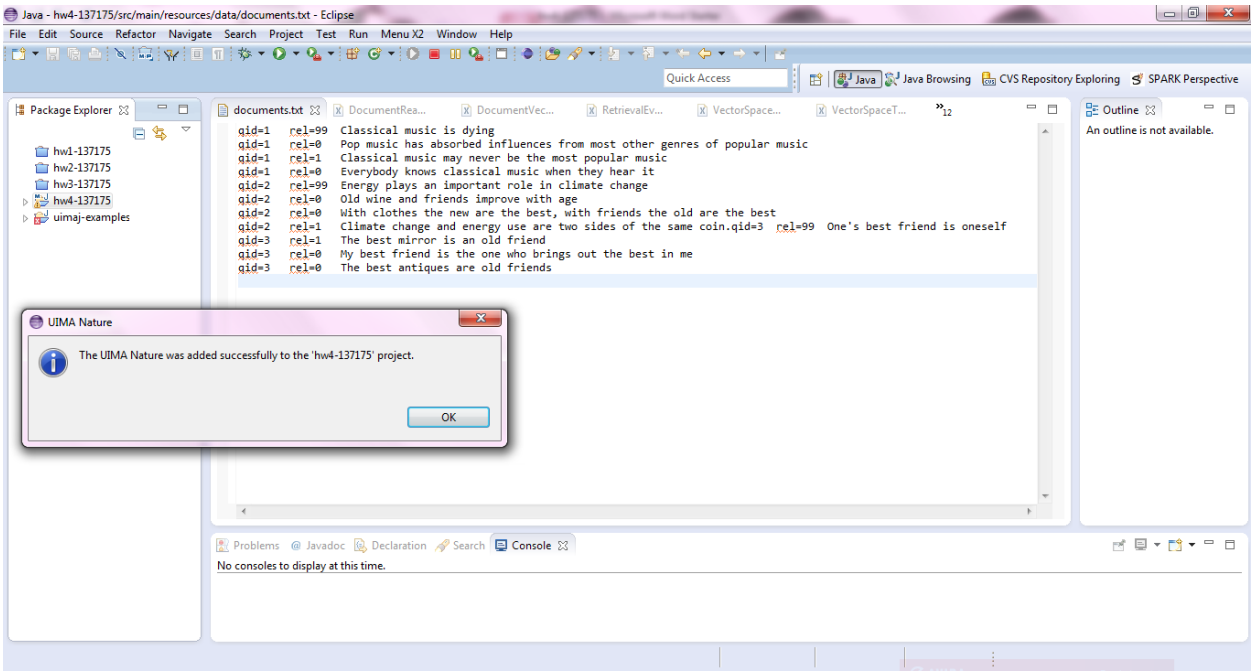
0. automatically generate the type system implementation using UIMA>>JCasGen

   Following the instructions that came on this link: https://uima.apache.org/doc-uima-annotator.html, the next Type System was created.

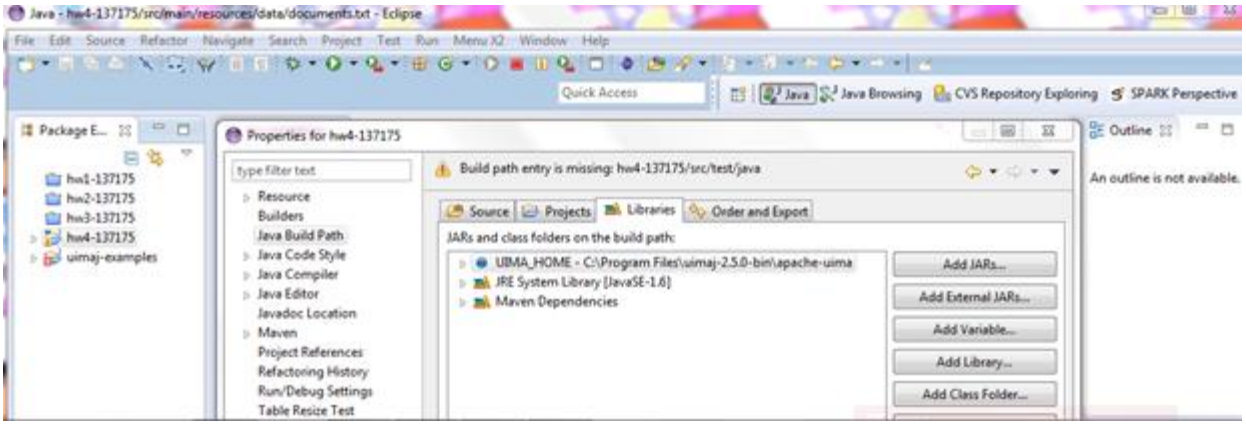
   a) Creating a new Java project in Eclipse workspace called hw4-137175, then add the UIMA nature to the project by right-clicking on the "hw4-137175" project and choose "Add UIMA Nature".



b) Verifying that UIMA Nature was added to hw4-137175.



c) Adding to this project the UIMA core libraries that are need to develop and run the annotator.

