A  Homework Report on

Captcha

In the partial fulfilment of the academic requirements for

M. S (Cyber Security) under Wright State University

By

Sudarshan Govindarajan               (U00793458)

**Implementation of the System**

The following is the report that illustrates the implementation of the captcha obtained by expanding the existing system.

Most of the java class files are same as provided by the author.

**Execution Steps:**

**Start Eclipse**

**File->Import->General->ExistingProjectsintoWorkSpace->PasswordCracker-Dictionary**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Captcha;

/\*\*

\*

\* @author lingg

\*/

import java.awt.Color;

import java.awt.List;

import java.awt.image.BufferedImage;

import java.awt.image.FilteredImageSource;

import java.awt.image.ImageFilter;

import java.awt.image.ImageProducer;

import java.awt.image.RGBImageFilter;

import java.io.File;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.Iterator;

import java.util.LinkedHashMap;

import javax.imageio.ImageIO;

public class DecodeCaptcha {

private static BufferedImage[] templates = new BufferedImage[10];

private static boolean debug = false;

private static boolean show = true;

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

int numTests = 100;

// read in the templates for each character

for (int i=0; i<10; i++) { // for each possible character

templates[i] = ImageIO.read(new File("template" + i+ ".gif"));

}

int correct = 0;

for (int i=0; i<numTests; i++) {

// generate a random string

String rightanswer = getRandomString(6);

// make a captcha from it

BufferedImage captchaImage = generateCaptcha(rightanswer);

if (show) {

ImageIO.write(captchaImage, "jpeg", new File("see.jpg"));

show = false;

}

// guess the captcha

String guess = guessCaptcha(captchaImage);

// see if we got it right

if (guess.equals(rightanswer)) correct++;

}

System.out.println("accuracy: " + (correct / (double) numTests));

}

public static String guessCaptcha(BufferedImage testImage) throws Exception {

String guess = "";

// convert the image from a jpeg to a gif

BufferedImage testImageGif = jpegToGif(testImage);

// cut the CAPTCHA image into subimages that hopefully contain one

// character each

// note: hardcoding the number of characters is not very robust

BufferedImage[] subImages = cutUpImage(testImageGif, 6);

//Printing of the subimages formed by cutting the captcha image

for(int i=0;i<6;i++)

{

ImageIO.write(subImages[i], "gif", new File("see"+i+".gif"));

}

// for each of these subimages, clean it up and then try to guess

// the character

//Printing of the normalized image and noisefiltered image

int k=0;

for (BufferedImage subImage: subImages) {

normalizeColor(subImage);

ImageIO.write(subImage, "gif", new File("colorNormalized"+k+".gif"));

filterNoise(subImage);

int bestGuess = makeGuess(subImage);

ImageIO.write(subImage, "gif", new File("noiseFiltered"+k+".gif"));

guess += "" + bestGuess;

k++;

}

return guess;

}

private static BufferedImage jpegToGif(BufferedImage image) {

// this is pretty kludgy... we're just going to write the jpeg

// out as a gif and read it back in

BufferedImage gifImage = image;

try {

ImageIO.write(image, "gif", new File("temp.gif"));

gifImage = ImageIO.read(new File("temp.gif"));

} catch (Exception e) {e.printStackTrace();}

return gifImage;

}

private static BufferedImage[] cutUpImage(BufferedImage image, int numSlices) {

BufferedImage[] subimages = new BufferedImage[numSlices];

int origHeight = image.getHeight();

int origWidth = image.getWidth();

int width = origWidth / numSlices;

for (int i=0; i<numSlices; i++) {

subimages[i] = image.getSubimage(i\*width, 0, width, origHeight);

}

return subimages;

}

private static void normalizeColor(BufferedImage image) {

// count the number of pixels of each color in the image

HashMap<Integer, Integer> counts = colorHistogram(image);

//Sorting all the color values obtained due to color Histogram function

HashMap<Integer, Integer> sortcount=sortHashMap(counts);

ArrayList keyvalues=new ArrayList(m.keySet());

