LAMPIRAN

PERANGKAT LUNAK

```
package com.fr.core;

import com.googlecode.javacv.cpp.opencv_core;
import com.googlecode.javacv.cpp.opencv_imgproc;
import com.googlecode.javacv.cpp.opencv_objdetect;
import java.awt.image.BufferedImage;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class FaceDetector
{
   private static final String CASCADE_FILE =
"C:\\opencv\\data\\haarcascades\\haarcascade_frontalface_alt_tree.xml";
//"haarcascade_frontalface_alt.xml";
   private static final int SCALE = 2;
   private static final opencv_objdetect.CvHaarClassifierCascade cascade = new
opencv_objdetect.CvHaarClassifierCascade(opencv_core.cvLoad(CASCADE_FI
LE)); // instantiate a classifier cascade for face detection

   public static BufferedImage detectFace(opencv_core.IplImage origImg)
   {
      BufferedImage faceBufferedImage = null;

      // convert to grayscale
      opencv_core.IplImage grayImg =
opencv_core.IplImage.create(origImg.width(), origImg.height(),
opencv_core.IPL_DEPTH_8U, 1);
      opencv_imgproc.cvCvtColor(origImg, grayImg,
opencv_imgproc.CV_BGR2GRAY);

      // scale the grayscale (to speed up face detection)
      opencv_core.IplImage smallImg =
opencv_core.IplImage.create(grayImg.width()/SCALE, grayImg.height()/SCALE,
opencv_core.IPL_DEPTH_8U, 1);
      opencv_imgproc.cvResize(grayImg, smallImg,
opencv_imgproc.CV_INTER_LINEAR);

      // equalize the small grayscale
      opencv_core.IplImage equImg =
opencv_core.IplImage.create(smallImg.width(), smallImg.height(),
opencv_core.IPL_DEPTH_8U, 1);
      opencv_imgproc.cvEqualizeHist(smallImg, equImg);

      // create temp storage, used during object detection
      opencv_core.CvMemStorage storage =
opencv_core.CvMemStorage.create();
      opencv_core.CvSeq faces = opencv_objdetect.cvHaarDetectObjects(equImg,
cascade, storage, 1.1, 3,
opencv_objdetect.CV_HAAR_DO_CANNY_PRUNING);
```

```
        opencv_core.cvClearMemStorage(storage);

        // iterate over the faces and draw yellow rectangles around them
        int total = faces.total();

        if(total>0)
        {
            opencv_core.CvRect faceRectangle = new
opencv_core.CvRect(opencv_core.cvGetSeqElem(faces, 0));

            int x1 = faceRectangle.x()*SCALE;
            int y1 = faceRectangle.y()*SCALE;

            int x2 = x1+(faceRectangle.width()*SCALE);
            int y2 = y1+(faceRectangle.height()*SCALE);

            int width = faceRectangle.width()*SCALE;
            int height = faceRectangle.height()*SCALE;

            faceBufferedImage = origImg.getBufferedImage().getSubimage(x1, y1,
width, height);
            opencv_core.cvRectangle(origImg, opencv_core.cvPoint(x1,y1),
opencv_core.cvPoint(x2,y2), opencv_core.CvScalar.YELLOW, 1,
opencv_core.CV_AA, 0); // undo the scaling
        }

        //Deallocate grayImg, smallImg, equImg because it's not used anymore.
        grayImg.release();
        smallImg.release();
        equImg.release();
        System.gc();

        return faceBufferedImage;
    }
}

package com.fr.core;

import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author Kurnia Novita Mutu
 */
@XmlRootElement
public class MapOfFeatureVector
{
    private LinkedHashMap<String,double[][]> mapOfFeatureVector = new
```

```
LinkedHashMap<>();

   public LinkedHashMap<String, double[][]> getMapOfFeatureVector()
   {
      return mapOfFeatureVector;
   }

   public void setMapOfFeatureVector(LinkedHashMap<String, double[][]>
mapOfFeatureVector)
   {
      this.mapOfFeatureVector = mapOfFeatureVector;
   }

   public double[][] put(String key, double[][] value)
   {
      return mapOfFeatureVector.put(key, value);
   }

   public Set<Map.Entry<String,double[][]>> entrySet()
   {
      return mapOfFeatureVector.entrySet();
   }
}

package com.fr.core;

import com.fr.properties.MyProperties;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class Matrix2D
{
   public static double[][] add(double[][] a, double[][] b, double[][] result)
   {
      double value;

      int rows = a.length;
      int cols = a[0].length;

      for(int i=0; i<rows; i++)
      {
         for(int j=0; j<cols; j++)
         {
            value = a[i][j]+b[i][j];
            result[i][j] = value;
         }
      }
      return result;
   }
```

```
   public static double[][] sub(double[][] a, double[][] b, double[][] result)