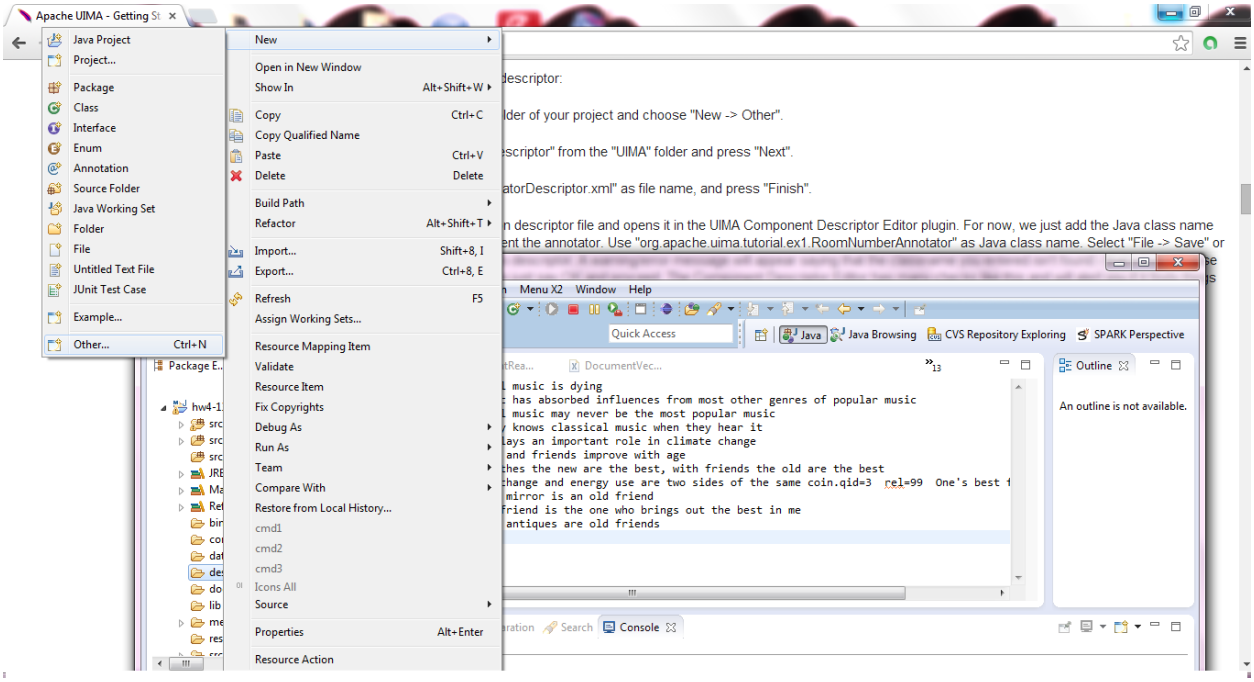


d) Start implementing the annotator then create some meta data for the annotator - the analysis engine descriptor.

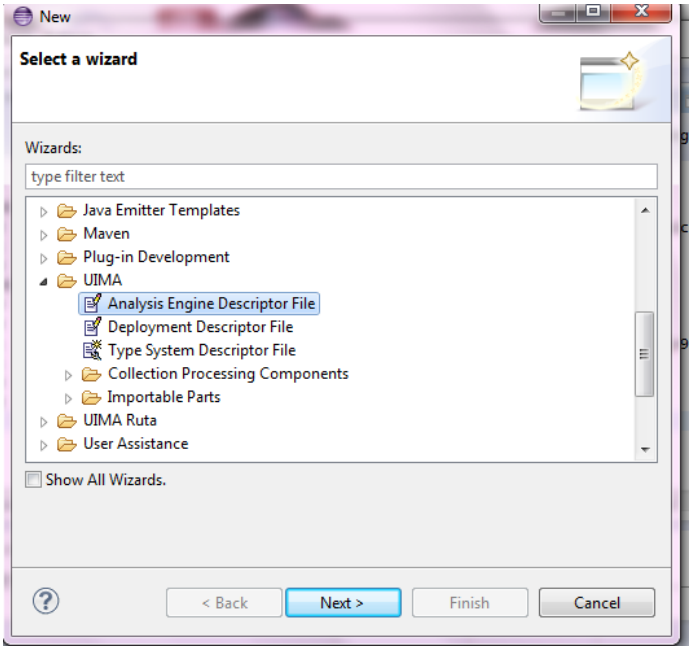
Analysis Engine Descriptor contains information about the annotator that is accessible without having access to source code. Contains information like data structures, annotator input and output, data types and resources that annotator uses. In the XML files are the description about each file.

Steps to create a new analysis engine descriptor:

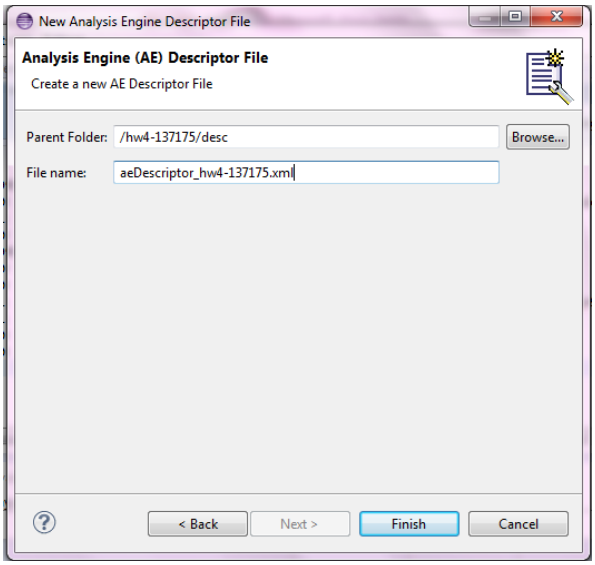
1. Go to desc folder then open New ->Other