/\*

System.out.println("sortedHashMap");

for(Integer i:sortcount.keySet())

{

System.out.println(i+":"+sortcount.get(i));

}

for(int i=0;i<keyvalues.size();i++)

{

System.out.println(keyvalues.get(i));

}

\*/

// find all values greater than 50 (a non-robust hardcoded parameter)

ArrayList<Integer> topValues = new ArrayList<>();

//Adding the 2nd highest color value as this indicates the number while the highest color indicates the Background color

for (Integer i: sortcount.keySet()) {

//System.out.println(m.get(i)+"::");

if (i == keyvalues.get(keyvalues.size()-2) || sortcount.get(i)<=50) {

//System.out.println(i+"::"+keyvalues.get(keyvalues.size()-2));

topValues.add(i);

}

}

//Undo the remove frequent value

// remove the most frequent value from topValues, as we are hoping

// it's the background color

/\*

Integer maxFreq = 0;

Integer topColor = 0;

for (Integer i: counts.keySet()) {

if (counts.get(i) > maxFreq) {

maxFreq = counts.get(i);

topColor = i;

}

}

topValues.remove(topColor);

\*/

// create a new image with the most second most frequent color black

// and all the others yellow

int white\_rgb = Color.YELLOW.getRGB();

int black\_rgb = Color.BLACK.getRGB();

for (int x=image.getWidth()-1; x>0; x--) {

for (int y=image.getHeight()-1; y>0; y--) {

int pixelVal = image.getRGB(x, y);

if (!topValues.contains(pixelVal)) {

image.setRGB(x, y, white\_rgb);

} else {

image.setRGB(x, y, black\_rgb);

}

}

}

if (debug) {

try {

ImageIO.write(image, "gif", new File("colorNormalized.gif"));

} catch (Exception e) {e.printStackTrace();}

}

}

private static void filterNoise(BufferedImage image) {

// try to clean up the image by removing stray marks

for (int x=0; x<image.getWidth(); x++) {

for (int y=0; y<image.getHeight(); y++) {

int pixelVal = image.getRGB(x, y);

// check how many pixels in a 2 x 2 rectangle with this point

// in the center have the same color as this point; if not

// many, flip this pixel's color

int startX = Math.max(x-2, 0);

int startY = Math.max(y-2, 0);

int endX = Math.min(x+2, image.getWidth()-1);

int endY = Math.min(y+2, image.getHeight()-1);

int matchCount = 0;

int totalCount = 0;

for (int i=startX; i<=endX; i++) {

for (int j=startY; j<=endY; j++) {

if (image.getRGB(i,j) == pixelVal) {

matchCount++;

}

totalCount++;

}

}

if ((matchCount / (double) totalCount) <.2) {

if (pixelVal == Color.YELLOW.getRGB()) {

image.setRGB(x, y, Color.BLACK.getRGB());

} else {

image.setRGB(x, y, Color.YELLOW.getRGB());

}

}

}

}

if (debug) {

try {

ImageIO.write(image, "gif", new File("noiseFiltered.gif"));

} catch (Exception e) {e.printStackTrace();}

}

}

public static int makeGuess(BufferedImage subImage) {

// check the degree of overlap between each character template and the image

double bestOverlap = -1;

int bestGuess = -1;

for (int i=0; i<templates.length; i++) { // for each possible character

int totalCount = 0;

int matchCount = 0;

for (int x=subImage.getWidth()-1; x>0; x--) {

for (int y=subImage.getHeight()-1; y>0; y--) {

int pixelVal = subImage.getRGB(x, y);

if (!isBlack(pixelVal)) continue;

if (isBlack(templates[i].getRGB(x, y))) matchCount++;

totalCount++;

}

}

if (debug)

System.out.println(i + ": matched " + matchCount + " / " + totalCount);

double overlap = matchCount / (double) totalCount;

if (overlap > bestOverlap) {

bestOverlap = overlap;

bestGuess = i;

}

}

return bestGuess;

