Παράρτημα. Υλοποίηση εφαρμογής σε Java

Thesis.java

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

public class Thesis {

    public static void main(String[] args) throws ParseException {

        String rdf_uri ="D:/eclipseWorkspace/Thesis/src/bangkok_dangerous_triples.nt";
        String ont_uri ="D:/eclipseWorkspace/Thesis/src/scriptontology.owl" ;

        CreateVectors cv = new CreateVectors();
        List<List<Integer>> workVectors = new ArrayList<List<Integer>>();
        List<Integer> workIndexes = new ArrayList<Integer>();
        List<String> workLabels = new ArrayList<String>();
        List<Integer> clusters = new ArrayList<Integer>();

        try{
            MyResult workResult = cv.vectorize(rdf_uri, ont_uri);
            workVectors = workResult.getFirst();
            workIndexes = workResult.getSecond();
            workLabels = workResult.getThird();
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
        }
        System.out.println("Printing to text files");

        String filename = "D:/eclipseWorkspace/Thesis/src/vectors.txt";
        Writer writer = null;
        try{
            writer = new FileWriter(filename);
            for (List<Integer> l : workVectors){
                for (int i : l){
                    writer.write(String.valueOf(i) + " ");
                }
                writer.write(System.getProperty("line.separator"));
            }
        }catch (IOException e){
            e.getMessage();
        }finally{
            try {writer.close();} catch (Exception ex) {}
        }

        String filename1 = "D:/eclipseWorkspace/Thesis/src/indexes.txt";
        Writer writer1 = null;
        try{
            writer1 = new FileWriter(filename1);
            for (int i : workIndexes){
                writer1.write(String.valueOf(i));
                writer1.write(System.getProperty("line.separator"));
```

```
            }
        }catch (IOException e1){
            e1.getMessage();
        }finally{
            try {writer1.close();} catch (Exception ex1) {}
        }

        String filename2 = "D:/eclipseWorkspace/Thesis/src/labels.txt";
        Writer writer2 = null;
        try{
            writer2 = new FileWriter(filename2);
            for (String s : workLabels){
                    writer2.write(s);
                writer2.write(System.getProperty("line.separator"));
            }
        }catch (IOException e2){
            e2.getMessage();
        }finally{
            try {writer2.close();} catch (Exception ex2) {}
        }
//-----------------------------------------------------------------
        System.out.println("Calculating cosine similarities");

        List<double[]> cosSims = new ArrayList<double[]>();

        for (int i=0; i<workVectors.size(); i++){
            double[] cos = new double[workVectors.size()];
            for (int j=0; j<workVectors.size(); j++){
                cos[j] = new CosineSimilarity().cosineSimilarity
                                (workVectors.get(i),workVectors.get(j));
            }
            cosSims.add(cos);
        }

        String filename2a = "D:/eclipseWorkspace/Thesis/src/DBsimilarities.csv";
        try{
            writer2 = new FileWriter(filename2a);
            int i = 0;
            for (double[] d : cosSims){
                int[] minIndex = ArrayStuff.min5(d);
                writer2.write("\"" + workIndexes.get(i) + "\"");
                for (int id : minIndex){
                    writer2.write(",\"" + workIndexes.get(id) + "\"");
                    writer2.write(",\"" + d[id] + "\"");
                }
                writer2.write(System.getProperty("line.separator"));
                i++;
            }
        }catch (IOException e2){
            e2.getMessage();
        }finally{
            try {writer2.close();} catch (Exception ex2) {}
        }

//-----------------------------------------------------------
        System.out.println("Initiating clustering");
        System.out.println("Clustering Complete, getting results");

        String resultsFile = "D:/eclipseWorkspace/Thesis/src/kmeansClusters.txt";
        try{
            Scanner scanner = new Scanner(new File(resultsFile));
            while(scanner.hasNextInt()){
                clusters.add(scanner.nextInt());
```

```
            }
            scanner.close();
        }
        catch(Exception e3){
            e3.getMessage();
        }

//-------------------------------------------------------------
        System.out.println("Creating the clusters");

        List<Cluster> workClusters = new ArrayList<Cluster>();
        int[] labelSums = new int[workLabels.size()];

        for(int i=1; i<=Collections.max(clusters);i++){
            Arrays.fill(labelSums, 0);
            Cluster c = new Cluster(i);

            for(int j=0; j<clusters.size();j++){
                if(clusters.get(j)==i){
                    c.inShots.add(workIndexes.get(j));
                    c.inVectors.add(workVectors.get(j));
                }
            }
            for (List<Integer> l:c.inVectors){
                for (int k=0;k<l.size();k++){
                    labelSums[k] += l.get(k);
                }
            }
            int[] maxIndex = ArrayStuff.top10(labelSums);
            for (int ind:maxIndex){
                c.topLabels.add(workLabels.get(ind));
            }
            workClusters.add(c);
        }

//-------------------------------------------------------------------------------------------
-
        System.out.println("Creating DB files");
        String filename3 = "D:/eclipseWorkspace/Thesis/src/KMEANSLabels.csv";
        Writer writer3 = null;
        try{
            writer3 = new FileWriter(filename3);
            for (Cluster c : workClusters){
                writer3.write("\"" + c.id + "\"");
                for (String s : c.topLabels){
                    writer3.write(",\"" + s + "\"");
                }
                writer3.write(System.getProperty("line.separator"));
            }
        }catch (IOException e3){
            e3.getMessage();
        }finally{
            try {writer3.close();} catch (Exception ex3) {}
        }

        String filename4 = "D:/eclipseWorkspace/Thesis/src/KMEANSShots.csv";
        try{
            writer3 = new FileWriter(filename4);
            for (Cluster c : workClusters){
                for (int i : c.inShots){
                    writer3.write("\"" + c.id + "\",\"" + i + "\"");
                    writer3.write(System.getProperty("line.separator"));
                }
```

```
            }
        }catch (IOException e3){
            e3.getMessage();
        }finally{
            try {writer3.close();} catch (Exception ex3) {}
        }

        String filename5 = "D:/eclipseWorkspace/Thesis/src/DBPics.csv";
        try{
            writer3 = new FileWriter(filename5);
            for (Cluster c : workClusters){
                for (int i : c.inShots){
                    writer3.write("\"" + i + "\",\"" + "picture"+
                            String.format("%04d",i) +".jpeg" + "\"");
                    writer3.write(System.getProperty("line.separator"));
                }
            }
        }catch (IOException e3){
            e3.getMessage();
        }finally{
            try {writer3.close();} catch (Exception ex3) {}
        }

        System.out.println("DONE");
    }

}
```

CreateVectors.java

```
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.Writer;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Set;
import java.util.TreeSet;

import com.google.common.collect.Lists;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.QuerySolutionMap;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;
import com.hp.hpl.jena.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;
```

```
import com.hp.hpl.jena.util.FileManager;

public class CreateVectors {

    public MyResult vectorize(String uri1, String uri2) throws IOException {

        FileManager.get().addLocatorClassLoader(CreateVectors.class.getClassLoader());
        Model myScript = FileManager.get().loadModel(uri1);

////////////////////////////////////////////////////////
        OntModel myOntology = ModelFactory.createOntologyModel
                                (OntModelSpec.OWL_MEM_MINI_RULE_INF);
        InputStream in = new FileInputStream(uri2);
        myOntology.read(in,null);
        in.close();

        OntClass shotClass = myOntology.getOntClass
                                ("http://moviedb.org/scriptontology/Shot");
        List<String> shotClassList = new ArrayList<String>();
        List<String> shotParent = new ArrayList<String>();
        Queue<OntClass> q = new LinkedList<OntClass>();
        q.add(shotClass);
        while (!q.isEmpty()) {
            OntClass oc = q.remove();
            shotClassList.add(oc.getURI().substring(oc.getURI().lastIndexOf("/")+1));

            OntClass p = oc.getSuperClass();
            if (p==null){
                shotParent.add("root");
            }
            else{
                shotParent.add(p.getURI().substring(p.getURI().lastIndexOf("/")+1));
            }

            Iterator<OntClass> it = oc.listSubClasses();
            List<OntClass> listit = Lists.newArrayList(it);
            for (OntClass noc : listit){
                q.add(noc);
            }
        }

        List<String> sceneClassList = new ArrayList<String>();
        Iterator<OntClass> it = myOntology.getOntClass
                    ("http://moviedb.org/scriptontology/Scene").listSubClasses();
        while (it.hasNext()) {
            OntClass oc = it.next();
            sceneClassList.add(oc.getURI().substring(oc.getURI().lastIndexOf("/")+1));
        }

////////////////////////////////////////////////

        List<List<Integer>> finalVectors = new ArrayList<List<Integer>>();
        List<Integer> indexes = new ArrayList<Integer>();
        List<String> labels = new ArrayList<String>();
        List<RDFNode> shots = new ArrayList<RDFNode>();
        List<String> roles = new ArrayList<String>(Collections.nCopies(48, ""));
        QuerySolutionMap initialBindings = new QuerySolutionMap();
//-------------------------------------------
        String queryStringDictionary =
                "PREFIX movie: <http://image.ntua.gr/scriptontology/> " +
                "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "SELECT ?z " +
                "WHERE {" +
```

```
                "{  ?x movie:description ?z .}" +
                "UNION" +
                "{  ?x movie:text ?z .}" +
                "UNION" +
                "{  ?x movie:name ?z .}" +
                "}";

        Query queryDictionary = QueryFactory.create(queryStringDictionary);
        QueryExecution qeDictionary = QueryExecutionFactory.create
                                        (queryDictionary,myScript);
        ResultSet resultsDictionary = qeDictionary.execSelect();
        //ResultSetFormatter.out(System.out, resultsDictionary, queryDictionary);

        StringBuilder allText = new StringBuilder();

        for ( ; resultsDictionary.hasNext() ;){
            QuerySolution soln = resultsDictionary.nextSolution();
            RDFNode n = soln.get("z");
            String value = ((Literal)n).toString().replaceAll("@en", "")
                                            .toLowerCase();
            allText.append(value + " ");
        }

        StanfordLemmatizer slemm = new StanfordLemmatizer();
        List<String> allLemmas = slemm.lemmatize(allText.toString());
        Set<String> distinctLemmas = new TreeSet<String>(allLemmas);
        List<String> dictionary = new ArrayList<String>(distinctLemmas);
        System.out.println(dictionary.size());
        qeDictionary.close();

//-----------------------------------------------
        String queryStringRole =
                "PREFIX movie: <http://image.ntua.gr/scriptontology/> " +
                "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "SELECT ?y ?z " +
                "WHERE {" +
                "   ?x movie:hasRole ?y ." +
                "   ?y movie:name ?z ." +
                "}";

        Query queryRole = QueryFactory.create(queryStringRole);
        QueryExecution qeRole = QueryExecutionFactory.create(queryRole,myScript);
        ResultSet resultsRole = qeRole.execSelect();
        //ResultSetFormatter.out(System.out, resultsRole, queryRole);

        for ( ;resultsRole.hasNext(); ){
            QuerySolution soln = resultsRole.nextSolution();
            RDFNode n1 = soln.get("y"); RDFNode n2 = soln.get("z");
            Resource r1 = (Resource)n1; Literal r2 = (Literal)n2;
            int index = Integer.parseInt(r1.getURI().substring
                                (r1.getURI().lastIndexOf("L")+1));
            roles.set(index, r2.toString().replaceAll("@en",""));
        }
        qeRole.close();

//-----------------------------------------------
        String queryStringShot =
                "PREFIX movie: <http://image.ntua.gr/scriptontology/> " +
                "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "SELECT *" +
                "WHERE {" +
                "   ?x rdf:type ?z ." +
                "}";
```

```
        Query queryShot = QueryFactory.create(queryStringShot);
        QueryExecution qeShot = QueryExecutionFactory.create(queryShot,myScript);
        ResultSet resultsShot = qeShot.execSelect();
        //ResultSetFormatter.out(System.out, resultsShot, queryShot);

        for ( ;resultsShot.hasNext(); ){
            QuerySolution soln = resultsShot.nextSolution();
            RDFNode n1 = soln.get("x"); RDFNode n2 = soln.get("z");
            Resource r2 = (Resource)n2;
            String attr = r2.getURI().substring(r2.getURI().lastIndexOf("/")+1);
            if (shotClassList.contains(attr)){
                if (!shots.contains(n1)){
                    shots.add(n1);
                }
            }
        }
        qeShot.close();
        List<String> startTimes = new ArrayList<String>(Collections.nCopies
                                                    ((shots.size()+5), " "));

////////////////////////////////////////////////////////////////////////////////

        for (RDFNode shot : shots){

            List<Integer> shotVector = new ArrayList<Integer>();
            List<Integer> temp1 = new ArrayList<Integer>(Collections.nCopies
                                        ((shotClassList.size()), 0));
            temp1.set(0, 1); //always subclass of shot
            List<Integer> temp2 = new ArrayList<Integer>(Collections.nCopies
                                        (sceneClassList.size(), 0));
            List<Integer> temp3 = new ArrayList<Integer>(Collections.nCopies
                                        (roles.size(), 0));
            List<Integer> temp4 = new ArrayList<Integer>(Collections.nCopies
                                        (dictionary.size(), 0));
            List<String> texts = new ArrayList<String>();

            System.out.println("Creating Vector for shot " + shot.toString());
            int ind = Integer.parseInt(shot.toString().substring
                                (shot.toString().lastIndexOf("H")+1));

            initialBindings.add("shot", shot);

//***1st Query : THE SHOT itself
//-----------------------------------------------
            String queryStringA =
                    "PREFIX movie: <http://image.ntua.gr/scriptontology/> " +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                    "SELECT * " +
                    "WHERE {" +
                    "   ?shot ?y ?z ." +
                    " }";
            Query queryA = QueryFactory.create(queryStringA);
            QueryExecution qeA = QueryExecutionFactory.create
                                        (queryA,myScript,initialBindings);
            ResultSet resultsA = qeA.execSelect();
            //ResultSetFormatter.out(System.out, resultsA, queryA);

            for ( ;resultsA.hasNext(); ){
                QuerySolution soln = resultsA.nextSolution();
                RDFNode n = soln.get("y");
                if ( n.isLiteral() )
                    System.out.println("error1");
```

```
                if ( n.isResource() ){
                    Resource r = (Resource)n;
                    if ( !r.isAnon() )
                    {
                        String attr = r.getURI();
                        if (attr.equals("http://www.w3.org/1999/02/22-rdf-syntax-
                                        ns#type"))
                        {
                            Resource m = soln.getResource("z");
                            String value = m.getURI().substring(m.getURI()
                                            .lastIndexOf("/")+1).replaceAll
                                            ("SideViewShot", "SideView");
                           //The replace is because the triples has error
                            if (value.equals("FlashbackScene")
                                  ||value.equals("FadeInScene")
                                  ||value.equals("DissolveScene")){
                                int index = sceneClassList.indexOf(value);
                                temp2.set(index, 1);
                            }
                            else {
                                int index = shotClassList.indexOf(value);
                                temp1.set(index, 1);
                                while (!(shotParent.get(index).equals("root"))){
                                    index = shotClassList.indexOf
                                            (shotParent.get(index));
                                    temp1.set(index, 1);
                                }
                            }
                        }
                        else if (attr.equals("http://image.ntua.gr/scriptontology/
                                            startTime"))
                        {
                            Literal l = soln.getLiteral("z");
                            String value = l.getValue().toString().replace("-T", "");
                            String fvalue = value.substring(0,8);
                            startTimes.set(ind, fvalue);
                        }
                        else if (attr.equals("http://image.ntua.gr/scriptontology/
                                            description"))
                        {
                            Literal l = soln.getLiteral("z");
                            String value = l.getValue().toString();
                            texts.add(value);
                        }
                        else if (attr.equals("http://image.ntua.gr/scriptontology/
                                            isNextOf"))
                        {
                        }
                    }
                }
            }

//***2nd Query : THE SCENE of the shot
//-----------------------------------------
            String queryStringB =
                    "PREFIX movie: <http://image.ntua.gr/scriptontology/> " +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                    "SELECT ?x ?y ?z " +
                    "WHERE {" +
                    "{  ?x movie:hasPart ?shot ." +
                    "   ?x rdf:type ?z ." +
                    "}" +
                    "UNION {" +
```

```
                "   ?x movie:hasPart ?shot ." +
                "   ?x movie:description ?z .}" +
                " }";
        Query queryB = QueryFactory.create(queryStringB);
        QueryExecution qeB = QueryExecutionFactory.create
                             (queryB,myScript,initialBindings);
        ResultSet resultsB = qeB.execSelect();
        //ResultSetFormatter.out(System.out, resultsB, queryB);

        for ( ;resultsB.hasNext(); )
        {
            QuerySolution soln = resultsB.nextSolution();
            RDFNode n = soln.get("z");
            if ( n.isLiteral() ){
            //its the description
                Literal l = (Literal)n;
                String value = l.getValue().toString();
                texts.add(value);
            }
            else if ( n.isResource() )
            //its the scene types
            {
                Resource r = (Resource)n;
                if ( !r.isAnon() )
                {
                    String attr = r.getURI();
                    String value = attr.substring(attr.lastIndexOf("/")+1);
                    int index = sceneClassList.indexOf(value);
                    temp2.set(index, 1);
                }
            }
        }

//***3rd Query : THE ACTS of the shot
//-----------------------------------------------------------------
        String queryStringC =
                "PREFIX movie: <http://image.ntua.gr/scriptontology/> " +
                "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "SELECT ?x ?y ?z " +
                "WHERE {" +
                "   ?x movie:happensIn ?shot ." +
                "   ?x ?y ?z ." +
                " }";
        Query queryC = QueryFactory.create(queryStringC);
        QueryExecution qeC = QueryExecutionFactory.create
                             (queryC,myScript,initialBindings);
        ResultSet resultsC = qeC.execSelect();
        //ResultSetFormatter.out(System.out, resultsC, queryC);

        for ( ;resultsC.hasNext(); )
        {
            QuerySolution soln = resultsC.nextSolution();
            RDFNode n = soln.get("y");
            if ( n.isLiteral() )
                System.out.println("error1");
            if ( n.isResource() ){
                Resource r = (Resource)n;
                if ( !r.isAnon() )
                {
                    String attr = r.getURI();
                    if (attr.equals("http://www.w3.org/1999/02/22-rdf-syntax-ns#
                                    type"))
                    {
```

```
                        //MOOT, All are Speaking Acts
                    }
                    else if ((attr.equals("http://image.ntua.gr/scriptontology/
                                        performedBy"))
                            ||(attr.equals("http://image.ntua.gr/scriptontology/
                                        addressedTo")))
                    {
                        Resource m = soln.getResource("z");
                        int index = Integer.parseInt(m.getURI().substring
                                        (m.getURI().lastIndexOf("L")+1));
                        temp3.set(index, 1);
                        String value = roles.get(index);
                        texts.add(value);
                    }
                    else if (attr.equals("http://image.ntua.gr/scriptontology/
                                        text"))
                    {
                        Literal l = soln.getLiteral("z");
                        String value = l.getValue().toString();
                        texts.add(value);
                    }
                    else if (attr.equals("http://image.ntua.gr/scriptontology/
                                        happensIn"))
                    {
                    }
                }
            }
        }

        StringBuilder shotText = new StringBuilder();
        for (String s : texts){
            shotText.append(s.toLowerCase() + " ");
        }

        StanfordLemmatizer slem = new StanfordLemmatizer();
        List<String> shotLemmas = slem.lemmatize(shotText.toString());
        for (String s : shotLemmas){
            int index = Collections.binarySearch(dictionary, s);
            if ((index>=0) && (index<=dictionary.size())){
                temp4.set(index, 1);
            }
            else {
                System.out.println("ERROR: word " + s + " with index "
                                + index + " not found");
            }
        }

        qeA.close();
        qeB.close();
        qeC.close();

        shotVector.addAll(temp4);shotVector.addAll(temp3);
       shotVector.addAll(temp2);shotVector.addAll(temp1);
        finalVectors.add(shotVector);
        indexes.add(ind);
        System.out.println("Vector for shot " + shot.toString() + " added");

    }

    labels.addAll(dictionary); labels.addAll(roles);
   labels.addAll(sceneClassList); labels.addAll(shotClassList);

//-----------------------------------------------------------
```

```
        String filename1 = "D:/eclipseWorkspace/Thesis/src/DBtimes.csv";
        Writer writer1 = null;
        try{
            writer1 = new FileWriter(filename1);
            int i = 0;
            for (String s : startTimes){
                int tmp = ArrayStuff.toSeconds(s);
                writer1.write(",\"" + tmp + "\"");
                writer1.write(System.getProperty("line.separator"));
                writer1.write("\"" + i + "\""); writer1.write(",\"" + tmp + "\"");
                i++;
            }
            writer1.write("\"" + "5435" + "\"");

        }catch (IOException e1){
            e1.getMessage();
        }finally{
            try {writer1.close();} catch (Exception ex1) {}
        }
//-------------------------------------------------------------
        return new MyResult(finalVectors, indexes,labels);
}
}
```

StanfordLemmatizer.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import edu.stanford.nlp.ling.CoreAnnotations.LemmaAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.SentencesAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.TokensAnnotation;
import edu.stanford.nlp.ling.CoreLabel;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.util.CoreMap;


public class StanfordLemmatizer {

    protected StanfordCoreNLP pipeline;

    public StanfordLemmatizer(){

        Properties props = new Properties();
        props.put("annotators","tokenize, ssplit, pos, lemma");

        this.pipeline = new StanfordCoreNLP(props);
    }

    public List<String> lemmatize(String documentText)
    {
        List<String> lemmas = new ArrayList<String>();

        Annotation document = new Annotation(documentText);
        this.pipeline.annotate(document);

        List<CoreMap> sentences = document.get(SentencesAnnotation.class);
        for (CoreMap sentence : sentences){
            for (CoreLabel token : sentence.get(TokensAnnotation.class)){
```

```
                lemmas.add(token.get(LemmaAnnotation.class));
            }
        }

        return lemmas;
    }

}
```

CosineSimilarity.java

```
import java.util.List;


public class CosineSimilarity {

    public double cosineSimilarity(List<Integer> vec1, List<Integer> vec2 ){
        double dotProduct = 0.0;
        double magnitude1 = 0.0;
        double magnitude2 = 0.0;
        double cosineSimilarity = 0.0;

        for (int i=0; i<vec1.size(); i++){
            dotProduct += vec1.get(i) * vec2.get(i);
            magnitude1 += Math.pow(vec1.get(i), 2);
            magnitude2 += Math.pow(vec2.get(i), 2);
        }

        magnitude1 = Math.sqrt(magnitude1);
        magnitude2 = Math.sqrt(magnitude2);

        if ((magnitude1 != 0.0) && (magnitude2 != 0.0)){
            cosineSimilarity = dotProduct/(magnitude1*magnitude2);
        }

        return cosineSimilarity;
    }

}
```

MyResult.java

```
import java.util.List;

final class MyResult {
    private final List<List<Integer>> first;
    private final List<Integer> second;
    private final List<String> third;

    public MyResult(List<List<Integer>> first,List<Integer> second, List<String>
third){
        this.first = first;
        this.second = second;
        this.third = third;
    }

    public List<List<Integer>> getFirst() {
        return first;
    }
```

```
    public List<Integer> getSecond() {
        return second;
    }

    public List<String> getThird() {
        return third;
    }

}
```

Cluster.java

```
import java.util.ArrayList;
import java.util.List;


final class Cluster {
    public int id;
    public List<Integer> inShots;
    public List<List<Integer>> inVectors;
    public List<String> topLabels;

    public Cluster(int id){
        this.id = id;
        this.inShots = new ArrayList<Integer>();
        this.inVectors = new ArrayList<List<Integer>>();
        this.topLabels = new ArrayList<String>();
    }

    public Cluster(int id, List<Integer> inShots, List<List<Integer>> inVectors,
List<String> topLabels){
        this.id = id;
        this.inShots = inShots;
        this.inVectors = inVectors;
        this.topLabels = topLabels;
    }

    public List<List<Integer>> getVectors() {
        return inVectors;
    }

    public List<Integer> getShots() {
        return inShots;
    }

    public List<String> getTopLabels() {
        return topLabels;
    }

    public void printCluster(){
        System.out.println("Cluster " + id + " has shots: " + inShots.size() + " and
                           vectors: " + inVectors.size());
        System.out.println("The most common features in the cluster are:");
        for (String s:topLabels){
            System.out.println(s);
        }
    }

}
```

ArrayStuff.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class ArrayStuff {

    public static int[] top10(int[] array){
        //Returns the indexes of the 10 largest elements of the input array

        int[] temp = new int[10];
        int[] maxIndex = new int[10];
        List<Integer> clone = new ArrayList<Integer>();

        for(int j:array){
            clone.add(j);
        }

        if (array.length<=10){
            return maxIndex;
        }
        else{
            Arrays.sort(array);
            for (int i=0;i<temp.length;i++){
                temp[i] = array[array.length-i-1];
            }
            int k=0;
            for (int i:temp){
                int key = clone.indexOf(i);
                maxIndex[k] = key;
                k++; clone.set(key,5000);
            }
        }
        return maxIndex;
    }

    public static int[] min5(double[] array){
        //Returns the indexes of the 5 smallest elements of the input array
        double[] temp = new double[5];
        int[] minIndex = new int[5];
        List<Double> clone = new ArrayList<Double>();

        for(double j:array){
            clone.add(j);
        }

        if (array.length<=5){
            return minIndex;
        }
        else{
            Arrays.sort(array);
            for (int i=0;i<temp.length;i++){
                temp[i] = array[i];
            }
            int k=0;
            for (double i:temp){
                int key = clone.indexOf(i);
                minIndex[k] = key;
                k++; clone.set(key,2.0);
            }
        }
```

```
        return minIndex;
    }

    public static int toSeconds(String time){
        //Convert timestamp of "hh:MM:ss" to seconds
        int sec = Integer.parseInt(time.substring(6,8));
        int min = Integer.parseInt(time.substring(3,5));
        int hour = Integer.parseInt(time.substring(0,2));

        int seconds = sec + 60*min + 3600*hour;

        return seconds;
    }

}
```