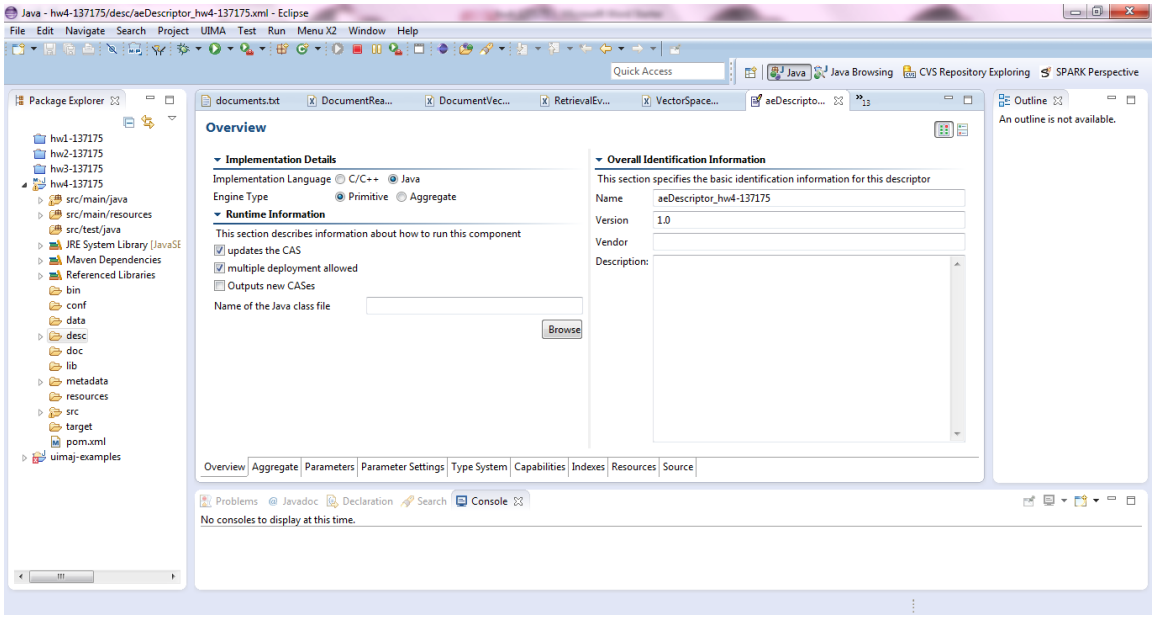
2. Then select the option “Analysis Engine Descriptor” from “UIMA” folder and click on “Next”



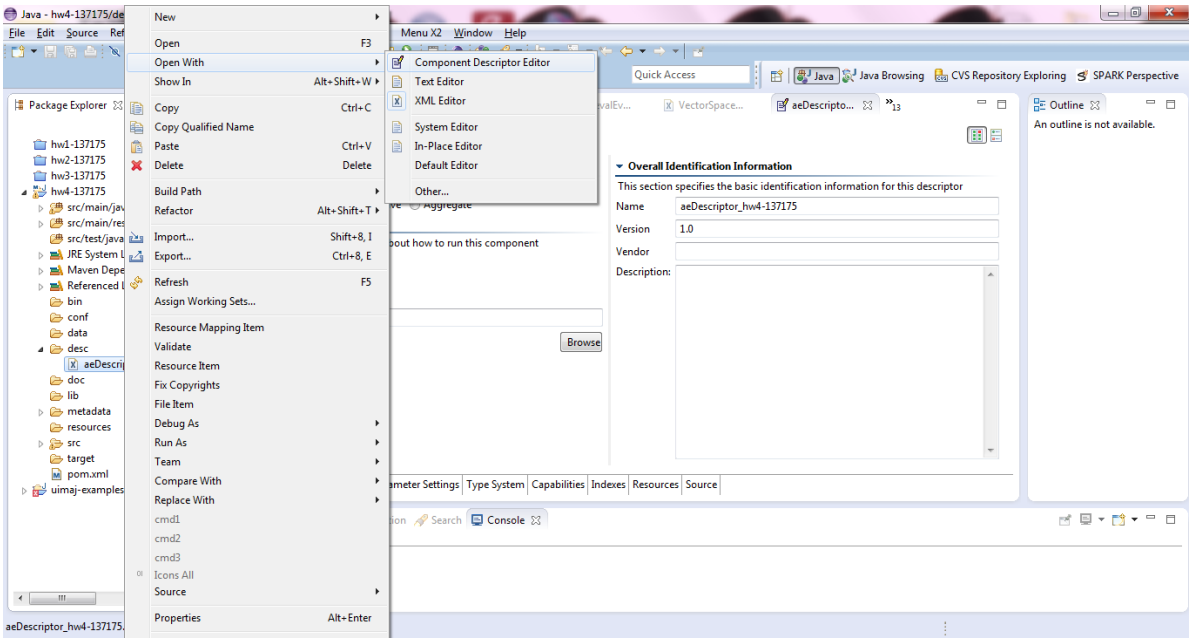
3. The new analysis engine descriptor file is aeDescriptor_hw4-137175.xml



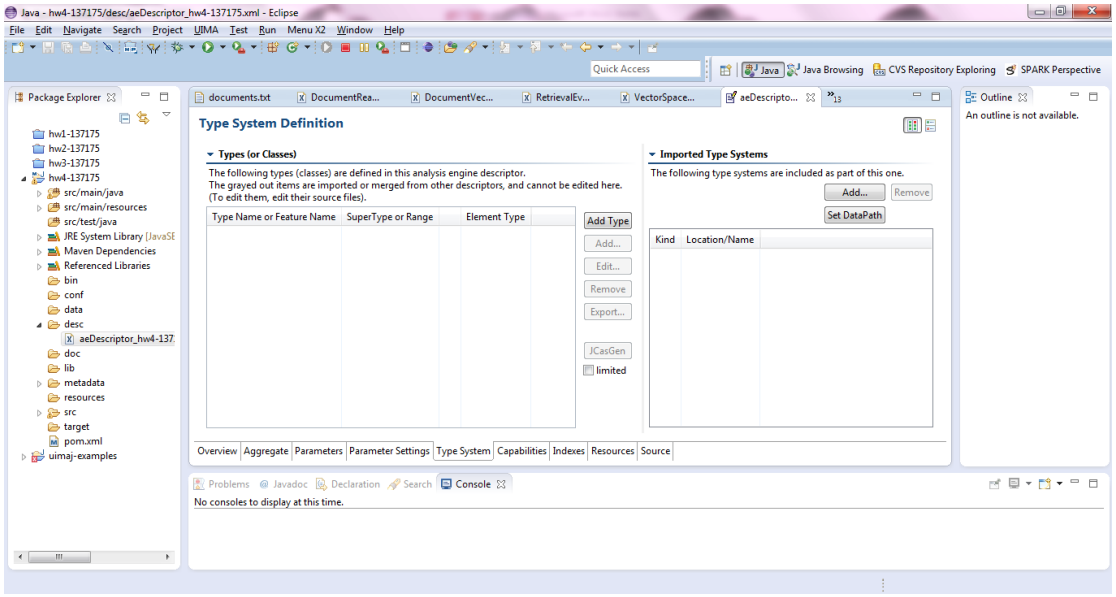
4. Those steps created a new skeleton descriptor file and open it in the UIMA Component Descriptor Editor plugin.



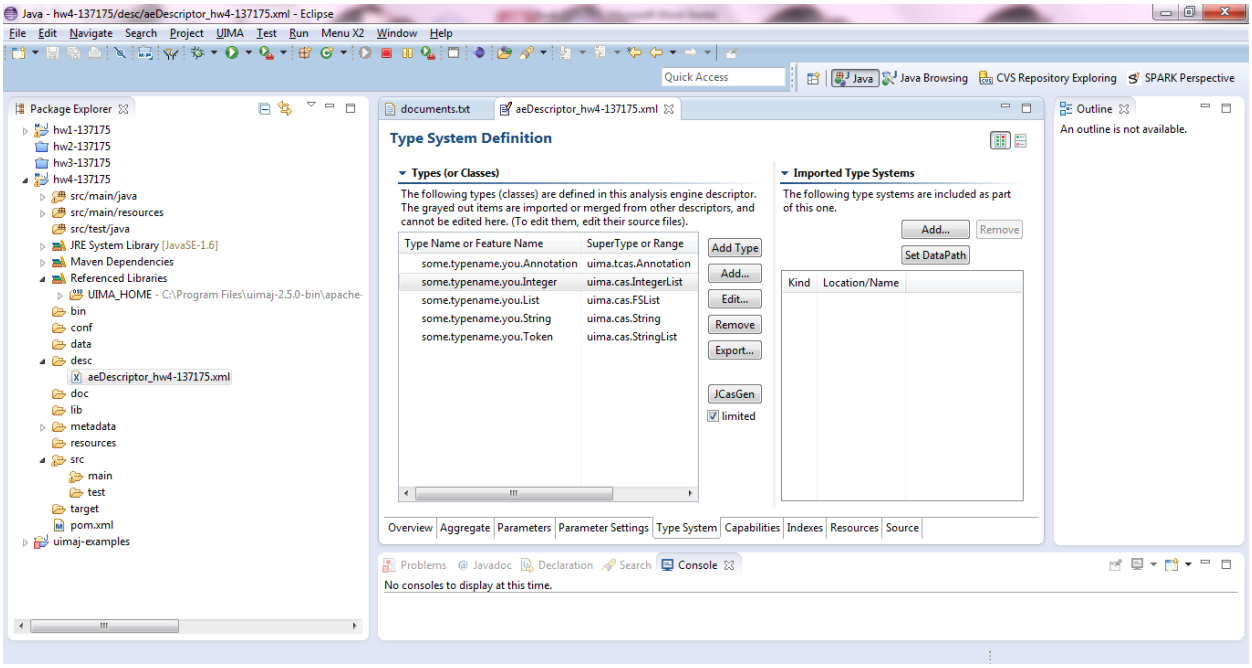
5. Go to UIMA Component Descriptor

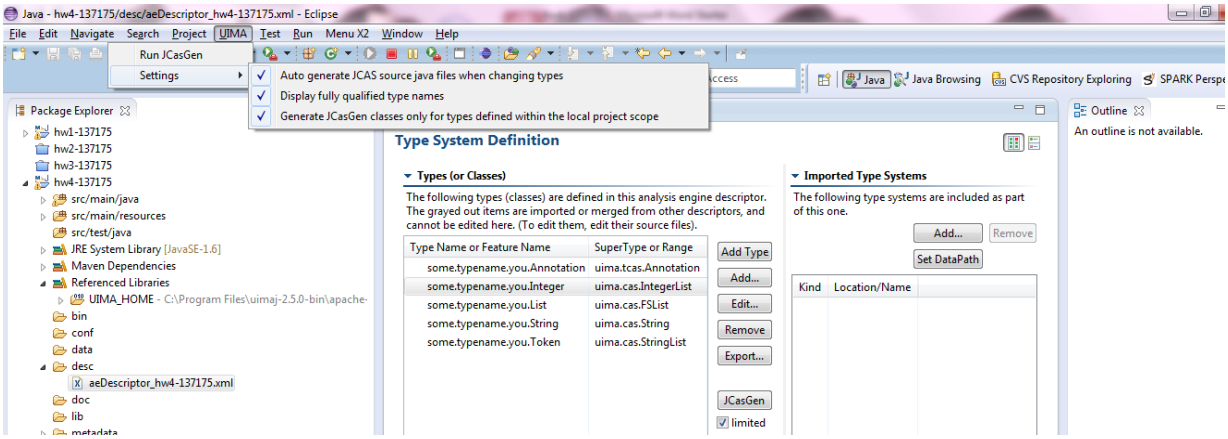


Adding more types a new Type SystemaeDescriptor-hw4_137175.xml the idea about make richer the other one or create a new one depending on task requirements, this is a test probably later should be changed.

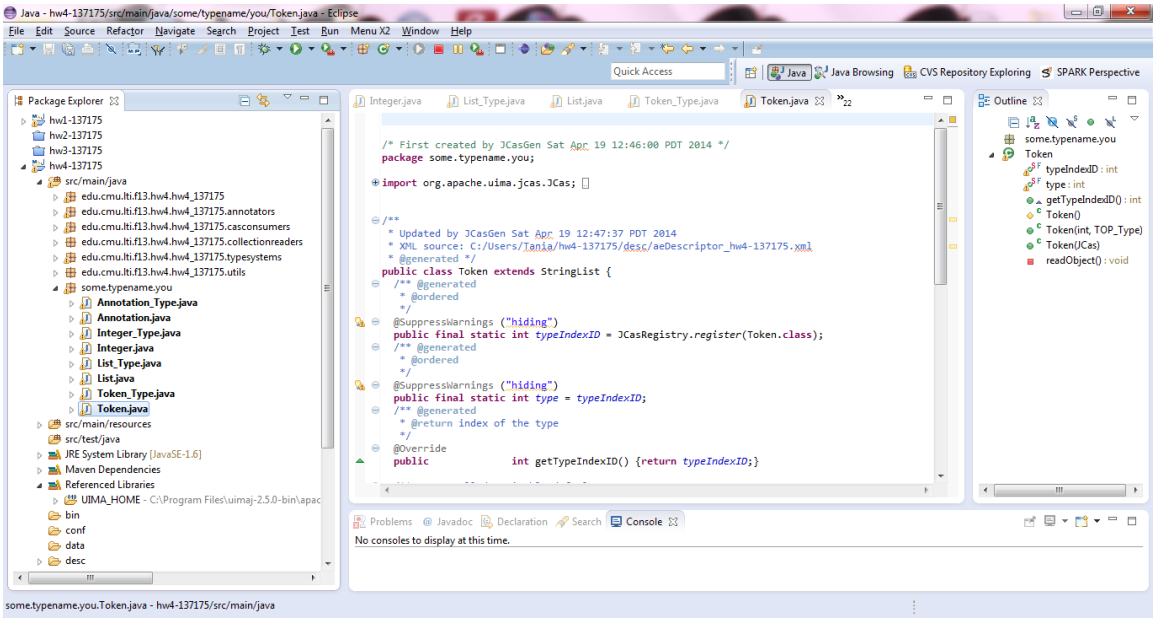


7. Type System Definition with more types





8. Verifying the code that was created using UIMA-JCas



6. Select the Type System tap and add new Types to the description to the Type System Definition. The initial type system contains Document and Token type.

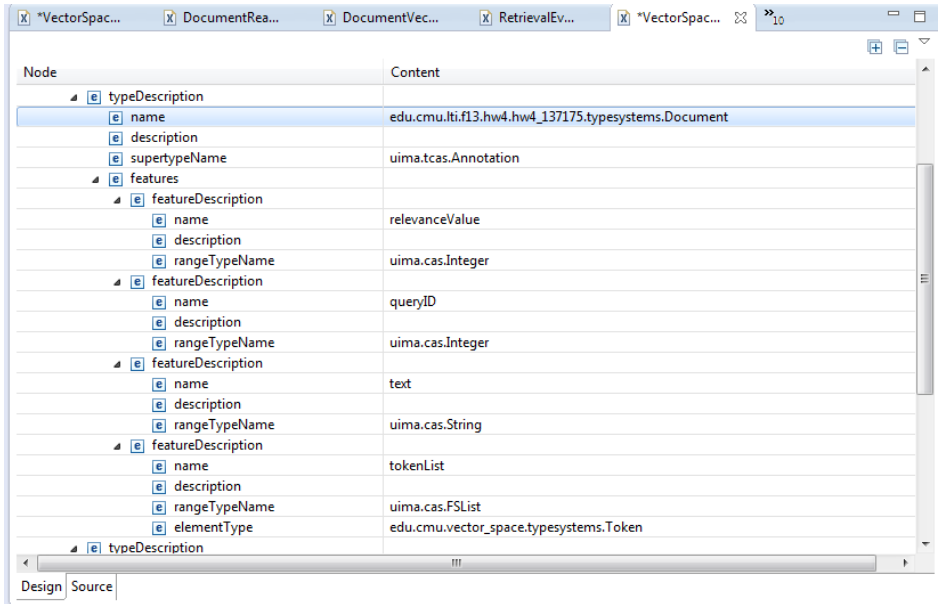
Document contains actually:

Relevance Value, Query ID, Text String, Token List.

Relevance Value, query ID, Text String are extracted from the text collection.

Token List represent 'bag of words' feature vectors and have to be constructed for the retrieval system.

Original Type System



References:

[1] http://uima.apache.org/d/uimaj-2.3.1/tools.html

[2] https://uima.apache.org/doc-uima-annotator.html

Task 1.3 Analysis Engine

1. Correctly extract bag of words feature vector from the input text collection

Bag of words, is a model that analyses the occurrence of words that maybe are on paragraphs, long pieces of text, record talksthose examples are part of the area (Natural Language Processing). On the other hand,if a classifier wants to analyze images or sounds, then applies bag of word model to organize important characteristics (features), where each part is like a puzzle piece, this corresponds to (Computer Vision Topics). In both cases, the main idea is to find specific features and make some data training and then make some data test to know which the behavior of the data is, when this model is applied.

Description about the Bag of words applied on the hw4-137175:

These are the java files that are part of this project: VectorSpaceRetrieval.java, DocumentVectorAnnotator.java, RetrievalEvaluator.java,DocumentReader.java,Document_Type.java,Document.java,Token_Type.java,Token.java, Utils.java.

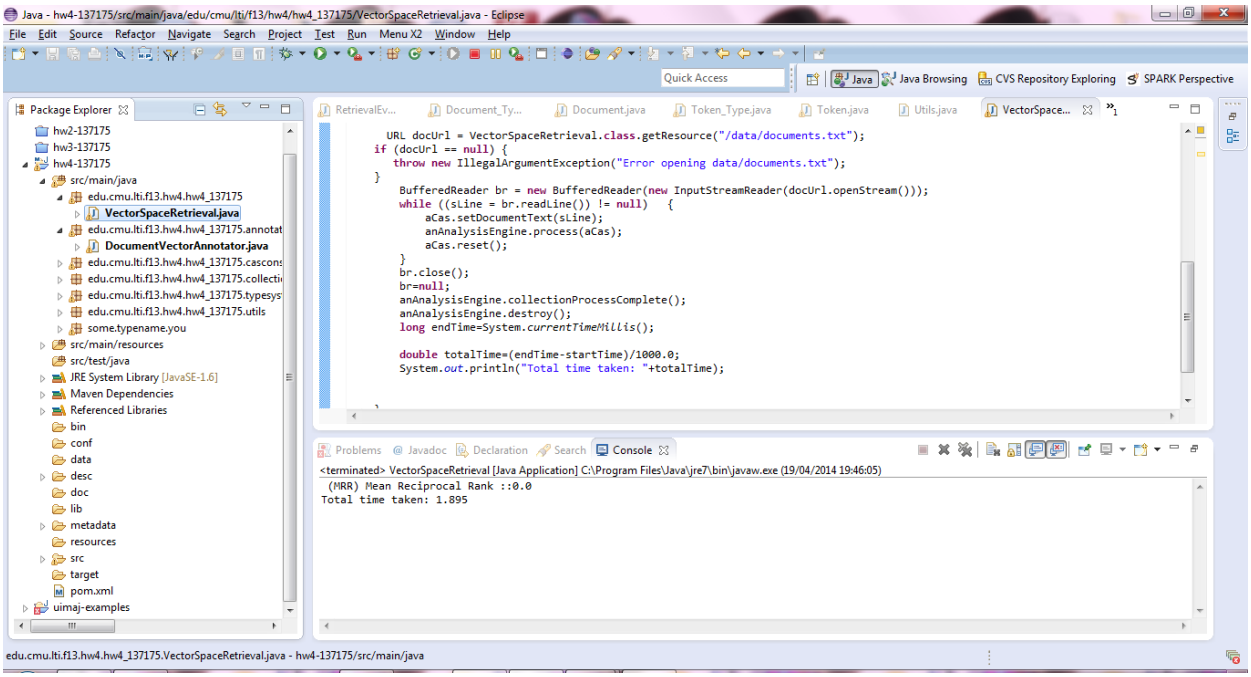
Others classes are Document_Type.java, Document.java, Token_Type.java, Token.java that extends from Annotation and Annotation_Type class. Each of those ones analyses in the document: frequence, features, relevanceValue, about words.

The next ones are the Java classes that were modified:

- **RetrievalEvaluator.java:**methods that were constructed: global word dictionary, cosine similarity and ***MMR (Mean Reciprocal Rank) metric*** but this one was not finished so I do not added in this version.
- **DocumentVectorAnnotator.java:**a vector of tokens was constructed and updatethe tokenList in CAS.
- **DocumentReader.java**: read and parser all document, gives a value to each keyword that finds with a possible feature.
- **VectorSpaceRetrieval.java:** contains the main of the program which implements the aggregate analysis engine VectorSpaceRetrieval.xml on a text corpus. This file contains the definition type system where is located a “Document”, “Token” type.
- **Bag of Words.**java: this contains the description about a bag of words , the bag was stored into:

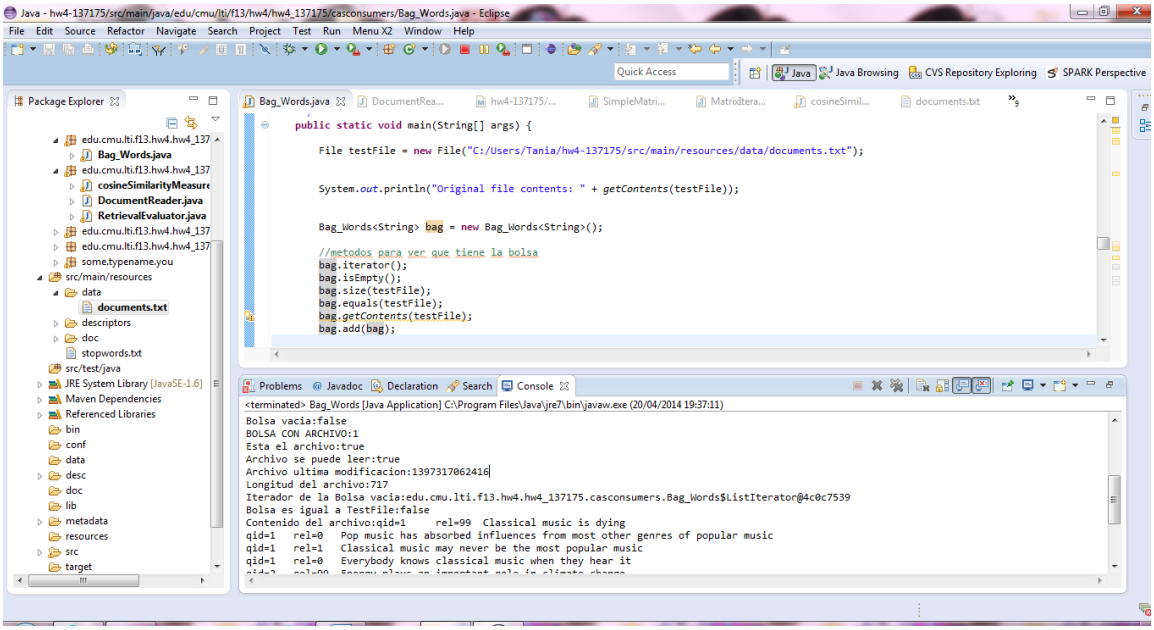
```
//bag
        ArrayList<String> bag = new ArrayList<String>();
```

Testing **VectorSpaceRetrieval.java:**



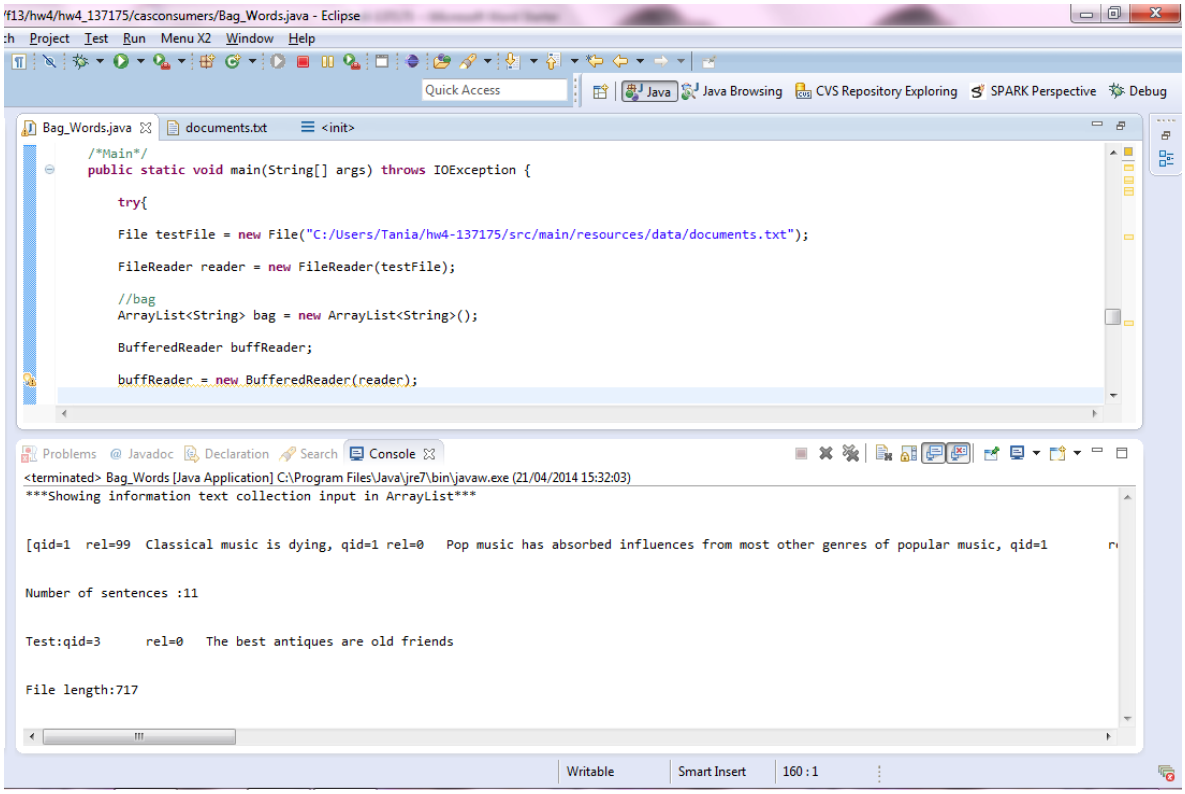
Bag of words Implementation from the input text collection: **document.txt**

Experiments with Bag of Words and document.txt files, to know what and which data is located on the Bag and in the textFile.



In the file **Bag_Words.java** are the code lines, related to how to load a file.txt, in a structure like Array List where many queries could be made finding the number of sentences, the file length, how many words are on the file, make some test searching in an specific position of the Array and know which sentence is located, like in `bag.get(10)` returns the sentence:

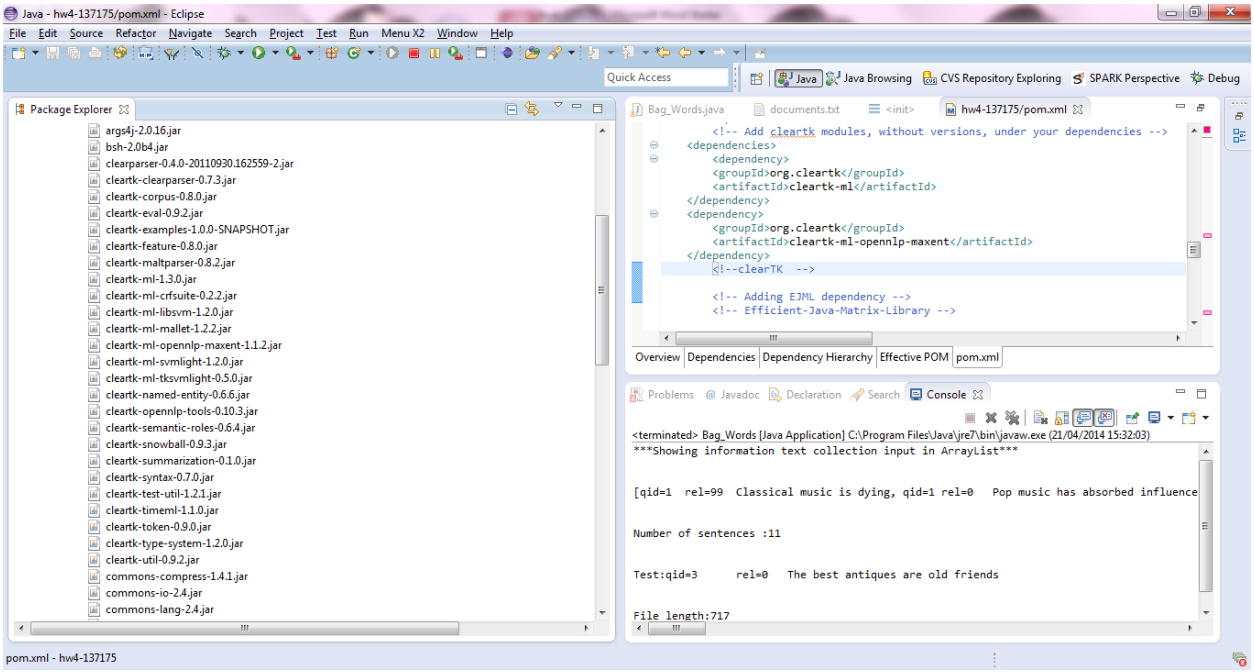
Test:qid=3 rel=0 The best antiques are old friends



Extract bag of words features vector from the input text collection

A toolCleartk , was found to make the extraction bag of words features from document.txt file.

Cleartk-release-1.4.1-bin.zip, this folder contains the next jar files:



Add this line in pom.xml file

```
<!--clearTK  -->
<!-- Declare cleartk-release as your parent -->
        <parent>
                <groupId>org.cleartk</groupId>
                <artifactId>cleartk-release</artifactId>
                <version>1.4.1</version>
                </parent>
<!-- Add cleartk modules, without versions, under your dependencies -->
        <dependencies>
                <dependency>
        <groupId>org.cleartk</groupId>
        <artifactId>cleartk-ml</artifactId>
        </dependency>
        <dependency>
        <groupId>org.cleartk</groupId>
        <artifactId>cleartk-ml-opennlp-maxent</artifactId>
        </dependency>
<!--clearTK  -->
```

References

[1] https://code.google.com/p/cleartk/downloads/detail?name=cleartk-release-1.4.1-bin.zip&can=2&q=

[2] http://mvnrepository.com/artifact/org.cleartk

2. Compute the cosine similarity between two sentences in the text collection.
   This is a fragment about how to compute the cosine similarity measure, some functions were called that are part of the import Jama.Matrix library like norm1, normF.

```
import  Jama.Matrix;

// TODO :: compute the cosine similarity measure

publicdoublecosineSimilarityMeasure(Matrix inputDocu, Matrix outputDocu){

                double dotProduct =  inputDocu.arrayTimes(outputDocu).norm1();
                double euclidianDist =  inputDocu.normF()* outputDocu.normF();
                returndotProduct / euclidianDist;
                }
```

The next functions were called in the computation of the cosine similarity between two sentences in the text collection: **document.txt**

```
/** One norm
   @return    maximum column sum.
   */

publicdouble norm1 () {
double f = 0;
for (int j = 0; j < n; j++) {
double s = 0;
for (int i = 0; i < m; i++) {
            s += Math.abs(A[i][j]);
         }
         f = Math.max(f,s);
      }
return f;
   }
```

```
/** Infinity norm
   @return    maximum row sum.
   */

publicdouble normInf () {
double f = 0;
for (int i = 0; i < m; i++) {
double s = 0;
for (int j = 0; j < n; j++) {
           s += Math.abs(A[i][j]);
         }
         f = Math.max(f,s);
      }
return f;
   }

/** Frobenius norm
   @return    sqrt of sum of squares of all elements.
   */

publicdouble normF () {
double f = 0;
for (int i = 0; i < m; i++) {
for (int j = 0; j < n; j++) {
           f = Maths.hypot(f,A[i][j]);
         }
      }
return f;
   }
```

Cosine Similarity

```
    public final double cosineSimilarityMeasure (Matrix inputDocu, Matrix outputDocu){

        double dotProduct =  inputDocu.arrayTimes(outputDocu).norm1();
        double euclidianDist =  inputDocu.normF()* outputDocu.normF();
        return dotProduct / euclidianDist;
    }
    /*
     * @return cosine_similarity
     */
     private List<RetrievalEvaluator> computeCosineSimilarity(
             //final Map<String, Integer> queryVector)
             final Map<String, Integer> docVector) {
             //double cosine_similarity=0.0;
             List<String> docNames = new ArrayList<String>();
             List <RetrievalEvaluator> result = new ArrayList <RetrievalEvaluator>();

         // TODO :: compute cosine similarity between two sentences
            Collections.sort(docNames, new Comparator<String>(){
         //comparing two strings
          public int compare(String s1, String s2) {
               return similarityMap.get(s2).compareTo(similarityMap.get(s1));
          }
          });
            for(String docName : docNames){
              double num = similarityMap.get(docName);
              if(num < 0.0001D){
                continue;
              }
              result.addAll((Collection<? extends RetrievalEvaluator>) new RetrievalEvaluatorResults(docName,num));
              }
            return result;
            }
```

Making some test in Maven build in the file pom.xml,the next dependencies were added: EJML library (efficient-java-matrix-library) and Jama (A Java Matrix Package) which is better? Which one is more apt for this task?. Finally I choose Jama, for making the experiment about cosine similarity.BUT I added a dependency related to EJML-Efficient Java Matrix Library.

```
<!-- Adding EJML dependency -->
<!-- Efficient-Java-Matrix-Library -->

      <dependency>
<groupId>com.googlecode.efficient-java-matrix-library</groupId>
<artifactId>ejml</artifactId>
<version>0.24</version>
</dependency>

<!-- Adding JAMA dependecy -->
<!-- A Java Matrix Package -->

<dependency>
<groupId>gov.nist.math</groupId>
<artifactId>jama</artifactId>
<version>1.0.2</version>
</dependency>
```

References

[1] http://en.wikipedia.org/wiki/Cosine_similarity

[2] http://ant.apache.org/ivy/ivyde/download.cgi

[3] http://ntwrkanlystlkit.sourceforge.net/java2html/Jama/Matrix.java.html

[4] http://math.nist.gov/javanumerics/jama/

[5] https://code.google.com/p/efficient-java-matrix-library/wiki/EjmlManual

- **DocumentVectorAnnotator.java:**a vector of tokens was constructed and updatethe tokenList in CAS.

```
     * @param doc
     */

    private static void createTermFreqVector(JCas jcas, Document doc) {

        String docText = doc.getText();

        //TO DO: construct a vector of tokens and update the tokenList in CAS

          Vector<StreamTokenizer> tokens = new Vector<StreamTokenizer>();
          StreamTokenizer token = new StreamTokenizer(new StringReader(docText));

          if(docText.length()!= 0){
              tokens.addElement(token);
             }else{
                 System.out.println("There are no more tokens!");

            }
          }
    }
```

DocumentReader.java

Trying to extract features about words with the method initialize

//kinds of features that are extracted from initialize method

```
publicvoid initialize(UimaContext context) throws ResourceInitializationException{

        super.initialize(context);

        Annotation token;
        this.tokenFeatureExtractor = new FeatureFunctionExtractor(
                        new NumericTypeFeatureFunction(),
                        new LowerCaseFeatureFunction(),
                        new Feature("is first letter 'c'?", token.getCoveredText().charAt(0) == 'c'),
                        new Feature("is first letter 'm'?", token.getCoveredText().charAt(0) == 'm'),
                        new Feature("is first letter 't'?", token.getCoveredText().charAt(0) == 't'),
                        new Feature("is first letter 'o'?", token.getCoveredText().charAt(0) == 'o'));

        this.contextFeatureExtractor = new CleartkExtractor(
         Token.class,
        new Feature(),
        newContext (2));
}
}
```