}

private static HashMap<Integer, Integer> colorHistogram(BufferedImage image) {

HashMap<Integer, Integer> counts = new HashMap<>();

for (int x=0; x<image.getWidth(); x++) {

for (int y=0; y<image.getHeight(); y++) {

int pixelVal = image.getRGB(x, y);

if (!counts.containsKey(pixelVal)) {

counts.put(pixelVal, 1);

} else {

counts.put(pixelVal, counts.get(pixelVal)+1);

}

}

}

if (debug) {

for (Integer i: counts.keySet()) {

System.out.println(i + ": " + counts.get(i));

}

}

return counts;

}

private static boolean isBlack(int value) {

return (16777216 - Math.abs(value)) / (double) 16777216 < .05;

}

private static String getRandomString(int length) {

String val = "";

for (int i=0; i<length; i++) {

double rand = Math.random();

int randInt = (int) (rand \* 10);

val += String.valueOf(randInt);

}

return val;

}

private static BufferedImage generateCaptcha(String answer) {

SkewImage skewImage = new SkewImage();

return skewImage.skewImage(answer);

}

//Sorting all the color values

public static LinkedHashMap sortHashMap(HashMap passedMap)

{

ArrayList mapKeys = new ArrayList(passedMap.keySet());

ArrayList mapValues = new ArrayList(passedMap.values());

Collections.sort(mapKeys);

Collections.sort(mapValues);

LinkedHashMap sortedMap = new LinkedHashMap();

Iterator valueIt=mapValues.iterator();

while (valueIt.hasNext()) {

Object val = valueIt.next();

Iterator keyIt = mapKeys.iterator();

while (keyIt.hasNext()) {

Object key = keyIt.next();

String comp1 = passedMap.get(key).toString();

String comp2 = val.toString();

if (comp1.equals(comp2)){

passedMap.remove(key);

mapKeys.remove(key);

sortedMap.put(key, val);

break;

}

}

}

return sortedMap;

}

}

**SkewImage.java**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Captcha;

/\*\*

\*

\* @author lingg

\*/

import java.awt.image.BufferedImage;

import java.awt.\*;

import java.awt.geom.AffineTransform;

/\*\*

\* <a href="http://skewpassim.sourceforge.net/">http://skewpassim.sourceforge.net/</a>

\* <br>

\* <b>This is a sample implementation of the ISkewImage class

\* in order to skew the secured chars passed encoded as a parameter</b>

\*/

public class SkewImage {

private static final int MAX\_LETTER\_COUNT = 6;

private static final int LETTER\_WIDTH = 33;

private static final int IMAGE\_HEIGHT = 40;

private static final double SKEW = 0.0;

private static final int DRAW\_LINES = 2;

private static final int DRAW\_BOXES = 2;

private static final int MAX\_X = LETTER\_WIDTH \* MAX\_LETTER\_COUNT;

private static final int MAX\_Y = IMAGE\_HEIGHT;

private static final Color [] RANDOM\_BG\_COLORS = {

Color.RED, Color.CYAN, Color.GREEN, Color.MAGENTA, Color.ORANGE, Color.PINK};

private static final Color [] RANDOM\_FG\_COLORS = {Color.BLACK, Color.BLUE, Color.DARK\_GRAY};

public BufferedImage skewImage(String securityChars) {

BufferedImage outImage = new BufferedImage(MAX\_X, MAX\_Y,

BufferedImage.TYPE\_INT\_RGB);

Graphics2D g2d = outImage.createGraphics();

g2d.setColor(java.awt.Color.WHITE);

g2d.fillRect(0, 0, MAX\_X, MAX\_Y);

for (int i = 0; i < DRAW\_BOXES; i++) {

paindBoxes(g2d);

}

Font font = new Font("dialog", 1, 33);

g2d.setFont(font);

g2d.setColor(Color.BLACK);

//Removing all the border color of the image

g2d.drawRect(0, 0, 0, 0);

AffineTransform affineTransform = new AffineTransform();

for (int i = 0; i < MAX\_LETTER\_COUNT; i++) {

double angle = 0;

if (Math.random() \* 2 > 1) {

angle = Math.random() \* SKEW;