   {
      double value;

      int rows = a.length;
      int cols = a[0].length;

      for(int i=0; i<rows; i++)
      {
         for(int j=0; j<cols; j++)
         {
            value = a[i][j]-b[i][j];
            result[i][j] = value;
         }
      }
      return result;
   }

   public static double[][] multiply(double[][] a, double[][] b)
   {
      int aRows = a.length;
      int aCols = a[0].length;

      int bRows = b.length;
      int bCols = b[0].length;

      if(aCols!=bRows)
      {
         throw new IllegalArgumentException("Matrices don't match: " + aCols + "
!= " + bRows);
      }

      double[][] result = new double[aRows][bCols];

      for(int i=0; i<aRows; i++)
      {
         for(int j=0; j<bCols; j++)
         {
            for(int k=0; k<aCols; k++)
            {
               result[i][j] += a[i][k]*b[k][j];
            }
         }
      }

      return result;
   }

   public static double[][] scale(double[][] a, double scalar, double[][] result)
   {
```

```
    int rows = a.length;
    int cols = a[0].length;

    for(int i=0; i<rows; i++)
    {
        for(int j=0; j<cols; j++)
        {
            result[i][j] = a[i][j]*scalar;
        }
    }

    return result;
}

public static double[][] transpose(double[][] a)
{
    int rows = a.length;
    int cols = a[0].length;

    double[][] result = new double[cols][rows];

    for(int i=0; i<rows; i++)
    {
        for(int j=0; j<cols; j++)
        {
            result[j][i] = a[i][j];
        }
    }

    return result;
}

public static double square(double[][] a, double[][] b)
{
    double result = 0;

    int rows = a.length;
    int cols = a[0].length;

    for(int i=0; i<rows; i++)
    {
        for(int j=0; j<cols; j++)
        {
            result += Math.pow(a[i][j]-b[i][j],2);
        }
    }

    result = Math.sqrt(result);

    return result;
}
```

```
    public static double[][] copy(double[][] a)
    {
        int rows = a.length;
        int cols = a[0].length;

        double[][] result = new double[rows][cols];

        for(int i=0; i<rows; i++)
        {
            System.arraycopy(a[i], 0, result[i], 0, cols);
        }

        return result;
    }

    public static String convertToString(double[][] a)
    {
        StringBuilder temp = new StringBuilder();

        int rows = a.length;
        int cols = a[0].length;

        for(int i=0; i<rows; i++)
        {
            for(int j=0; j<cols; j++)
            {

temp.append(a[i][j]).append(MyProperties.PCA_2D_LOG_SEPARATOR);
            }
        }

        return temp.substring(0, temp.length()-1);
    }

    public static void print(double[][] a)
    {
        int rows = a.length;
        int cols = a[0].length;

        for(int i=0; i<rows; i++)
        {
            for(int j=0; j<cols; j++)
            {
                System.out.print(a[i][j]+"\t");
            }
            System.out.println();
        }
    }
}
package com.fr.core;
```

```
import com.fr.file.FileUtils;
import com.fr.properties.MyProperties;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import org.ejml.data.Matrix64F;
import org.ejml.factory.EigenDecomposition;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class MyEigenSelector implements Comparable<MyEigenSelector>
{
   private int index;
   private double j;

   private static double[][] convertMatrix64F(Matrix64F matrix64F)
   {
      int eigenVectorRows = matrix64F.numRows;
      int eigenVectorCols = matrix64F.numCols;

      double[][] eigenVector = new double[eigenVectorRows][eigenVectorCols];

      for(int i=0; i<eigenVectorRows; i++)
      {
         for(int j=0; j<eigenVectorCols; j++)
         {
            eigenVector[i][j] = matrix64F.get(i, j);
         }
      }

      return eigenVector;
   }

   public static List<Integer> getListOfOrthonormalIndex(EigenDecomposition
ed)
   {
      char sep = MyProperties.PCA_2D_LOG_SEPARATOR;

      StringBuilder log = new StringBuilder();
      log.append("sep=").append(sep).append("\r\n");
      log.append("List of Eigen Value & Eigen Vector\r\n");
      log.append("Index").append(sep).append("Eigen
Value").append(sep).append("Eigen Vector\r\n");

      List<Integer> listOfOrthonormalIndex = new LinkedList<>();
      List<double[][]> temp = new ArrayList<>();
```

```
        for(int i=0; i<ed.getNumberOfEigenvalues(); i++)
        {
           double[][] eigenVector = convertMatrix64F(ed.getEigenVector(i));

log.append(i).append(MyProperties.PCA_2D_LOG_SEPARATOR).append(ed.ge
tEigenvalue(i).real).append(MyProperties.PCA_2D_LOG_SEPARATOR).append
(Matrix2D.convertToString(eigenVector)).append("\r\n");
           temp.add(eigenVector);
        }

        int no = 1;
        log.append("\r\n");
        log.append("List of Orthonormal Matrices's Index\r\n");
        log.append("No.").append(sep).append("Index Matrix
I").append(sep).append("Index Matrix II\r\n");

        for(int i=0; i<temp.size(); i++)
        {
           for(int j=i+1; j<temp.size(); j++)
           {
              double[][] a = temp.get(i);
              double[][] at = Matrix2D.transpose(a);
              double[][] b = temp.get(j);
              double[][] c = Matrix2D.multiply(at, b);

              if(c[0][0]==0)
              {

log.append(no).append(sep).append(i).append(sep).append(j).append("\r\n");
                 no++;

                 if(!listOfOrthonormalIndex.contains(i))
                 {
                    listOfOrthonormalIndex.add(i);
                 }

                 if(!listOfOrthonormalIndex.contains(j))
                 {
                    listOfOrthonormalIndex.add(j);
                 }
              }
           }
        }

        FileUtils.writeLog(log, false);

        return listOfOrthonormalIndex;
     }

     public static void selectEigenVector(EigenDecomposition ed, double[][]
covariance, LinkedList<Double> listOfEigenValue, LinkedList<double[][]>
```

```
listOfEigenVector)
   {
      char sep = MyProperties.PCA_2D_LOG_SEPARATOR;

      StringBuilder log = new StringBuilder();
      log.append("\r\n");
      log.append("List of selected Eigen Value & Eigen Vector\r\n");
      log.append("Index").append(sep).append("Eigen
Value").append(sep).append("J(Sx)").append(sep).append("Eigen Vector\r\n");

      StringBuilder orthonormalInfo = new StringBuilder();

      List<Integer> listOfSelectedIndex = getListOfOrthonormalIndex(ed);

      List<MyEigenSelector> listOfEigenSelectors = new LinkedList<>();

      for(int i=0; i<listOfSelectedIndex.size(); i++)
      {
         int selectedIndex = listOfSelectedIndex.get(i);
         double[][] t = convertMatrix64F(ed.getEigenVector(selectedIndex));
         double[][] xt = Matrix2D.transpose(t);
         double[][] xtCovariance = Matrix2D.multiply(xt, covariance);
         double[][] jx = Matrix2D.multiply(xtCovariance, t);

         MyEigenSelector myEigenSelector = new MyEigenSelector();
         myEigenSelector.setIndex(selectedIndex);
         myEigenSelector.setJ(jx[0][0]);
         listOfEigenSelectors.add(myEigenSelector);
      }

      Collections.sort(listOfEigenSelectors);

      int numberOfUsedEigenValue =
Preference.getInstance().getNumberOfSelectedEigenVector();

      for(int i=0; i<numberOfUsedEigenValue; i++)
      {
         if(i<listOfEigenSelectors.size())
         {
            int selectedIndex = listOfEigenSelectors.get(i).getIndex();
            double eigenValue = ed.getEigenvalue(selectedIndex).real;
            double[][] eigenVector =
convertMatrix64F(ed.getEigenVector(selectedIndex));

            listOfEigenValue.add(eigenValue);
            listOfEigenVector.add(eigenVector);


log.append(selectedIndex).append(MyProperties.PCA_2D_LOG_SEPARATOR).
append(eigenValue).append(MyProperties.PCA_2D_LOG_SEPARATOR).appen
d(listOfEigenSelectors.get(i).j).append(MyProperties.PCA_2D_LOG_SEPARAT
```

```
OR).append(Matrix2D.convertToString(eigenVector)).append("\r\n");
            }
        }

        FileUtils.writeLog(log, true);
    }

    public int getIndex()
    {
        return index;
    }

    public void setIndex(int index)
    {
        this.index = index;
    }

    public double getJ()
    {
        return j;
    }

    public void setJ(double j)
    {
        this.j = j;
    }

    @Override
    public int compareTo(MyEigenSelector obj)
    {
        return Double.compare(obj.getJ(),j);
    }
}package com.fr.core;

import com.fr.image.util.ImageReader;
import com.fr.properties.MyProperties;
import java.io.File;
import java.io.FileFilter;
import java.io.FilenameFilter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import org.ejml.data.DenseMatrix64F;
import org.ejml.factory.DecompositionFactory;
import org.ejml.factory.EigenDecomposition;

/**
 *
```

```
 * @author Kurnia Novita Mutu
 */
public class Pca2dAlgorithm
{
   public static double[][] getMean(ArrayList<double[][]> listOfArrayData)
   {
      int size = listOfArrayData.size();

      double[][] result = Matrix2D.copy(listOfArrayData.get(0));

      for(int i=1; i<size; i++)
      {
         Matrix2D.add(result, listOfArrayData.get(i), result);
      }

      Matrix2D.scale(result, 1D/size, result);

      return result;
   }

   public static Pca2dResult trainingData()
   {
      File rootOfFaceRecognitionDatabase = new
File(MyProperties.PARENT_FILE_PATH);

      File[] listOfDatabaseFolder = rootOfFaceRecognitionDatabase.listFiles(new
FileFilter()
      {
         @Override
         public boolean accept(File pathname)
         {
            return pathname.isDirectory();
         }
      });

      LinkedHashMap<String, double[][]> mapOfOriginalImageData = new
LinkedHashMap<>();

      for(File file : listOfDatabaseFolder)
      {
         String name = file.getName();

         File[] listOfImageFile = file.listFiles(new FilenameFilter()
         {
            @Override
            public boolean accept(File dir, String name)
            {
               return name.toLowerCase().endsWith(".bmp");
            }
         });
```

```
            for(File imageFile : listOfImageFile)
            {
               String key = name+"-
"+imageFile.getName().toLowerCase().replace(".bmp", "");
               double[][] imageData =
ImageReader.getData(imageFile.getAbsolutePath());
               mapOfOriginalImageData.put(key, imageData);
            }
         }
         Pca2dResult pca2dResult = calculatedPca2d(mapOfOriginalImageData);
         return pca2dResult;
      }

      public static Pca2dResult calculatedPca2d(LinkedHashMap<String,double[][]>
mapOfOriginalImageData)
      {
         //Calculate the mean of image matrix.
         ArrayList<double[][]> listOfData = new
ArrayList<>(mapOfOriginalImageData.values());
         double[][] mean = Pca2dAlgorithm.getMean(listOfData);

         //Normalize all image matrix.
         HashMap<String,double[][]> mapOfNormalizedImageData = new
HashMap<>();

         for(Map.Entry<String,double[][]> entry :
mapOfOriginalImageData.entrySet())
         {
            double[][] n = Matrix2D.copy(entry.getValue()); //n is normalized matrix
            Matrix2D.sub(n, mean, n);
            mapOfNormalizedImageData.put(entry.getKey(), n);
         }

         //Calculate the covariance matrix.
         double[][] c = null;

         for(Map.Entry<String,double[][]> entry :
mapOfNormalizedImageData.entrySet())
         {
            double[][] t = Matrix2D.transpose(entry.getValue());
            double[][] multiplyResult = Matrix2D.multiply(t, entry.getValue());

            if(c!=null)
            {
               Matrix2D.add(c, multiplyResult, c);
            }
            else
            {
               c = multiplyResult;
            }
         }
```

```
        Matrix2D.scale(c, 1D/mapOfNormalizedImageData.size(), c);

        //Calculate the EigenVector.
        DenseMatrix64F dm = new DenseMatrix64F(c);
        EigenDecomposition ed = DecompositionFactory.eig(100, true); //Matrix
size = 100 is not used, but must use to create an instance of EigenDecomposition.
I used 100 because the picture is 100x100.
        ed.decompose(dm);

        //Get eigen value & eigen vector
        LinkedList<Double> listOfEigenValue = new LinkedList<>();
        LinkedList<double[][]> listOfEigenVector = new LinkedList<>();
        MyEigenSelector.selectEigenVector(ed, c, listOfEigenValue,
listOfEigenVector);
        LinkedList<MapOfFeatureVector> listOfMapOfFeatureVector = new
LinkedList<>();

        for(Map.Entry<String,double[][]> entry :
mapOfOriginalImageData.entrySet())
        {
           MapOfFeatureVector mapOfOneImageFeatureVector = new
MapOfFeatureVector();

           for(int i=0; i<listOfEigenValue.size(); i++)
           {
              double[][] featureVector = Matrix2D.multiply(entry.getValue(),
listOfEigenVector.get(i));
              mapOfOneImageFeatureVector.put(entry.getKey()+"-"+(i+1),
featureVector);
           }

           listOfMapOfFeatureVector.add(mapOfOneImageFeatureVector);
        }

        Pca2dResult pca2dResult = new Pca2dResult();
        pca2dResult.setListOfEigenValue(listOfEigenValue);
        pca2dResult.setListOfEigenVector(listOfEigenVector);
        pca2dResult.setListOfMapOfFeatureVector(listOfMapOfFeatureVector);

        return pca2dResult;
    }

    public static String recognized(Pca2dResult pca2dResult, double[][] testImage,
Map<String,Double> mapOfResultInfo)
    {
        List<Double> listOfEigenValue = pca2dResult.getListOfEigenValue();
        List<double[][]> listOfEigenVector = pca2dResult.getListOfEigenVector();
        List<MapOfFeatureVector> listOfMapOfFeatureVector =
pca2dResult.getListOfMapOfFeatureVector();
```

```
        List<double[][]> listOfFeatureVectorOfTestImage = new LinkedList<>();

        for(int i=0; i<listOfEigenValue.size(); i++)
        {
           double[][] featureVectorOfTestImage = Matrix2D.multiply(testImage,
listOfEigenVector.get(i));
           listOfFeatureVectorOfTestImage.add(featureVectorOfTestImage);
        }

        double minRange = Double.POSITIVE_INFINITY;
        String result = "";

        for(MapOfFeatureVector mapOfFeatureVector : listOfMapOfFeatureVector)
        {
           int i = 0;
           double range = 0;
           String key = "";

           for(Map.Entry<String,double[][]> entry : mapOfFeatureVector.entrySet())
           {
              key = entry.getKey();
              double[][] value = entry.getValue();
              range += Matrix2D.square(listOfFeatureVectorOfTestImage.get(i),
value);
              i++;
           }

           //range = Math.sqrt(range);

           String[] temp = key.split("-");
           String name = "";

           for(int j=0; j<temp.length-1; j++)
           {
              name += temp[j];

              if(j!=temp.length-2)
              {
                 name += "-";
              }
           }

           mapOfResultInfo.put(name, range);

           if(range<minRange)
           {
              result = key;
              minRange = range;
           }
        }
```

```
        return result;
    }
}
package com.fr.core;

import java.io.Serializable;
import java.util.LinkedList;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author Kurnia Novita Mutu
 */
@XmlRootElement
public class Pca2dResult implements Serializable
{
    private LinkedList<Double> listOfEigenValue;
    private LinkedList<double[][]> listOfEigenVector;
    private LinkedList<MapOfFeatureVector> listOfMapOfFeatureVector;

    public Pca2dResult()
    {
    }

    public LinkedList<Double> getListOfEigenValue()
    {
        return listOfEigenValue;
    }

    public void setListOfEigenValue(LinkedList<Double> listOfEigenValue)
    {
        this.listOfEigenValue = listOfEigenValue;
    }

    public LinkedList<double[][]> getListOfEigenVector()
    {
        return listOfEigenVector;
    }

    public void setListOfEigenVector(LinkedList<double[][]> listOfEigenVector)
    {
        this.listOfEigenVector = listOfEigenVector;
    }

    public LinkedList<MapOfFeatureVector> getListOfMapOfFeatureVector()
    {
        return listOfMapOfFeatureVector;
    }

    public void setListOfMapOfFeatureVector(LinkedList<MapOfFeatureVector>
listOfMapOfFeatureVector)
```

```
    {
        this.listOfMapOfFeatureVector = listOfMapOfFeatureVector;
    }
}
package com.fr.core;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class Preference
{
    private static Preference instance;
    private int numberOfSelectedEigenVector = 10;

    private Preference()
    {
    }

    public int getNumberOfSelectedEigenVector()
    {
        return numberOfSelectedEigenVector;
    }

    public void setNumberOfSelectedEigenVector(int
numberOfSelectedEigenVector)
    {
        this.numberOfSelectedEigenVector = numberOfSelectedEigenVector;
    }

    public static Preference getInstance()
    {
        if(instance==null)
        {
            instance = new Preference();
        }

        return instance;
    }
}

package com.fr.file;

import com.fr.core.Pca2dResult;
import com.fr.properties.MyProperties;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
```

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import javax.imageio.ImageIO;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class FileUtils
{
   public static void writePca2dResult(Pca2dResult pca2dResult, String path)
   {
      try
      {
         JAXBContext context = JAXBContext.newInstance(Pca2dResult.class);

         Marshaller marshaller = context.createMarshaller();
         marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
true);
         marshaller.marshal(pca2dResult, new BufferedWriter(new
FileWriter(path)));
      }
      catch(JAXBException | IOException ex)
      {
         ex.printStackTrace(System.out);
      }
   }

   public static Pca2dResult readPca2dResult(String path)
   {
      Pca2dResult pca2dResult = null;

      try
      {
         JAXBContext context = JAXBContext.newInstance(Pca2dResult.class);

         Unmarshaller unmarshaller = context.createUnmarshaller();
         pca2dResult = (Pca2dResult) unmarshaller.unmarshal(new
BufferedReader(new FileReader(path)));
      }
      catch(JAXBException | IOException ex)
      {
         ex.printStackTrace(System.out);
      }
```

```
    return pca2dResult;
}

public static boolean addSampleImage(String name, BufferedImage image)
{
    boolean result = false;

    try
    {
        String parentPath = MyProperties.PARENT_FILE_PATH;

        File file = null;

        for(int i = 1; i<1000; i++)
        {
            file = new File(parentPath+"\\"+name+"\\"+i+".bmp");

            if(!file.exists())
            {
                break;
            }
        }

        file.getParentFile().mkdirs();
        result = ImageIO.write(image, "BMP", file);
    }
    catch(Exception ex)
    {
        ex.printStackTrace(System.out);
    }

    return result;
}

public static void writeLog(StringBuilder log, boolean append)
{
    try
    {
        File file = new File(MyProperties.PCA_2D_LOG);
        FileOutputStream fos = new FileOutputStream(file, append);
        PrintWriter pw = new PrintWriter(fos);
        pw.append(log);
        pw.close();
        fos.close();
    }
    catch(FileNotFoundException ex)
    {
    }
    catch(IOException ex)
    {
    }
```

```
    }
}

package com.fr.file;

import java.io.File;
import java.io.Serializable;
import javax.swing.filechooser.FileFilter;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class MyFileFilter extends FileFilter implements Serializable
{
    private String filter;
    private String name;

    public MyFileFilter(String filter, String name)
    {
        this.filter = filter.toLowerCase();
        this.name = name;
    }

    @Override
    public boolean accept(File f)
    {
        if(f.isDirectory())
        {
            return true;
        }

        String fileName = f.getName().toLowerCase();
        return fileName.endsWith(filter);
    }

    @Override
    public String getDescription()
    {
        return name;
    }
}
package com.fr.image.util;

import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

/**
```

```
    }

    public void refreshImageViewer(opencv_core.IplImage img)
    {
        BufferedImage faceBufferedImage = FaceDetector.detectFace(img);
        imageViewer1.setBufferedImage(img.getBufferedImage());

        if(faceBufferedImage!=null)
        {
            Image scaleFaceImage =
faceBufferedImage.getScaledInstance(MyProperties.FACE_IMAGE_WIDTH,
MyProperties.FACE_IMAGE_HEIGHT, BufferedImage.SCALE_SMOOTH);
            BufferedImage scaleFaceBufferedImage = new
BufferedImage(MyProperties.FACE_IMAGE_WIDTH,
MyProperties.FACE_IMAGE_HEIGHT, BufferedImage.TYPE_BYTE_GRAY);
            scaleFaceBufferedImage.getGraphics().drawImage(scaleFaceImage, 0, 0,
null);
            imageViewer2.setBufferedImage(scaleFaceBufferedImage);
        } // Proses Rescale dan Grayscale
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        /*
         * Create and display the form
         */
        java.awt.EventQueue.invokeLater(new Runnable()
        {
            @Override
            public void run()
            {
                new MainFrame().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton addBtn;
    private javax.swing.JButton browseSourceImageBtn;
    private com.fr.image.viewer.ImageViewer imageViewer1;
    private com.fr.image.viewer.ImageViewer imageViewer2;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JMenuItem jMenuItem4;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
```

```
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JSplitPane jSplitPane1;
    private javax.swing.JButton recognizeBtn;
    // End of variables declaration
}

package com.fr.main;

import java.awt.image.BufferedImage;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;
import javax.swing.DefaultRowSorter;
import javax.swing.ImageIcon;
import javax.swing.RowSorter;
import javax.swing.SortOrder;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class ResultDialog extends javax.swing.JDialog
{
    public ResultDialog(java.awt.Frame parent, boolean modal)
    {
        super(parent, modal);
        initComponents();
        tableResult.setAutoCreateRowSorter(true);

        DefaultRowSorter sorter = ((DefaultRowSorter)tableResult.getRowSorter());
        ArrayList list = new ArrayList();
        list.add(new RowSorter.SortKey(1, SortOrder.ASCENDING));
        sorter.setSortKeys(list);
        sorter.sort();

        setLocationRelativeTo(null);
    }

    public void setResult(String name, LinkedHashMap<String,Double>
mapOfResultInfo, BufferedImage bufferedImageResult)
    {
        if(bufferedImageResult!=null)
        {
            imageFrame.setIcon(new
ImageIcon(bufferedImageResult.getScaledInstance(imageFrame.getWidth(),
imageFrame.getHeight(), BufferedImage.SCALE_SMOOTH)));
        }
```

```
        lblResultName.setText(name);
        DefaultTableModel dtm = (DefaultTableModel)tableResult.getModel();

        for(Map.Entry<String,Double> entry : mapOfResultInfo.entrySet())
        {
            Object[] row = new Object[2];
            row[0] = entry.getKey();
            row[1] = BigDecimal.valueOf(entry.getValue()).setScale(2,
RoundingMode.CEILING);
            dtm.addRow(row);
        }
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {

        lblResultName = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        tableResult = new javax.swing.JTable();
        imageFrame = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOS
E);
        setTitle("Result Dialog");


lblResultName.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        lblResultName.setText("Result Name");

        tableResult.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][]
            {

            },
            new String []
            {
                "Name", "Nearest Matrix Features"
            }
        )
        {
            Class[] types = new Class []
```

```
            {
               java.lang.String.class, java.lang.Double.class
            };
            boolean[] canEdit = new boolean []
            {
               false, false
            };

            public Class getColumnClass(int columnIndex)
            {
               return types [columnIndex];
            }

            public boolean isCellEditable(int rowIndex, int columnIndex)
            {
               return canEdit [columnIndex];
            }
         });
         jScrollPane1.setViewportView(tableResult);

         imageFrame.setBorder(javax.swing.BorderFactory.createTitledBorder(""));

         javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
         getContentPane().setLayout(layout);
         layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
               .addGap(0, 152, Short.MAX_VALUE)
               .addComponent(imageFrame,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
               .addGap(148, 148, 148))
            .addGroup(layout.createSequentialGroup()
               .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING)
                  .addComponent(lblResultName,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                  .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 0, Short.MAX_VALUE))
               .addContainerGap())
         );
         layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
```

```
                .addContainerGap()
                .addComponent(imageFrame,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(lblResultName)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 202, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    // Variables declaration - do not modify
    private javax.swing.JLabel imageFrame;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lblResultName;
    private javax.swing.JTable tableResult;
    // End of variables declaration
}

package com.fr.properties;

/**
 *
 * @author Kurnia Novita Mutu
 */
public class MyProperties
{
    public static final String PARENT_FILE_PATH = "D:\\FR2";
    public static final String PCA_2D_RESULT_PATH =
PARENT_FILE_PATH+"\\Pca2dResult.xml";
    public static final String PCA_2D_LOG =
PARENT_FILE_PATH+"\\Pca2dLog.csv";
    public static final char PCA_2D_LOG_SEPARATOR = '\t';

    public static final int FACE_IMAGE_WIDTH = 100;
    public static final int FACE_IMAGE_HEIGHT = 100;
}
```