} else {

angle = Math.random() \* -SKEW;

}

affineTransform.rotate(angle, (LETTER\_WIDTH \* i) + (LETTER\_WIDTH / 2), MAX\_Y / 2);

g2d.setTransform(affineTransform);

setRandomFont(g2d);

setRandomFGColor(g2d);

g2d.drawString(securityChars.substring(i, i + 1),

// (i \* LETTER\_WIDTH) + 3, 28 + (int) (Math.random() \* 6));

(i \* LETTER\_WIDTH) + 3, 31 + (int) (Math.random() \* 0));

affineTransform.rotate(-angle, (LETTER\_WIDTH \* i) + (LETTER\_WIDTH / 2), MAX\_Y / 2);

}

for (int i = 0; i < DRAW\_LINES; i ++) {

g2d.setXORMode(Color.RED);

setRandomBGColor(g2d);

g2d.setStroke(new BasicStroke(4));

int y1 = (int) (Math.random() \* MAX\_Y);

int y2 = (int) (Math.random() \* MAX\_Y);

g2d.drawLine(0, y1, MAX\_X, y2);

}

return outImage;

}

private void paindBoxes(Graphics2D g2d) {

int colorId = (int) (Math.random() \* RANDOM\_BG\_COLORS.length);

g2d.setColor(RANDOM\_BG\_COLORS[colorId]);

g2d.fillRect(getRandomX(), getRandomY(),

getRandomX(), getRandomY());

}

private int getRandomX() {

return (int) (Math.random() \* MAX\_X);

}

private int getRandomY() {

return (int) (Math.random() \* MAX\_Y);

}

private void setRandomFont(Graphics2D g2d) {

Font font = new Font("dialog", 1, 33);

g2d.setFont(font);

}

private void setRandomFGColor(Graphics2D g2d) {

int colorId = (int) (Math.random() \* RANDOM\_FG\_COLORS.length);

g2d.setColor(RANDOM\_FG\_COLORS[colorId]);

}

private void setRandomBGColor(Graphics2D g2d) {

int colorId = (int) (Math.random() \* RANDOM\_BG\_COLORS.length);

g2d.setColor(RANDOM\_BG\_COLORS[colorId]);

}

}

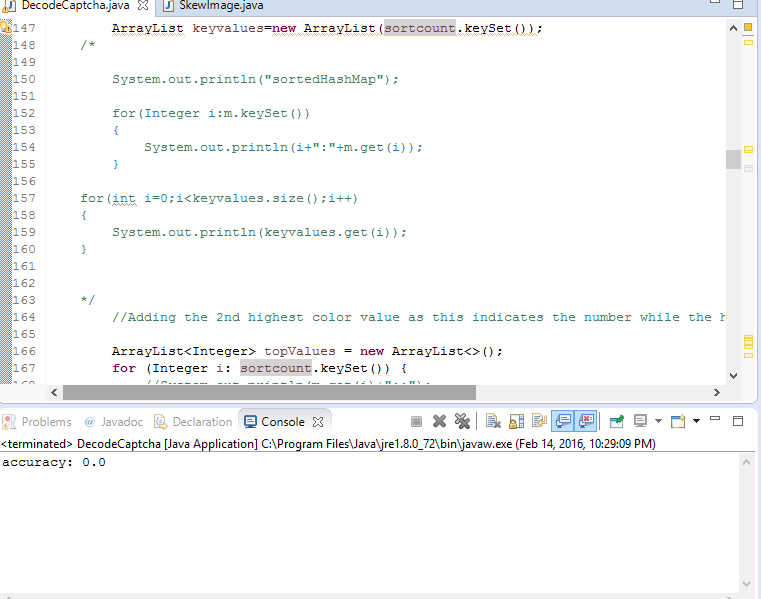
Sorted Code Ref:

**Execution Steps:**

**Start Eclipse**

**File->Import->General->ExistingProjectsintoWorkSpace->Captcha**

Output in windows



Output in Ubuntu:

