```java
package test;
import java.util.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.*;
public class ImageModificationClass {

    private BufferedImage image;
    private BufferedImage rawImage;
    private int tempRoot1;
    private int tempRoot2;
    private int tempRoot3;
    private int tempRoot4;
    private static int pixelSize;
    private int tempRoot5;
    private int tempRoot6;

    private int tempA1;
    private int tempA2;
    private int tempA3;
    private int tempA4;
    private int tempA5;
    private int tempA6;

    private int tempC1;
    private int tempC2;
    private int tempD1;
    private int tempD2;
    private int tempK1;
    private int tempK2;

    private static int safePrime;





    public int getSafePrime () {
            return safePrime;
    }

    public ArrayList<Integer> tempRootArray() {
        ArrayList<Integer> result = new ArrayList<Integer>();
        result.add(tempRoot1);
        result.add(tempRoot2);
        result.add(tempRoot3);
        result.add(tempRoot4);
        result.add(tempRoot5);
        result.add(tempRoot6);
        return result;


    }

    public ArrayList<Integer> enhancedTempRootArray() {
        ArrayList<Integer> result = new ArrayList<Integer>();
        result.add(tempRoot1);
        result.add(tempRoot2);
        result.add(tempRoot3);
        result.add(tempRoot4);
        result.add(tempRoot5);
        result.add(tempRoot6);
        result.add(tempA1);
        result.add(tempA2);
        result.add(tempA3);
        result.add(tempA4);
        result.add(tempA5);
        result.add(tempA6);
        return result;


    }

    public ArrayList<Integer> linearTempRootArray() {
        ArrayList<Integer> result = new ArrayList<Integer>();
        result.add(tempC1);
        result.add(tempC2);
        result.add(tempD1);
        result.add(tempD2);
        result.add(tempK1);
        result.add(tempK2);
        return result;


    }
    private int [] safePrimes = {5, 7, 11, 23, 47, 59, 83, 107, 167, 179, 227, 263, 347, 359, 383,
467, 479, 503, 563, 587, 719, 839, 863, 887, 983, 1019, 1187, 1283, 1307, 1319, 1367, 1439,
1487, 1523, 1619, 1823, 1907, 2027, 2039, 2063, 2099, 2207, 2447, 2459, 2579, 2819, 2879,
2903, 2963, 2999, 3023, 3119, 3167, 3203, 3467, 3623, 3779, 3803, 3863, 3947, 4007, 4079,
4127, 4139, 4259, 4283, 4547, 4679, 4703, 4787, 4799, 4919};
    public void setSafePrimeIndex (int spIndex) {
        safePrime = safePrimes[spIndex];
    }

    public ArrayList<Integer> getSafePrimesList () {
        ArrayList<Integer> returnedList = new ArrayList<Integer>();
        for (int i = 0; i < safePrimes.length; i++) {
            returnedList.add(safePrimes[i]);
        }
        return returnedList;
    }

    public void setMosaicPixelSize (int pxSize) {
        pixelSize = pxSize;
    }

    private ArrayList<BufferedImage> approxImages = new ArrayList<BufferedImage>();

    public void setApproxImages (String directoryName, int size) throws IOException {
        File folder = new File (directoryName);
        for (File approxImage : folder.listFiles()) {
            BufferedImage bi = ImageIO.read(approxImage);
            BufferedImage addedImage = resizeImage(bi, size, size);
            approxImages.add(addedImage);
        }

    }

    public ArrayList<BufferedImage> getApproxImages () {
        return approxImages;
    }

    private ArrayList<Color> computeAverages (ArrayList<BufferedImage> input) {
        ArrayList<Color> result = new ArrayList<Color>();
        for (int i = 0; i < input.size(); i++) {
            result.add(computeAverage(input.get(i)));
        }
        return result;
    }

    private static Color computeAverage (BufferedImage bi) {
        long rSum = 0;
        long gSum = 0;
        long bSum = 0;
        for (int i = 0; i < bi.getWidth(); i++) {
            for (int j = 0; j < bi.getHeight(); j++) {
                    Color c = new Color (bi.getRGB(i, j));
                    rSum += c.getRed();
                    gSum += c.getGreen();
                    bSum += c.getBlue();
            }
        }
        int totalPixels = bi.getWidth() * bi.getHeight();
        return new Color((int) (rSum / totalPixels), (int) (gSum / totalPixels), (int) (bSum /
totalPixels));
    }

    private int determineClosestIndex (Color input) {
        ArrayList<Color> colorBank = computeAverages (approxImages);

        int minDistance = 99999999;
        int minDistanceIndex = 0;

        for (int i = 0; i < colorBank.size(); i++) {
            Color candidate = colorBank.get(i);
            double candidateRed = candidate.getRed();
            double candidateGreen = candidate.getGreen();
            double candidateBlue = candidate.getBlue();
            double red = input.getRed();
            double green = input.getGreen();
            double blue = input.getBlue();
            double squared = Math.pow(candidateRed - red, 2) + Math.pow(candidateGreen
- green, 2) + Math.pow(candidateBlue - blue, 2);
            double distance = Math.sqrt(squared);

            if (distance < minDistance) {
                    minDistance = (int) distance;
                    minDistanceIndex = i;
            }

        }
```

```
        return minDistanceIndex;


    }

    public void emojify (int width, int height, int emojiSize) throws IOException {
        //read image
        File f = null;
         try{
                f = new File(fileName); //image file path
                rawImage = new BufferedImage((safePrime - 1), (safePrime - 1),
BufferedImage.TYPE_INT_ARGB);
                rawImage = ImageIO.read(f);
                image = resizeImage(rawImage, width*emojiSize, height*emojiSize);
                System.out.println("Reading complete.");
         }catch(IOException e){
                System.out.println("Error: "+e);
         }

         for (int x = 0; x < width; x++) {
                for (int y = 0; y < height; y++) {

                        //compute average RGB of cell
                        BufferedImage sub = image.getSubimage(x * emojiSize, y*emojiSize,
emojiSize, emojiSize);

                        Color inputColor = computeAverage (sub);
                        int index = determineClosestIndex (inputColor);
                        BufferedImage replace = approxImages.get(index);


                        for (int i = x*emojiSize; i < (x+1)*emojiSize; i++) {
                                for (int j = y*emojiSize; j < (y+1)*emojiSize; j++) {
                                        int newRGB = replace.getRGB(i % emojiSize, j %
emojiSize);
                                        image.setRGB(i, j, newRGB);


                        }
                        }

                }
         }
         System.out.println("Image emojified with " + width + " emojis by " + height + " emojis. ");
        ImageIO.write(image, "png", f);
        System.out.println("Writing complete.");

}



private String fileName;
public ImageModificationClass (String fn) {
        fileName = fn;

}

public ImageModificationClass (BufferedImage im) {
        rawImage = im;

}

public BufferedImage getImage () {
        return image;
}


        public void enhancedMosaicEncrypt () throws IOException{
                 Random random = new Random();
                int width = (safePrime - 1) * pixelSize;    //width of the image
                int height = (safePrime - 1) * pixelSize;   //height of the image
                //generate the primitive roots mod safePrime

                ArrayList <Integer> proots = new ArrayList <Integer> ();
                for (int i = 0; i < (safePrime - 1); i++) {
                        proots.add(i);
                }
                for (int i = 0; i < safePrime; i++) {
                        int j = proots.indexOf((i*i) % safePrime);
                        if (j != -1) {
                                proots.remove(j);
                        }

                }
                int randomIndex1 = random.nextInt(proots.size());
                int randomIndex2 = random.nextInt(proots.size());
                int randomIndex3 = random.nextInt(proots.size());
                int randomIndex4 = random.nextInt(proots.size());
                int randomIndex5 = random.nextInt(proots.size());
                int randomIndex6 = random.nextInt(proots.size());

                int proot1 = proots.get(randomIndex1);
                int proot2 = proots.get(randomIndex2);
                int proot3 = proots.get(randomIndex3);
                int proot4 = proots.get(randomIndex4);
                int proot5 = proots.get(randomIndex5);
                int proot6 = proots.get(randomIndex6);

                ArrayList <Integer> relPrimes = new ArrayList <Integer> ();
                for (int i = 0; i < (safePrime - 1); i++) {
                        if (((i % 2) == 1) && (i != (safePrime - 1) / 2)) {
                                relPrimes.add(i);
                        }
                }

                int aIndex1 = random.nextInt(relPrimes.size());
                int aIndex2 = random.nextInt(relPrimes.size());
                int aIndex3 = random.nextInt(relPrimes.size());
                int aIndex4 = random.nextInt(relPrimes.size());
                int aIndex5 = random.nextInt(relPrimes.size());
                int aIndex6 = random.nextInt(relPrimes.size());

                int a1 = relPrimes.get(aIndex1);
                int a2 = relPrimes.get(aIndex2);
                int a3 = relPrimes.get(aIndex3);
                int a4 = relPrimes.get(aIndex4);
                int a5 = relPrimes.get(aIndex5);
                int a6 = relPrimes.get(aIndex6);

                 tempRoot1 = proot1;
                 tempRoot2 = proot2;
                 tempRoot3 = proot3;
                 tempRoot4 = proot4;
                 tempRoot5 = proot5;
                 tempRoot6 = proot6;

                 tempA1 = a1;
                 tempA2 = a2;
                 tempA3 = a3;
                 tempA4 = a4;
                 tempA5 = a5;
                 tempA6 = a6;
```

```
                File f = null;

                //read image
                try{
                f = new File(fileName); //image file path
                rawImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);
                rawImage = ImageIO.read(f);
                image = resizeImage(rawImage, width, height);
                System.out.println("Reading complete.");
                }catch(IOException e){
                System.out.println("Error: "+e);
                }

                BufferedImage scrambledImage = mosaicScramble(image, proot1);

                BufferedImage scrambledImage2 = mosaicScramble2(scrambledImage, proot2);

                BufferedImage scrambledImage3 = mosaicScrambleD1(scrambledImage2,
proot3, proot4);

                BufferedImage scrambledImage4 = mosaicScrambleD2(scrambledImage3,
proot5, proot6);

                BufferedImage scrambledImage5 = funnyMosaicScramble(scrambledImage4,
a1);

                BufferedImage scrambledImage6 = funnyMosaicScramble2(scrambledImage5,
a2);

                BufferedImage scrambledImage7 = funnyMosaicScrambleD1
(scrambledImage6, a3, a4);

                BufferedImage scrambledImage8 = funnyMosaicScrambleD2
(scrambledImage7, a5, a6);

                System.out.println("Image Scrambled.");

                image = scrambledImage8;

                ImageIO.write(image, "png", f);
```

```
                    System.out.println("Image writing complete.");


                    String secretCode = proot1 + " " + proot2 + " " + proot3 + " " + proot4 + " " +
proot5 + " " + proot6 + " " + a1 + " " + a2 + " " + a3 + " " + a4 + " " + a5 + " " + a6;

                    System.out.println("Your decryption code is: \n" + secretCode + ". \nKeep this
code to yourself but don't lose it!");


          }

          private ArrayList<Integer> matrix;

          public ImageModificationClass (ArrayList<Integer> mt) {
                    matrix = mt;
          }

          public ArrayList<Integer> getMatrix() {
                    return matrix;
          }



          public void enhancedMatrixEncryptParams (int pr1, int pr2, int pr3, int pr4, int pr5, int pr6,
int a1, int a2, int a3, int a4, int a5, int a6) throws IOException{

                    ArrayList<Integer> scrambledImage = matrixScramble(matrix, pr1);

                    ArrayList<Integer> scrambledImage2 = matrixScramble2(scrambledImage, pr2);

                    ArrayList<Integer> scrambledImage3 = matrixScrambleD1(scrambledImage2,
pr3, pr4);

                    ArrayList<Integer> scrambledImage4 = matrixScrambleD2(scrambledImage3,
pr5, pr6);

                    ArrayList<Integer> scrambledImage5 = funnyMatrixScramble(scrambledImage4,
a1);

                    ArrayList<Integer> scrambledImage6 =
funnyMatrixScramble2(scrambledImage5, a2);
```

```
                    ArrayList<Integer> scrambledImage7 = funnyMatrixScrambleD1
(scrambledImage6, a3, a4);

                    ArrayList<Integer> scrambledImage8 = funnyMatrixScrambleD2
(scrambledImage7, a5, a6);

                    matrix = scrambledImage8;




          }


          public void enhancedMatrixEncryptWithAffineParams (int pr1, int pr2, int pr3, int pr4, int
pr5, int pr6, int a1, int a2, int a3, int a4, int a5, int a6, int c1, int c2, int d1, int d2, int k1, int k2)
throws IOException{

                   int sophieGermainPrime = (safePrime - 1) / 2;

                    ArrayList<Integer> scrambledImage = matrixScramble(matrix, pr1);

                    ArrayList<Integer> scrambledImage2 = matrixScramble2(scrambledImage, pr2);

                    ArrayList<Integer> scrambledImage3 = matrixScrambleD1(scrambledImage2,
pr3, pr4);

                    ArrayList<Integer> scrambledImage4 = matrixScrambleD2(scrambledImage3,
pr5, pr6);

                    ArrayList<Integer> scrambledImage5 = funnyMatrixScramble(scrambledImage4,
a1);

                    ArrayList<Integer> scrambledImage6 =
funnyMatrixScramble2(scrambledImage5, a2);

                    ArrayList<Integer> scrambledImage7 = funnyMatrixScrambleD1
(scrambledImage6, a3, a4);

                    ArrayList<Integer> scrambledImage8 = funnyMatrixScrambleD2
(scrambledImage7, a5, a6);

                    ArrayList<Integer> scrambledImage9 = linearMatrixScramble
(scrambledImage8, c1, c2, d1, d2, k1, k2, sophieGermainPrime);
```

```
            matrix = scrambledImage9;




}


public void enhancedMosaicEncryptWithAffine () throws IOException{
             Random random = new Random();
            int width = (safePrime - 1) * pixelSize;    //width of the image
            int height = (safePrime - 1) * pixelSize;   //height of the image
            //generate the primitive roots mod safePrime

            ArrayList <Integer> proots = new ArrayList <Integer> ();
            for (int i = 0; i < (safePrime - 1); i++) {
                        proots.add(i);
            }
            for (int i = 0; i < safePrime; i++) {
                        int j = proots.indexOf((i*i) % safePrime);
                        if (j != -1) {
                                    proots.remove(j);
                        }

            }
            int randomIndex1 = random.nextInt(proots.size());
            int randomIndex2 = random.nextInt(proots.size());
            int randomIndex3 = random.nextInt(proots.size());
            int randomIndex4 = random.nextInt(proots.size());
            int randomIndex5 = random.nextInt(proots.size());
            int randomIndex6 = random.nextInt(proots.size());

            int proot1 = proots.get(randomIndex1);
            int proot2 = proots.get(randomIndex2);
            int proot3 = proots.get(randomIndex3);
            int proot4 = proots.get(randomIndex4);
            int proot5 = proots.get(randomIndex5);
            int proot6 = proots.get(randomIndex6);

            ArrayList <Integer> relPrimes = new ArrayList <Integer> ();
            for (int i = 0; i < (safePrime - 1); i++) {
                       if (((i % 2) == 1) && (i != (safePrime - 1) / 2)) {
                                  relPrimes.add(i);
```

```
        }
}

int aIndex1 = random.nextInt(relPrimes.size());
int aIndex2 = random.nextInt(relPrimes.size());
int aIndex3 = random.nextInt(relPrimes.size());
int aIndex4 = random.nextInt(relPrimes.size());
int aIndex5 = random.nextInt(relPrimes.size());
int aIndex6 = random.nextInt(relPrimes.size());

int a1 = relPrimes.get(aIndex1);
int a2 = relPrimes.get(aIndex2);
int a3 = relPrimes.get(aIndex3);
int a4 = relPrimes.get(aIndex4);
int a5 = relPrimes.get(aIndex5);
int a6 = relPrimes.get(aIndex6);

 tempRoot1 = proot1;
 tempRoot2 = proot2;
 tempRoot3 = proot3;
 tempRoot4 = proot4;
 tempRoot5 = proot5;
 tempRoot6 = proot6;

 tempA1 = a1;
 tempA2 = a2;
 tempA3 = a3;
 tempA4 = a4;
 tempA5 = a5;
 tempA6 = a6;


 int sophieGermainPrime = (safePrime - 1) / 2;
 int c1 = 1;
 int c2 = 1;
int d1 = 1;
int d2 = 1;

while (!independent(c1, c2, d1, d2, sophieGermainPrime)) {
        c1 = random.nextInt(sophieGermainPrime - 1) + 1;
        c2 = random.nextInt(sophieGermainPrime - 1) + 1;
        d1 = random.nextInt(sophieGermainPrime - 1) + 1;
        d2 = random.nextInt(sophieGermainPrime - 1) + 1;
}
```

```
                    tempC1 = c1;
                    tempC2 = c2;
                    tempD1 = d1;
                    tempD2 = d2;

                    int k1 = random.nextInt(sophieGermainPrime);
                    int k2 = random.nextInt(sophieGermainPrime);
                    File f = null;

                    //read image
                    try{
                    f = new File(fileName); //image file path
                    rawImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);
                    rawImage = ImageIO.read(f);
                    image = resizeImage(rawImage, width, height);
                    System.out.println("Reading complete.");
                    }catch(IOException e){
                    System.out.println("Error: "+e);
                    }

                    BufferedImage scrambledImage = mosaicScramble(image, proot1);

                    BufferedImage scrambledImage2 = mosaicScramble2(scrambledImage, proot2);

                    BufferedImage scrambledImage3 = mosaicScrambleD1(scrambledImage2,
proot3, proot4);

                    BufferedImage scrambledImage4 = mosaicScrambleD2(scrambledImage3,
proot5, proot6);

                    BufferedImage scrambledImage5 = funnyMosaicScramble(scrambledImage4,
a1);

                    BufferedImage scrambledImage6 = funnyMosaicScramble2(scrambledImage5,
a2);

                    BufferedImage scrambledImage7 = funnyMosaicScrambleD1
(scrambledImage6, a3, a4);

                    BufferedImage scrambledImage8 = funnyMosaicScrambleD2
(scrambledImage7, a5, a6);
```

```
                    BufferedImage scrambledImage9 = linearScramble (scrambledImage8, c1, c2,
d1, d2, k1, k2, sophieGermainPrime);

                    System.out.println("Image Scrambled.");

                    image = scrambledImage9;

                    ImageIO.write(image, "png", f);
                    System.out.println("Image writing complete.");


                    String secretCode = proot1 + " " + proot2 + " " + proot3 + " " + proot4 + " " +
proot5 + " " + proot6 + " " + a1 + " " + a2 + " " + a3 + " " + a4 + " " + a5 + " " + a6 + " " + c1 + " " +
c2 + " " + d1 + " " + d2 + " " + k1 + " " + k2;

                    System.out.println("Your decryption code is: \n" + secretCode + ". \nKeep this
code to yourself but don't lose it!");


          }

          public void linearEncrypt () throws IOException{
                     Random random = new Random();
                    int width = (safePrime) * pixelSize;    //width of the image
                    int height = (safePrime) * pixelSize;   //height of the image

                    int c1 = 1;
                    int c2 = 1;
                    int d1 = 1;
                    int d2 = 1;

                    while (!independent(c1, c2, d1, d2, safePrime)) {
                              c1 = random.nextInt(safePrime - 1) + 1;
                              c2 = random.nextInt(safePrime - 1) + 1;
                              d1 = random.nextInt(safePrime - 1) + 1;
                              d2 = random.nextInt(safePrime - 1) + 1;
                    }

                    tempC1 = c1;
                    tempC2 = c2;
                    tempD1 = d1;
                    tempD2 = d2;

                    int k1 = random.nextInt(safePrime);
```

```
		int k2 = random.nextInt(safePrime);


		File f = null;

		//read image
		try{
		f = new File(fileName); //image file path
		rawImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);
		rawImage = ImageIO.read(f);
		image = resizeImage(rawImage, width, height);
		System.out.println("Reading complete.");
		}catch(IOException e){
		System.out.println("Error: "+e);
		}

		BufferedImage scrambledImage = linearScramble(image, c1, c2, d1, d2, k1, k2,
safePrime);


		System.out.println("Image Scrambled.");

		image = scrambledImage;

		ImageIO.write(image, "png", f);
		System.out.println("Image writing complete.");


		String secretCode = c1 + " " + c2 + " " + d1 + " " + d2 + " " + k1 + " " + k2;

		System.out.println("Your decryption code is: \n" + secretCode + ". \nKeep this
code to yourself but don't lose it!");


	}

	private BufferedImage affineEncrypt (BufferedImage original, int c1, int c2, int d1, int d2,
int k1, int k2, int sp) throws IOException {
		BufferedImage scrambledImage = linearScramble(original, c1, c2, d1, d2, k1, k2,
sp);
		return scrambledImage;
	}
```

```
        public void enhancedMosaicDecrypt () throws IOException{
                Scanner scan = new Scanner(System.in);

               System.out.println("Enter the exact safe prime used: ");
               String im = scan.nextLine();
                safePrime = Integer.parseInt(im);

                int width = (safePrime - 1) * pixelSize;    //width of the image
                int height = (safePrime - 1) * pixelSize;   //height of the image
                BufferedImage image = null;
                File f = null;

                //read image
                try{
                        f = new File(fileName); //image file path
                        BufferedImage rawImage = new BufferedImage(width, height,
    BufferedImage.TYPE_INT_ARGB);
                        rawImage = ImageIO.read(f);
                        image = resizeImage(rawImage, width, height);
                        System.out.println("Reading complete.");
                }catch(IOException e){
                        System.out.println("Error: "+e);
                }



                System.out.println("Enter the decryption code for that image: ");

                String codestring = scan.nextLine();
                String[] rootsArray = codestring.split(" ");

                int root1 = Integer.parseInt(rootsArray[0]);
                int root2 = Integer.parseInt(rootsArray[1]);
                int root3 = Integer.parseInt(rootsArray[2]);
                int root4 = Integer.parseInt(rootsArray[3]);
                int root5 = Integer.parseInt(rootsArray[4]);
                int root6 = Integer.parseInt(rootsArray[5]);

                int a1 = Integer.parseInt(rootsArray[6]);
                int a2 = Integer.parseInt(rootsArray[7]);
                int a3 = Integer.parseInt(rootsArray[8]);
                int a4 = Integer.parseInt(rootsArray[9]);
                int a5 = Integer.parseInt(rootsArray[10]);
```

```
                    int a6 = Integer.parseInt(rootsArray[11]);


                    BufferedImage unscrambledImage1 = funnyMosaicUnscrambleD2 (image, a5,
a6);
                    BufferedImage unscrambledImage2 = funnyMosaicUnscrambleD1
(unscrambledImage1, a3, a4);
                    BufferedImage unscrambledImage3 = funnyMosaicUnscrambleRows
(unscrambledImage2, a2);
                    BufferedImage unscrambledImage4 = funnyMosaicUnscrambleCols
(unscrambledImage3, a1);

                    BufferedImage unscrambledImage5 = mosaicUnscrambleD2
(unscrambledImage4, root5, root6);
                    BufferedImage unscrambledImage6 = mosaicUnscrambleD1
(unscrambledImage5, root3, root4);
                    BufferedImage unscrambledImage7 = mosaicUnscrambleRows
(unscrambledImage6, root2);
                    BufferedImage unscrambledImage8 = mosaicUnscrambleCols
(unscrambledImage7, root1);

                    System.out.println("Image unscrambled.");


                    ImageIO.write(unscrambledImage8, "png", f);
                    System.out.println("Writing complete.");

                    image = unscrambledImage8;


                   }


          private static int prToThe (int base, int k) {
            int value = 1;
            for (int i = 0; i < k; i++) {
                      value*=base;
                      value%=safePrime;
            }
            return value;
          }

          static BufferedImage resizeImage(BufferedImage originalImage, int targetWidth, int
targetHeight) throws IOException {
```

```
            BufferedImage resizedImage = new BufferedImage(targetWidth, targetHeight,
BufferedImage.TYPE_INT_RGB);
            Graphics2D graphics2D = resizedImage.createGraphics();
            graphics2D.drawImage(originalImage, 0, 0, targetWidth, targetHeight, null);
            graphics2D.dispose();
            return resizedImage;
           }



    public void polynomialEncrypt () throws IOException {
           Random random = new Random();
                     int width = (safePrime) * pixelSize;    //width of the image
                     int height = (safePrime) * pixelSize;   //height of the image
                     //generate the primitive roots mod safePrime

                     ArrayList<ArrayList<Integer>> basis = getPolynomialBasis();
                     ArrayList<ArrayList<Integer>> polynomials = new
ArrayList<ArrayList<Integer>>();

                     for (int i = 0; i < 6; i++) {
                               int randomIndex1 = random.nextInt(basis.size());
                               int randomIndex2 = random.nextInt(basis.size());
                               int randomIndex3 = random.nextInt(basis.size());
                               int randomIndex4 = random.nextInt(basis.size());
                               int randomIndex5 = random.nextInt(basis.size());
                               ArrayList<Integer> p1 = basis.get(randomIndex1);
                               ArrayList<Integer> p2 = basis.get(randomIndex2);
                               ArrayList<Integer> p3 = basis.get(randomIndex3);
                               ArrayList<Integer> p4 = basis.get(randomIndex4);
                               ArrayList<Integer> p5 = basis.get(randomIndex5);
                               ArrayList<Integer> polynomial = p1;
                               polynomial = composition(p2, polynomial, safePrime);
                               polynomial = normalize(polynomial);
                               polynomial = composition(p3, polynomial, safePrime);
                               polynomial = normalize(polynomial);
                               polynomial = composition(p4, polynomial, safePrime);
                               polynomial = normalize(polynomial);
                               polynomial = composition(p5, polynomial, safePrime);
                               polynomial = normalize(polynomial);
                               polynomials.add(polynomial);
                     }

                     File f = null;
```

```
                    //read image
                    try{
                    f = new File(fileName); //image file path
                    rawImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);
                    rawImage = ImageIO.read(f);
                    image = resizeImage(rawImage, width, height);
                    System.out.println("Reading complete.");
                    }catch(IOException e){
                    System.out.println("Error: "+e);
                    }

                    BufferedImage scrambledImage = polyMosaicScramble(image,
polynomials.get(0));

                    BufferedImage scrambledImage2 = polyMosaicScramble2(scrambledImage,
polynomials.get(1));

                    BufferedImage scrambledImage3 = polyMosaicScrambleD1(scrambledImage2,
polynomials.get(2), polynomials.get(3));

                    BufferedImage scrambledImage4 = polyMosaicScrambleD2(scrambledImage3,
polynomials.get(4), polynomials.get(5));


                    System.out.println("Image Scrambled.");

                    image = scrambledImage4;

                    ImageIO.write(image, "png", f);
                    System.out.println("Image writing complete.");


                    System.out.println("Polynomials used: ");
                    for (int i = 0; i < polynomials.size(); i++) {
                         System.out.println(polynomials.get(i));
                    }
   }

   public void polynomialDecrypt () throws IOException{
                    Scanner scan = new Scanner(System.in);

                   System.out.println("Enter the exact safe prime used: ");
```

```
                    String im = scan.nextLine();
                     safePrime = Integer.parseInt(im);

                     int width = (safePrime) * pixelSize;    //width of the image
                     int height = (safePrime) * pixelSize;   //height of the image
                     BufferedImage image = null;
                     File f = null;

                     //read image
                     try{
                               f = new File(fileName); //image file path
                               BufferedImage rawImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);
                               rawImage = ImageIO.read(f);
                               image = resizeImage(rawImage, width, height);
                               System.out.println("Reading complete.");
                     }catch(IOException e){
                               System.out.println("Error: "+e);
                     }



                     System.out.println("Enter the coefficients of polynomial 1 separated by spaces:
\n");
                     String codestring0 = scan.nextLine();
                     String[] coeffs0 = codestring0.split("\\s+");
                     ArrayList<Integer> poly0 = toArrayList(coeffs0);

                    System.out.println("Enter the coefficients of polynomial 2 separated by spaces:
\n");
                     String codestring1 = scan.nextLine();
                     String[] coeffs1 = codestring1.split("\\s+");
                     ArrayList<Integer> poly1 = toArrayList(coeffs1);

                    System.out.println("Enter the coefficients of polynomial 3 separated by spaces:
\n");
                     String codestring2 = scan.nextLine();
                     String[] coeffs2 = codestring2.split("\\s+");
                     ArrayList<Integer> poly2 = toArrayList(coeffs2);

                    System.out.println("Enter the coefficients of polynomial 4 separated by spaces:
\n");
                     String codestring3 = scan.nextLine();
                     String[] coeffs3 = codestring3.split("\\s+");
                     ArrayList<Integer> poly3 = toArrayList(coeffs3);

                    System.out.println("Enter the coefficients of polynomial 5 separated by spaces:
\n");
                     String codestring4 = scan.nextLine();
                     String[] coeffs4 = codestring4.split("\\s+");
                     ArrayList<Integer> poly4 = toArrayList(coeffs4);

                    System.out.println("Enter the coefficients of polynomial 6 separated by spaces:
\n");
                     String codestring5 = scan.nextLine();
                     String[] coeffs5 = codestring5.split("\\s+");
                     ArrayList<Integer> poly5 = toArrayList(coeffs5);


                     BufferedImage unscrambledImage1 = polyMosaicUnscrambleD2 (image, poly4,
poly5);
                     BufferedImage unscrambledImage2 = polyMosaicUnscrambleD1
(unscrambledImage1, poly2, poly3);
                     BufferedImage unscrambledImage3 = polyMosaicUnscrambleRows
(unscrambledImage2, poly1);
                     BufferedImage unscrambledImage4 = polyMosaicUnscrambleCols
(unscrambledImage3, poly0);


                     System.out.println("Image unscrambled.");


                     ImageIO.write(unscrambledImage4, "png", f);
                     System.out.println("Writing complete.");

                     image = unscrambledImage4;


                    }

    private static ArrayList<Integer> toArrayList (String[] array) {
          ArrayList<Integer> result = new ArrayList<Integer>();
          for (int i = 0; i < array.length; i++) {
                    result.add(Integer.parseInt(array[i]));
          }
          return result;
    }
          public static int discreteLogBasePrModsafePrime (int base, int k) {
```

```
        int value = 0;
        for (int i = 0; i < safePrime; i++) {
                if (prToThe(base, i) % safePrime == k) {
                        value = i;
                        break;
                }
        }
        return value;
}

public static int discreteLogBasePrModsafePrime_ (int base, int k) {
        return discreteLogarithm(base, k, safePrime);
}




static int discreteLogarithm(int a, int b, int m)
{
int n = (int) (Math.sqrt (m) + 1);

// Calculate a ^ n
int an = 1;
for (int i = 0; i < n; ++i)
an = (an * a) % m;

int[] value=new int[m];

// Store all values of a^(n*i) of LHS
for (int i = 1, cur = an; i <= n; ++i)
{
if (value[ cur ] == 0)
       value[ cur ] = i;
cur = (cur * an) % m;
}

for (int i = 0, cur = b; i <= n; ++i)
{
// Calculate (a ^ j) * b and check
// for collision
if (value[cur] > 0)
{
       int ans = value[cur] * n - i;
```

```
                if (ans < m)
                return ans;
        }
        cur = (cur * a) % m;
        }
        return -1;
        }



        private static ArrayList<BufferedImage> mosaicEncrypt (ArrayList <BufferedImage>
original, int pr) {
          ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
          for (int i = 0; i < original.size(); i++) {
                  newArr.add(original.get(prToThe(pr, i) - 1));

          }
          return newArr;
        }

        private static ArrayList<BufferedImage> mosaicEncrypt2 (ArrayList <BufferedImage>
original, int pr) {
          ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
          for (int i = 0; i < original.size(); i++) {
                  newArr.add(original.get(prToThe(pr, i) - 1));
          }
          return newArr;
        }


        private static ArrayList<Integer> matrixEncrypt (ArrayList <Integer> original, int pr) {
          ArrayList<Integer> newArr = new ArrayList<Integer>();
          for (int i = 0; i < original.size(); i++) {
                  newArr.add(original.get(prToThe(pr, i) - 1));

          }
          return newArr;
        }

        private static ArrayList<Integer> matrixEncrypt2 (ArrayList <Integer> original, int pr) {
          ArrayList<Integer> newArr = new ArrayList<Integer>();
          for (int i = 0; i < original.size(); i++) {
                  newArr.add(original.get(prToThe(pr, i) - 1));
          }
```

```
            return newArr;
          }


          private static ArrayList<BufferedImage> funnyMosaicEncrypt (ArrayList
<BufferedImage> original, int a) {
                    //a is relatively prime to safePrime - 1
            ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
            for (int i = 0; i < original.size(); i++) {
                      newArr.add(original.get(prToThe(i+1, a) - 1));

            }
            return newArr;
          }

          private static ArrayList<Integer> funnyMatrixEncrypt (ArrayList <Integer> original, int a) {
                    //a is relatively prime to safePrime - 1
            ArrayList<Integer> newArr = new ArrayList<Integer>();
            for (int i = 0; i < original.size(); i++) {
                      newArr.add(original.get(prToThe(i+1, a) - 1));

            }
            return newArr;
          }

          private static ArrayList<BufferedImage> polyEncrypt (ArrayList <BufferedImage>
original, ArrayList<Integer> poly) {
            ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
            for (int i = 0; i < original.size(); i++) {
                      int desiredIndex = polyResult(poly, i, safePrime);
                      newArr.add(original.get(desiredIndex));

            }
            return newArr;
          }

          private static ArrayList<BufferedImage> polyDecrypt (ArrayList <BufferedImage>
original, ArrayList<Integer> poly) {
                      ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
                      for (int i = 0; i < original.size(); i++) {
                                int desiredIndex = inversePolyResult(poly, i, safePrime);
                                newArr.add(original.get(desiredIndex));

                      }
```

```
                return newArr;
              }

        //use pr1
        public static ArrayList<BufferedImage> mosaicDecryptRows (ArrayList <BufferedImage>
original, int pr) {
                ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
                for (int i = 0; i < original.size(); i++) {

                        newArr.add(original.get(discreteLogBasePrModsafePrime_(pr, i+1) %
(safePrime - 1)));
                }
                return newArr;
        }

        //use pr2
        public static ArrayList<BufferedImage> mosaicDecryptCols (ArrayList <BufferedImage>
original, int pr) {
                ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
                for (int i = 0; i < original.size(); i++) {
                        newArr.add(original.get(discreteLogBasePrModsafePrime_(pr, i+1) %
(safePrime - 1)));
                }
                return newArr;
        }

        private static ArrayList<BufferedImage> funnyMosaicDecrypt (ArrayList
<BufferedImage> original, int a) {
                //a is relatively prime to safePrime - 1
                ArrayList<BufferedImage> newArr = new ArrayList<BufferedImage>();
                 for (int i = 0; i < original.size() - 1; i++) {
                        newArr.add(original.get(computeNthRootModSafePrime(a, i+1) - 1));
                 }
                 newArr.add(original.get(safePrime - 2));
                 return newArr;
        }

        public static int computeNthRootModSafePrime (int n, int origNum) {
                int result = -1;
                for (int i = 0; i < safePrime; i++) {
                        if (prToThe(i, n) == origNum) {
                                result = i;
                                break;
                        }
                }
                return result;

        }


        private static BufferedImage mosaicScramble (BufferedImage original, int pr) throws
IOException {
                BufferedImage scrambled = original;
                Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);
                //scramble each row
                for (int y = 0; y < safePrime - 1; y++) {
                         ArrayList<BufferedImage> origRow = new ArrayList<BufferedImage>();

                        for (int j = 0; j < safePrime - 1; j++) {
                                BufferedImage c = m.getPixel(j, y);
                                origRow.add(c);
                        }

                        for (BufferedImage bi : origRow) {
                        }

                        //System.out.println();

                        //line #1
                        ArrayList<BufferedImage> scrambledRow = mosaicEncrypt (origRow,
pr);

                        //this for loop used for testing - outputs the correct and
                        //expected values for scrambledRow
                        for (int k = 0; k < safePrime - 1; k++) {
                        }

                        for (int j = 0; j < safePrime - 1; j++) {
                                BufferedImage newImage = scrambledRow.get(j);

                                //this line used for testing - somehow outputs  different values for
scrambledRow
                                //despite it being the same code to loop through the arraylist of
size safeprime - 1
                                //computeAverage function does NOT change either newImage
or scrambledRow
                                //nor did I ever change the scrambledRow arraylist in between
this loop and line #1
```

```
                        //does .get somehow change the arraylist values?

                        m.setPixel(j, y, newImage);

                    }



                }
                scrambled = m.getMosaic();
                return scrambled;

            }



        private static BufferedImage mosaicScramble2 (BufferedImage original, int pr) throws
IOException {
            BufferedImage scrambled = original;

            Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);
            //scramble each column
            for (int x = 0; x < safePrime - 1; x++) {
                ArrayList<BufferedImage> origCol = new ArrayList<BufferedImage>();
                for (int j = 0; j < safePrime - 1; j++) {
                    BufferedImage c = m.getPixel(x, j);
                    origCol.add(c);
                }

                ArrayList<BufferedImage> scrambledCol = mosaicEncrypt2 (origCol, pr);
                for (int j = 0; j < safePrime - 1; j++) {
                    BufferedImage newImage = scrambledCol.get(j);
                    m.setPixel(x, j, newImage);
                }



            }
            scrambled = m.getMosaic();
            return scrambled;
        }

        private static BufferedImage mosaicScrambleD1 (BufferedImage original, int pr1, int pr2)
throws IOException {
            BufferedImage scrambled = original;

            Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);

            //scramble each row
            for (int y = 0; y < safePrime - 1; y++) {
                ArrayList<BufferedImage> origRow = new ArrayList<BufferedImage>();
                for (int j = 0; j < safePrime - 1; j++) {
                    int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
                    BufferedImage c = m.getPixel(j, modifiedY);
                    origRow.add(c);
                }

                ArrayList<BufferedImage> scrambledRow = mosaicEncrypt (origRow, pr2);
                for (int j = 0; j < safePrime - 1; j++) {
                    int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
                    BufferedImage newImage = scrambledRow.get(j);

                    m.setPixel(j, modifiedY, newImage);

                }



            }
            scrambled = m.getMosaic();
            return scrambled;
        }

        private static BufferedImage mosaicScrambleD2 (BufferedImage original, int pr1, int pr2)
throws IOException{
                BufferedImage scrambled = original;
                Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);

                //scramble each column
                for (int x = 0; x < safePrime - 1; x++) {
                    ArrayList<BufferedImage> origCol = new ArrayList<BufferedImage>();
                    for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
                        BufferedImage c = m.getPixel(modifiedX, j);
                        origCol.add(c);
                    }
                    ArrayList<BufferedImage> scrambledCol = mosaicEncrypt2 (origCol,
pr2);
                    for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
                        BufferedImage newImage = scrambledCol.get(j);
                        m.setPixel(modifiedX, j, newImage);
                    }



                }
                scrambled = m.getMosaic();
                return scrambled;
        }



        private static boolean independent (int a1, int a2, int b1, int b2, int modulus) {
                int quotient1 = -1;
                int quotient2 = -1;

                boolean status = true;
                for (int k = 0; k < modulus; k++) {
                    if ((a1 * k) % modulus == b1) {
                        quotient1 = k;
                    }
                }
                for (int k = 0; k < modulus; k++) {
                    if ((a2 * k) % modulus == b2) {
                        quotient2 = k;
                    }
                }

                if ((quotient1 == quotient2) && (quotient1 != -1)) {
                    status = false;
                }

                return status;
        }
```

```
	private static BufferedImage funnyMosaicScramble (BufferedImage original, int pr)
throws IOException {
			BufferedImage scrambled = original;
			Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);
			//scramble each row
			for (int y = 0; y < safePrime - 1; y++) {
				 ArrayList<BufferedImage> origRow = new ArrayList<BufferedImage>();

				for (int j = 0; j < safePrime - 1; j++) {
					BufferedImage c = m.getPixel(j, y);
					origRow.add(c);
				}

				for (BufferedImage bi : origRow) {
				}

				System.out.println();

				//line #1
				ArrayList<BufferedImage> scrambledRow = funnyMosaicEncrypt
(origRow, pr);

				//this for loop used for testing - outputs the correct and
				//expected values for scrambledRow
				for (int k = 0; k < safePrime - 1; k++) {
				}

				for (int j = 0; j < safePrime - 1; j++) {
					BufferedImage newImage = scrambledRow.get(j);

					//this line used for testing - somehow outputs  different values for
scrambledRow
					//despite it being the same code to loop through the arraylist of
size safeprime - 1
					//computeAverage function does NOT change either newImage
or scrambledRow
					//nor did I ever change the scrambledRow arraylist in between
this loop and line #1
					//does .get somehow change the arraylist values?

					m.setPixel(j, y, newImage);

				}
```

```
                }
                scrambled = m.getMosaic();
                return scrambled;

        }

        private static BufferedImage funnyMosaicScramble2 (BufferedImage original, int pr)
throws IOException {
                BufferedImage scrambled = original;

                Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);
                //scramble each column
                for (int x = 0; x < safePrime - 1; x++) {
                        ArrayList<BufferedImage> origCol = new ArrayList<BufferedImage>();
                        for (int j = 0; j < safePrime - 1; j++) {
                                BufferedImage c = m.getPixel(x, j);
                                origCol.add(c);
                        }

                        ArrayList<BufferedImage> scrambledCol = funnyMosaicEncrypt
(origCol, pr);
                        for (int j = 0; j < safePrime - 1; j++) {
                                BufferedImage newImage = scrambledCol.get(j);
                                m.setPixel(x, j, newImage);
                        }



                }
                scrambled = m.getMosaic();
                return scrambled;
               }



        private static BufferedImage linearScramble (BufferedImage original, int a1, int a2, int
b1, int b2, int k1, int k2, int sp) throws IOException {
               BufferedImage scrambled = original;
               Mosaic4 m = new Mosaic4 (sp, pixelSize*2, scrambled);
               ArrayList<BufferedImage> origPics = new ArrayList<BufferedImage>();
               for (int x = 0; x < sp; x++) {
                       for (int y = 0; y < sp; y++) {
                        origPics.add(m.getPixel(x,  y));
                    }
                }
                for (int x = 0; x < sp; x++) {
                    for (int y = 0; y < sp; y++) {
                        int [] mapResult = affineMapping (a1, a2, b1, b2, k1, k2, sp, x, y);
                        int desiredIndex = (sp)*mapResult[0] + mapResult[1];
                        m.setPixel(x, y, origPics.get(desiredIndex));
                    }
                }

                scrambled = m.getMosaic();
                return scrambled;


            }


            private static int[] affineMapping (int a1, int a2, int b1, int b2, int k1, int k2, int modulus, int
x, int y) {
                int component1 = (a1*x+b1*y+k1) % modulus;
                int component2 = (a2*x+b2*y+k2) % modulus;
                int [] resultArray = {component1, component2};
                return resultArray;
            }

            private static BufferedImage funnyMosaicScrambleD1 (BufferedImage original, int pr1,
int pr2) throws IOException {
                BufferedImage scrambled = original;

                Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);

                //scramble each row
                for (int y = 0; y < safePrime - 1; y++) {
                    ArrayList<BufferedImage> origRow = new ArrayList<BufferedImage>();
                    for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedY = (y + (prToThe(j+1, pr1))) % (safePrime - 1);
                        BufferedImage c = m.getPixel(j, modifiedY);
                        origRow.add(c);
                    }

                    ArrayList<BufferedImage> scrambledRow = funnyMosaicEncrypt
(origRow, pr2);
                    for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedY = (y + (prToThe(j+1, pr1))) % (safePrime - 1);
                        BufferedImage newImage = scrambledRow.get(j);

                        m.setPixel(j, modifiedY, newImage);

                    }



                }
                scrambled = m.getMosaic();
                return scrambled;
            }

        private static BufferedImage funnyMosaicScrambleD2 (BufferedImage original, int pr1,
int pr2) throws IOException{
                BufferedImage scrambled = original;
                Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, scrambled);

                //scramble each column
                for (int x = 0; x < safePrime - 1; x++) {
                    ArrayList<BufferedImage> origCol = new ArrayList<BufferedImage>();
                    for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedX = (x + (prToThe(j+1, pr1))) % (safePrime - 1);
                        BufferedImage c = m.getPixel(modifiedX, j);
                        origCol.add(c);
                    }

                    ArrayList<BufferedImage> scrambledCol = funnyMosaicEncrypt
(origCol, pr2);
                    for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedX = (x + (prToThe(j+1, pr1))) % (safePrime - 1);
                        BufferedImage newImage = scrambledCol.get(j);
                        m.setPixel(modifiedX, j, newImage);
                    }



                }
                scrambled = m.getMosaic();
                return scrambled;
        }
```

```
public static BufferedImage mosaicUnscrambleRows (BufferedImage original, int pr) throws
IOException {
        BufferedImage unscrambled = original;
        Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

        //scramble each column
        for (int x = 0; x < safePrime - 1; x++) {
                ArrayList<BufferedImage> origCol = new ArrayList <BufferedImage> ();
                for (int j = 0; j < safePrime - 1; j++) {
                        BufferedImage c = m.getPixel(x, j);
                        origCol.add(c);
                }

                ArrayList<BufferedImage> unscrambledCol = mosaicDecryptCols (origCol, pr);
                for (int j = 0; j < safePrime - 1; j++) {
                        BufferedImage newImage = unscrambledCol.get(j);
                        m.setPixel(x, j, newImage);
                }
                System.out.println("Row " + x + " unscrambled");



        }
        unscrambled = m.getMosaic();
        return unscrambled;
       }

        public static BufferedImage mosaicUnscrambleCols (BufferedImage original, int pr)
throws IOException {
                BufferedImage unscrambled = original;
                Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

                //scramble each row
                for (int y = 0; y < safePrime - 1; y++) {
                        ArrayList<BufferedImage> origRow = new ArrayList <BufferedImage> ();
                        for (int j = 0; j < safePrime - 1; j++) {
                                BufferedImage c = m.getPixel(j, y);
                                origRow.add(c);
                        }

                        ArrayList<BufferedImage> unscrambledRow = mosaicDecryptRows
(origRow, pr);
                        for (int j = 0; j < safePrime - 1; j++) {
                                BufferedImage newImage = unscrambledRow.get(j);
```

```
                    m.setPixel(j, y, newImage);
                }
                System.out.println("Column " + y + " unscrambled");




            }
            unscrambled = m.getMosaic();
            return unscrambled;
        }

    public static BufferedImage mosaicUnscrambleD1 (BufferedImage original, int pr1, int
pr2) throws IOException {
            BufferedImage unscrambled = original;
            Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

            //scramble each row
            for (int y = 0; y < safePrime - 1; y++) {
                ArrayList<BufferedImage> origRow = new ArrayList <BufferedImage> ();
                for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
                    BufferedImage c = m.getPixel(j, modifiedY);
                    origRow.add(c);
                }

                ArrayList<BufferedImage> unscrambledRow = mosaicDecryptRows
(origRow, pr2);
                for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
                    BufferedImage newImage = unscrambledRow.get(j);
                    m.setPixel(j, modifiedY, newImage);
                }
                System.out.println("Column " + y + " unscrambled");



            }
            unscrambled = m.getMosaic();
            return unscrambled;
        }

    public static BufferedImage mosaicUnscrambleD2 (BufferedImage original, int pr1, int
pr2) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

		//scramble each column
		for (int x = 0; x < safePrime - 1; x++) {
			ArrayList<BufferedImage> origCol = new ArrayList <BufferedImage> ();
			for (int j = 0; j < safePrime - 1; j++) {
				int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
				BufferedImage c = m.getPixel(modifiedX, j);
				origCol.add(c);
			}

			ArrayList<BufferedImage> unscrambledCol = mosaicDecryptCols
(origCol, pr2);
			for (int j = 0; j < safePrime - 1; j++) {
				int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
				BufferedImage newImage = unscrambledCol.get(j);
				m.setPixel(modifiedX, j, newImage);
			}
			System.out.println("Row " + x + " unscrambled");



		}
		unscrambled = m.getMosaic();
		return unscrambled;
	}

	public static BufferedImage funnyMosaicUnscrambleRows (BufferedImage original, int
pr) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

		//scramble each column
		for (int x = 0; x < safePrime - 1; x++) {
			ArrayList<BufferedImage> origCol = new ArrayList <BufferedImage> ();
			for (int j = 0; j < safePrime - 1; j++) {
				BufferedImage c = m.getPixel(x, j);
				origCol.add(c);
			}

			ArrayList<BufferedImage> unscrambledCol = funnyMosaicDecrypt
(origCol, pr);
			for (int j = 0; j < safePrime - 1; j++) {
```

```
                BufferedImage newImage = unscrambledCol.get(j);
                m.setPixel(x, j, newImage);
            }
            System.out.println("Row " + x + " unscrambled");



        }
        unscrambled = m.getMosaic();
        return unscrambled;
       }

    public static BufferedImage funnyMosaicUnscrambleCols (BufferedImage original, int pr)
throws IOException {
        BufferedImage unscrambled = original;
        Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

        //scramble each row
        for (int y = 0; y < safePrime - 1; y++) {
            ArrayList<BufferedImage> origRow = new ArrayList <BufferedImage> ();
            for (int j = 0; j < safePrime - 1; j++) {
                BufferedImage c = m.getPixel(j, y);
                origRow.add(c);
            }

            ArrayList<BufferedImage> unscrambledRow = funnyMosaicDecrypt
(origRow, pr);
            for (int j = 0; j < safePrime - 1; j++) {
                BufferedImage newImage = unscrambledRow.get(j);
                m.setPixel(j, y, newImage);
            }
            System.out.println("Column " + y + " unscrambled");




        }
        unscrambled = m.getMosaic();
        return unscrambled;
     }

    public static BufferedImage funnyMosaicUnscrambleD1 (BufferedImage original, int pr1,
int pr2) throws IOException {
        BufferedImage unscrambled = original;
```

```
		Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

		//scramble each row
		for (int y = 0; y < safePrime - 1; y++) {
			ArrayList<BufferedImage> origRow = new ArrayList <BufferedImage> ();
			for (int j = 0; j < safePrime - 1; j++) {
					int modifiedY = (y + (prToThe(j+1, pr1))) % (safePrime -
1);
				BufferedImage c = m.getPixel(j, modifiedY);
				origRow.add(c);
			}

			ArrayList<BufferedImage> unscrambledRow = funnyMosaicDecrypt
(origRow, pr2);
			for (int j = 0; j < safePrime - 1; j++) {
					int modifiedY = (y + (prToThe(j+1, pr1))) % (safePrime -
1);
				BufferedImage newImage = unscrambledRow.get(j);
				m.setPixel(j, modifiedY, newImage);
			}
			System.out.println("Column " + y + " unscrambled");


		}
		unscrambled = m.getMosaic();
		return unscrambled;
	 }

	public static BufferedImage funnyMosaicUnscrambleD2 (BufferedImage original, int pr1,
int pr2) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic4 m = new Mosaic4 (safePrime - 1, pixelSize, unscrambled);

		//scramble each column
		for (int x = 0; x < safePrime - 1; x++) {
			ArrayList<BufferedImage> origCol = new ArrayList <BufferedImage> ();
			for (int j = 0; j < safePrime - 1; j++) {
				int modifiedX = (x + (prToThe(j+1, pr1))) % (safePrime - 1);
				BufferedImage c = m.getPixel(modifiedX, j);
				origCol.add(c);
			}

			ArrayList<BufferedImage> unscrambledCol = funnyMosaicDecrypt
(origCol, pr2);
```

```
                    for (int j = 0; j < safePrime - 1; j++) {
                            int modifiedX = (x + (prToThe(j+1, pr1))) % (safePrime - 1);
                            BufferedImage newImage = unscrambledCol.get(j);
                            m.setPixel(modifiedX, j, newImage);
                    }
                    System.out.println("Row " + x + " unscrambled");



            }
            unscrambled = m.getMosaic();
            return unscrambled;
    }


    private static BufferedImage polyMosaicScramble (BufferedImage original,
ArrayList<Integer> pr) throws IOException {
            BufferedImage scrambled = original;
            Mosaic4 m = new Mosaic4 (safePrime, pixelSize, scrambled);
            //scramble each row
            for (int y = 0; y < safePrime; y++) {
                     ArrayList<BufferedImage> origRow = new ArrayList<BufferedImage>();

                    for (int j = 0; j < safePrime; j++) {
                            BufferedImage c = m.getPixel(j, y);
                            origRow.add(c);
                    }


                    ArrayList<BufferedImage> scrambledRow = polyEncrypt (origRow, pr);



                    for (int j = 0; j < safePrime; j++) {
                            BufferedImage newImage = scrambledRow.get(j);



                            m.setPixel(j, y, newImage);

                    }
```

```
		}
		scrambled = m.getMosaic();
		return scrambled;

	 }

	private static BufferedImage polyMosaicScramble2 (BufferedImage original,
ArrayList<Integer> pr) throws IOException {
		BufferedImage scrambled = original;

		Mosaic4 m = new Mosaic4 (safePrime, pixelSize, scrambled);
		//scramble each column
		for (int x = 0; x < safePrime; x++) {
			ArrayList<BufferedImage> origCol = new ArrayList<BufferedImage>();
			for (int j = 0; j < safePrime; j++) {
				BufferedImage c = m.getPixel(x, j);
				origCol.add(c);
			}

			ArrayList<BufferedImage> scrambledCol = polyEncrypt (origCol, pr);
			for (int j = 0; j < safePrime; j++) {
				BufferedImage newImage = scrambledCol.get(j);
				m.setPixel(x, j, newImage);
			}



		}
		scrambled = m.getMosaic();
		return scrambled;
	       }

	private static BufferedImage polyMosaicScrambleD1 (BufferedImage original,
ArrayList<Integer> pr1, ArrayList<Integer> pr2) throws IOException {
		BufferedImage scrambled = original;

		Mosaic4 m = new Mosaic4 (safePrime, pixelSize, scrambled);

		//scramble each row
		for (int y = 0; y < safePrime; y++) {
			ArrayList<BufferedImage> origRow = new ArrayList<BufferedImage>();
			for (int j = 0; j < safePrime; j++) {

				int modifiedY = (y + polyResult(pr1, j, safePrime)) % safePrime;
```

```
                BufferedImage c = m.getPixel(j, modifiedY);
                origRow.add(c);
            }

            ArrayList<BufferedImage> scrambledRow = polyEncrypt (origRow, pr2);
            for (int j = 0; j < safePrime; j++) {
                int modifiedY = (y + polyResult(pr1, j, safePrime)) % safePrime;

                BufferedImage newImage = scrambledRow.get(j);

                m.setPixel(j, modifiedY, newImage);

            }



        }
        scrambled = m.getMosaic();
        return scrambled;
    }

    private static BufferedImage polyMosaicScrambleD2 (BufferedImage original,
ArrayList<Integer> pr1, ArrayList<Integer> pr2) throws IOException{
        BufferedImage scrambled = original;
        Mosaic4 m = new Mosaic4 (safePrime, pixelSize, scrambled);

        //scramble each column
        for (int x = 0; x < safePrime; x++) {
            ArrayList<BufferedImage> origCol = new ArrayList<BufferedImage>();
            for (int j = 0; j < safePrime; j++) {
                int modifiedX = (x + polyResult(pr1, j, safePrime)) % safePrime;
                BufferedImage c = m.getPixel(modifiedX, j);
                origCol.add(c);
            }

            ArrayList<BufferedImage> scrambledCol = polyEncrypt (origCol, pr2);
            for (int j = 0; j < safePrime; j++) {
                int modifiedX = (x + polyResult(pr1, j, safePrime)) % safePrime;
                BufferedImage newImage = scrambledCol.get(j);
                m.setPixel(modifiedX, j, newImage);
            }
```

```
		}
		scrambled = m.getMosaic();
		return scrambled;
	}

	public static BufferedImage polyMosaicUnscrambleRows (BufferedImage original,
ArrayList<Integer> pr) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic4 m = new Mosaic4 (safePrime, pixelSize, unscrambled);

		//scramble each column
		for (int x = 0; x < safePrime; x++) {
			ArrayList<BufferedImage> origCol = new ArrayList <BufferedImage> ();
			for (int j = 0; j < safePrime; j++) {
				BufferedImage c = m.getPixel(x, j);
				origCol.add(c);
			}

			ArrayList<BufferedImage> unscrambledCol = polyDecrypt (origCol, pr);
			for (int j = 0; j < safePrime; j++) {
				BufferedImage newImage = unscrambledCol.get(j);
				m.setPixel(x, j, newImage);
			}
			System.out.println("Row " + x + " unscrambled");



		}
		unscrambled = m.getMosaic();
		return unscrambled;
	       }

	public static BufferedImage polyMosaicUnscrambleCols (BufferedImage original,
ArrayList<Integer> pr) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic4 m = new Mosaic4 (safePrime, pixelSize, unscrambled);

		//scramble each row
		for (int y = 0; y < safePrime; y++) {
			ArrayList<BufferedImage> origRow = new ArrayList <BufferedImage> ();
			for (int j = 0; j < safePrime; j++) {
				BufferedImage c = m.getPixel(j, y);
				origRow.add(c);
			}
```

```
                ArrayList<BufferedImage> unscrambledRow = polyDecrypt (origRow,
pr);
                for (int j = 0; j < safePrime; j++) {
                    BufferedImage newImage = unscrambledRow.get(j);
                    m.setPixel(j, y, newImage);
                }
                System.out.println("Column " + y + " unscrambled");




            }
            unscrambled = m.getMosaic();
            return unscrambled;
        }

    public static BufferedImage polyMosaicUnscrambleD1 (BufferedImage original,
ArrayList<Integer> pr1, ArrayList<Integer> pr2) throws IOException {
            BufferedImage unscrambled = original;
            Mosaic4 m = new Mosaic4 (safePrime, pixelSize, unscrambled);

            //scramble each row
            for (int y = 0; y < safePrime; y++) {
                ArrayList<BufferedImage> origRow = new ArrayList <BufferedImage> ();
                for (int j = 0; j < safePrime; j++) {
                        int modifiedY = (y + polyResult(pr1, j, safePrime)) %
safePrime;
                    BufferedImage c = m.getPixel(j, modifiedY);
                    origRow.add(c);
                }

                ArrayList<BufferedImage> unscrambledRow = polyDecrypt (origRow,
pr2);
                for (int j = 0; j < safePrime; j++) {
                        int modifiedY = (y + polyResult(pr1, j, safePrime)) %
safePrime;
                    BufferedImage newImage = unscrambledRow.get(j);
                    m.setPixel(j, modifiedY, newImage);
                }
                System.out.println("Column " + y + " unscrambled");


            }
```

```
		unscrambled = m.getMosaic();
		return unscrambled;
	 }

	public static BufferedImage polyMosaicUnscrambleD2 (BufferedImage original,
ArrayList<Integer> pr1, ArrayList<Integer> pr2) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic4 m = new Mosaic4 (safePrime, pixelSize, unscrambled);

		//scramble each column
		for (int x = 0; x < safePrime; x++) {
			ArrayList<BufferedImage> origCol = new ArrayList <BufferedImage> ();
			for (int j = 0; j < safePrime; j++) {
				int modifiedX = (x + polyResult(pr1, j, safePrime)) % safePrime;
				BufferedImage c = m.getPixel(modifiedX, j);
				origCol.add(c);
			}

			ArrayList<BufferedImage> unscrambledCol = polyDecrypt (origCol, pr2);
			for (int j = 0; j < safePrime; j++) {
				int modifiedX = (x + polyResult(pr1, j, safePrime)) % safePrime;
				BufferedImage newImage = unscrambledCol.get(j);
				m.setPixel(modifiedX, j, newImage);
			}
			System.out.println("Row " + x + " unscrambled");



		}
		unscrambled = m.getMosaic();
		return unscrambled;
	}



	//--------------------------------------


	private static ArrayList<Integer> matrixScramble (ArrayList<Integer> original, int pr)
throws IOException {
		ArrayList<Integer> scrambled = original;
		IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);
		//scramble each row
		for (int y = 0; y < safePrime - 1; y++) {
```

```
                         ArrayList<Integer> origRow = new ArrayList<Integer>();

                        for (int j = 0; j < safePrime - 1; j++) {
                                int c = m.getPixel(j, y);
                                origRow.add(c);
                        }

                        for (Integer bi : origRow) {
                        }

                        //System.out.println();

                        //line #1
                        ArrayList<Integer> scrambledRow = matrixEncrypt (origRow, pr);

                        //this for loop used for testing - outputs the correct and
                        //expected values for scrambledRow
                        for (int k = 0; k < safePrime - 1; k++) {
                        }

                        for (int j = 0; j < safePrime - 1; j++) {
                                int newImage = scrambledRow.get(j);

                                //this line used for testing - somehow outputs  different values for
scrambledRow
                                //despite it being the same code to loop through the arraylist of
size safeprime - 1
                                //computeAverage function does NOT change either newImage
or scrambledRow
                                //nor did I ever change the scrambledRow arraylist in between
this loop and line #1
                                //does .get somehow change the arraylist values?

                                m.setPixel(j, y, newImage);

                        }



                }
                scrambled = m.getMosaic();
                return scrambled;

        }
```

```
	private static ArrayList<Integer> matrixScramble2 (ArrayList<Integer> original, int pr)
throws IOException {
		ArrayList<Integer> scrambled = original;

	 IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);
	 //scramble each column
	 for (int x = 0; x < safePrime - 1; x++) {
		ArrayList<Integer> origCol = new ArrayList<Integer>();
		for (int j = 0; j < safePrime - 1; j++) {
			int c = m.getPixel(x, j);
			origCol.add(c);
		}

		ArrayList<Integer> scrambledCol = matrixEncrypt2 (origCol, pr);
		for (int j = 0; j < safePrime - 1; j++) {
			int newImage = scrambledCol.get(j);
			m.setPixel(x, j, newImage);
		}



	 }
	 scrambled = m.getMosaic();
	 return scrambled;
	}

	private static ArrayList<Integer> matrixScrambleD1 (ArrayList<Integer> original, int pr1,
int pr2) throws IOException {
	 ArrayList<Integer> scrambled = original;

	 IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);

	 //scramble each row
	 for (int y = 0; y < safePrime - 1; y++) {
		ArrayList<Integer> origRow = new ArrayList<Integer>();
		for (int j = 0; j < safePrime - 1; j++) {
			int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
			int c = m.getPixel(j, modifiedY);
			origRow.add(c);
		}

		ArrayList<Integer> scrambledRow = matrixEncrypt (origRow, pr2);
```

```
                for (int j = 0; j < safePrime - 1; j++) {
                        int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
                        int newImage = scrambledRow.get(j);

                        m.setPixel(j, modifiedY, newImage);

                }



        }
        scrambled = m.getMosaic();
        return scrambled;
  }

        private static ArrayList<Integer> matrixScrambleD2 (ArrayList<Integer> original, int pr1,
int pr2) throws IOException{
                ArrayList<Integer> scrambled = original;
                IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);

                //scramble each column
                for (int x = 0; x < safePrime - 1; x++) {
                        ArrayList<Integer> origCol = new ArrayList<Integer>();
                        for (int j = 0; j < safePrime - 1; j++) {
                                int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
                                int c = m.getPixel(modifiedX, j);
                                origCol.add(c);
                        }

                        ArrayList<Integer> scrambledCol = matrixEncrypt2 (origCol, pr2);
                        for (int j = 0; j < safePrime - 1; j++) {
                                int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
                                int newImage = scrambledCol.get(j);
                                m.setPixel(modifiedX, j, newImage);
                        }



                }
                scrambled = m.getMosaic();
                return scrambled;
        }
```

```
	private static boolean matrixIndependent (int a1, int a2, int b1, int b2, int modulus) {
		int quotient1 = -1;
		int quotient2 = -1;

		boolean status = true;
		for (int k = 0; k < modulus; k++) {
			if ((a1 * k) % modulus == b1) {
				quotient1 = k;
			}
		}
		for (int k = 0; k < modulus; k++) {
			if ((a2 * k) % modulus == b2) {
				quotient2 = k;
			}
		}

		if ((quotient1 == quotient2) && (quotient1 != -1)) {
			status = false;
		}

		return status;
	}




	private static ArrayList<Integer> funnyMatrixScramble (ArrayList<Integer> original, int pr)
throws IOException {
		 ArrayList<Integer> scrambled = original;
		 IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);
		 //scramble each row
		 for (int y = 0; y < safePrime - 1; y++) {
			  ArrayList<Integer> origRow = new ArrayList<Integer>();

			 for (int j = 0; j < safePrime - 1; j++) {
				 int c = m.getPixel(j, y);
				 origRow.add(c);
			 }




			 //line #1
```

```
            ArrayList<Integer> scrambledRow = funnyMatrixEncrypt (origRow, pr);

            //this for loop used for testing - outputs the correct and
            //expected values for scrambledRow
            for (int k = 0; k < safePrime - 1; k++) {
            }

            for (int j = 0; j < safePrime - 1; j++) {
                int newImage = scrambledRow.get(j);

                //this line used for testing - somehow outputs  different values for
scrambledRow
                //despite it being the same code to loop through the arraylist of
size safeprime - 1
                //computeAverage function does NOT change either newImage
or scrambledRow
                //nor did I ever change the scrambledRow arraylist in between
this loop and line #1
                //does .get somehow change the arraylist values?

                m.setPixel(j, y, newImage);

            }



        }
        scrambled = m.getMosaic();
        return scrambled;

    }

    private static ArrayList<Integer> funnyMatrixScramble2 (ArrayList<Integer> original, int
pr) throws IOException {
        ArrayList<Integer> scrambled = original;

        IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);
        //scramble each column
        for (int x = 0; x < safePrime - 1; x++) {
            ArrayList<Integer> origCol = new ArrayList<Integer>();
            for (int j = 0; j < safePrime - 1; j++) {
                int c = m.getPixel(x, j);
                origCol.add(c);
            }
```

```
                    ArrayList<Integer> scrambledCol = funnyMatrixEncrypt (origCol, pr);
                    for (int j = 0; j < safePrime - 1; j++) {
                            int newImage = scrambledCol.get(j);
                            m.setPixel(x, j, newImage);
                    }



            }
            scrambled = m.getMosaic();
            return scrambled;
           }



        private static ArrayList<Integer> linearMatrixScramble (ArrayList<Integer> original, int
a1, int a2, int b1, int b2, int k1, int k2, int sp) throws IOException {
                ArrayList<Integer> scrambled = original;
                IntegerMosaic m = new IntegerMosaic (sp, pixelSize*2, scrambled);
                ArrayList<Integer> origPics = new ArrayList<Integer>();
                for (int x = 0; x < sp; x++) {
                        for (int y = 0; y < sp; y++) {
                                origPics.add(m.getPixel(x,  y));
                        }
                }
                for (int x = 0; x < sp; x++) {
                        for (int y = 0; y < sp; y++) {
                                int [] mapResult = affineMapping (a1, a2, b1, b2, k1, k2, sp, x, y);
                                int desiredIndex = (sp)*mapResult[0] + mapResult[1];
                                m.setPixel(x, y, origPics.get(desiredIndex));
                        }
                }

                scrambled = m.getMosaic();
                return scrambled;


        }


        private static int[] affineMatrixMapping (int a1, int a2, int b1, int b2, int k1, int k2, int
modulus, int x, int y) {
                int component1 = (a1*x+b1*y+k1) % modulus;
                int component2 = (a2*x+b2*y+k2) % modulus;
                int [] resultArray = {component1, component2};
                return resultArray;
        }



        private static ArrayList<Integer> funnyMatrixScrambleD1 (ArrayList<Integer> original, int
pr1, int pr2) throws IOException {
                ArrayList<Integer> scrambled = original;

                IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);

                //scramble each row
                for (int y = 0; y < safePrime - 1; y++) {
                        ArrayList<Integer> origRow = new ArrayList<Integer>();
                        for (int j = 0; j < safePrime - 1; j++) {
                                int modifiedY = (y + (prToThe(j+1, pr1))) % (safePrime - 1);
                                int c = m.getPixel(j, modifiedY);
                                origRow.add(c);
                        }

                        ArrayList<Integer> scrambledRow = funnyMatrixEncrypt (origRow, pr2);
                        for (int j = 0; j < safePrime - 1; j++) {
                                int modifiedY = (y + (prToThe(j+1, pr1))) % (safePrime - 1);
                                int newImage = scrambledRow.get(j);

                                m.setPixel(j, modifiedY, newImage);

                        }



                }
                scrambled = m.getMosaic();
                return scrambled;
          }

        private static ArrayList<Integer> funnyMatrixScrambleD2 (ArrayList<Integer> original, int
pr1, int pr2) throws IOException{
                ArrayList<Integer> scrambled = original;
                IntegerMosaic m = new IntegerMosaic (safePrime - 1, pixelSize, scrambled);

                //scramble each column
```

```
            for (int x = 0; x < safePrime - 1; x++) {
                    ArrayList<Integer> origCol = new ArrayList<Integer>();
                    for (int j = 0; j < safePrime - 1; j++) {
                            int modifiedX = (x + (prToThe(j+1, pr1))) % (safePrime - 1);
                            int c = m.getPixel(modifiedX, j);
                            origCol.add(c);
                    }

                    ArrayList<Integer> scrambledCol = funnyMatrixEncrypt (origCol, pr2);
                    for (int j = 0; j < safePrime - 1; j++) {
                            int modifiedX = (x + (prToThe(j+1, pr1))) % (safePrime - 1);
                            int newImage = scrambledCol.get(j);
                            m.setPixel(modifiedX, j, newImage);
                    }



            }
            scrambled = m.getMosaic();
            return scrambled;
    }



    //more helper functions





    public static ArrayList<ArrayList<Integer>> getPolynomialBasis () {
           ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
            ArrayList <Integer> relPrimes = new ArrayList <Integer> ();
            for (int i = 0; i < (safePrime - 1); i++) {
                    if (((i % 2) == 1) && (i != (safePrime - 1) / 2)) {
                            relPrimes.add(i);
                    }
            }

            for (int b1 = 1; b1 < safePrime; b1++) {
                    for (int b2 = 1; b2 < safePrime; b2++) {
                            for (int aIndex = 0; aIndex < relPrimes.size(); aIndex++) {
                                    int a = relPrimes.get(aIndex);
                                    ArrayList<Integer> thepoly = new ArrayList<Integer>();
```

```
                                        thepoly.add(b1);
                                        for (int i = 1; i < a; i++) {
                                                thepoly.add(0);
                                        }
                                        thepoly.add(b2);
                                        result.add(thepoly);
                                }
                        }
                }
                return result;

        }




        public static ArrayList<Integer> composition (ArrayList<Integer> p1, ArrayList<Integer>
p2, int modulus) {
                ArrayList<ArrayList<Integer>> terms = new ArrayList<ArrayList<Integer>>();
                for (int i = 0; i < p1.size(); i++) {
                        ArrayList<Integer> p1Coeff = new ArrayList<Integer>();
                        p1Coeff.add(p1.get(i));
                        terms.add(product(p1Coeff, power(p2, i, modulus), modulus));
                }

                /*for (int i = 0; i < terms.size(); i++) {
                        System.out.println(terms.get(i));
                }*/

                ArrayList<Integer> result = sum(terms, modulus);
                return result;
        }

        public static ArrayList<Integer> sum (ArrayList<ArrayList<Integer>> polys, int modulus) {

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

                int maxLength = 0;
                for (ArrayList<Integer> p : polys) {
                        if (p.size() > maxLength) {
                                maxLength = p.size();
                        }
                }
        for (int i = 0; i < maxLength; i++) {
            for (ArrayList<Integer> p : polys) {
                if (i >= p.size()) {
                    p.add(0);
                }
            }
        }

        for (int i = 0; i < maxLength; i++) {
            int coeff = 0;
            for (ArrayList<Integer> p : polys) {
                coeff += p.get(i) % modulus;
            }
            coeff = coeff % modulus;
            result.add(coeff);
        }

        return result;
    }



    public static ArrayList<Integer> product (ArrayList<Integer> p1, ArrayList<Integer> p2,
int modulus) {
        int[] newCoeffs = new int[p1.size() + p2.size() - 1];
        for (int i = 0; i < p1.size(); i++) {
            for (int j = 0; j < p2.size(); j++) {
                newCoeffs[i+j] += (p1.get(i) * p2.get(j)) % modulus;
            }
        }
        ArrayList<Integer> result = new ArrayList<Integer>();
        for (int i = 0; i < newCoeffs.length; i++) {
            result.add(newCoeffs[i] % modulus);
        }
        return result;
    }

    public static ArrayList<Integer> power (ArrayList<Integer> p, int exponent, int modulus) {
        ArrayList<Integer> baseCase = new ArrayList<Integer> ();
        baseCase.add(1);
        if (exponent > 0) {
            return product (p, power(p, exponent-1, modulus), modulus);
        }
        else {
```

```
				return baseCase;
			}
		}

		public static ArrayList<Integer> normalize (ArrayList<Integer> p) {
			ArrayList<Integer> result = new ArrayList<Integer>();
			int[] resultArray = new int [safePrime - 1];
			for (int i = 0; i < safePrime - 1; i++) {
				for (int j = i; j < p.size(); j += safePrime - 1) {
					resultArray[i] += p.get(j);
				}
				resultArray[i] %= safePrime;
			}

			for (int i = 0; i < resultArray.length; i++) {
				result.add(resultArray[i]);
			}
			return result;


		}

		public static ArrayList<ArrayList<Integer>> getAllPolys (int currDegree, int lBound, int
	uBound) {
			ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
			if (currDegree == -1) {
				result.add(new ArrayList<Integer>());
				return result;
			}
			for (int i = lBound; i <= uBound; i++) {
				result.addAll(appendAtEnd(getAllPolys(currDegree - 1, lBound, uBound),
	i));
			}
			return result;
		}

		public static boolean isBijective (ArrayList<Integer> poly, int modulus) {
			ArrayList<Integer> polyImage = new ArrayList<Integer>();
			ArrayList<Integer> match = new ArrayList<Integer>();
			for (int i = 0; i < modulus; i++) {
				match.add(i);
			}
			for (int i = 0; i < modulus; i++) {
				polyImage.add(polyResult(poly, i, modulus));
		}
		Collections.sort(polyImage);
		return polyImage.equals(match);
	}

	public static int polyResult (ArrayList<Integer> poly, int x, int modulus) {
		int total = 0;
		for (int exponent = 0; exponent < poly.size(); exponent++) {
			total += (poly.get(exponent) * (power(x, exponent, modulus)));
		}
		return (total % modulus);
	}

	//for bijective polynomials only
	public static int inversePolyResult (ArrayList<Integer> poly, int y, int modulus) {
		int result = 0;
		for (int x = 0; x < modulus; x++) {
			if (polyResult(poly, x, modulus) == y) {
				result = x;
				break;
			}
		}
		return result;
	}

	public static int power (int n, int k, int modulus) {
		int result = 1;
		for (int i = 0; i < k; i++) {
			result *= n;
			result %= modulus;
		}
		return result;
	}


	public static ArrayList<ArrayList<Integer>> appendAtEnd (ArrayList<ArrayList<Integer>>
input, int x) {
		ArrayList<ArrayList<Integer>> result = input;
		for (ArrayList<Integer> al : result) {
			al.add(x);
		}
		return result;

	}
```

```
}

class Mosaic4 {
                    private ArrayList<BufferedImage> mosaicImages = new
ArrayList<BufferedImage>();
                    private int dimension;
                    private int pixelSize;
                    private BufferedImage orig;
                    private BufferedImage origMosaic;
                    private BufferedImage mosaic;
                    public Mosaic4 (int spd, int ps, BufferedImage b) throws IOException {
                              dimension = spd;
                              pixelSize = ps;
                              orig = b;
                              fractionate();

                    }

                    private void fractionate () throws IOException {
                              mosaic = resizeImage(orig, dimension*pixelSize, dimension*pixelSize);
                              origMosaic = resizeImage(orig, dimension*pixelSize,
dimension*pixelSize);

                              for (int x = 0; x < dimension; x++) {
                     for (int y = 0; y < dimension; y++) {

                               //compute average RGB of cell
                               BufferedImage sub = origMosaic.getSubimage(x * pixelSize, y*pixelSize,
pixelSize, pixelSize);

                               mosaicImages.add(sub);



                     }
          }

                    }

                    public BufferedImage getMosaic () {
                              return mosaic;
                    }
                    public ArrayList<BufferedImage> getMosaicImages() {
                              return mosaicImages;
                    }

                    public void setPixel (int x, int y, BufferedImage bi) {

                              for (int i = x*pixelSize; i < (x+1)*pixelSize; i++) {
                              for (int j = y*pixelSize; j < (y+1)*pixelSize; j++) {
                                        int newRGB = bi.getRGB(i % pixelSize, j % pixelSize);
                                        mosaic.setRGB(i, j, newRGB);
                              }
                     }
                    }

                    public BufferedImage getPixel (int xCoord, int yCoord) {
                              BufferedImage returnedImage = mosaicImages.get(dimension * xCoord +
yCoord);
                              return returnedImage;
                    }

                    static BufferedImage resizeImage(BufferedImage originalImage, int targetWidth,
int targetHeight) throws IOException {
                               BufferedImage resizedImage = new BufferedImage(targetWidth,
targetHeight, BufferedImage.TYPE_INT_RGB);
                               Graphics2D graphics2D = resizedImage.createGraphics();
                               graphics2D.drawImage(originalImage, 0, 0, targetWidth, targetHeight,
null);
                               graphics2D.dispose();
                               return resizedImage;
                              }

                    private static Color computeAverage (BufferedImage bi) {
                    long rSum = 0;
                    long gSum = 0;
                    long bSum = 0;
                    for (int i = 0; i < bi.getWidth(); i++) {
                              for (int j = 0; j < bi.getHeight(); j++) {
                                        Color c = new Color (bi.getRGB(i, j));
                                        rSum += c.getRed();
                                        gSum += c.getGreen();
                                        bSum += c.getBlue();
                              }
                    }
```

```
                int totalPixels = bi.getWidth() * bi.getHeight();
                return new Color((int) (rSum / totalPixels), (int) (gSum / totalPixels), (int) (bSum /
totalPixels));
          }

        }


//--------------------------------------------------



class IntegerMosaic {
        private ArrayList<Integer> mosaicImages = new ArrayList<Integer>();
        private int dimension;
        private int pixelSize;
        private ArrayList<Integer> orig;
        private ArrayList<Integer> origMosaic;
        private ArrayList<Integer> mosaic = new ArrayList<Integer>();
        public IntegerMosaic (int spd, int ps, ArrayList<Integer> b) throws IOException {
                dimension = spd;
                pixelSize = ps;
                orig = b;
                fractionate();


        }

        private void fractionate () throws IOException {

                for (int x = 0; x < dimension; x++) {
                 for (int y = 0; y < dimension; y++) {

                     //compute average RGB of cell
                     int sub = orig.get((dimension*x + y));

                     mosaicImages.add(sub);

                     mosaic.add(sub);

                 }

         }
```

```
	}

	public ArrayList<Integer> getMosaic () {
		return mosaic;
	}

	public ArrayList<Integer> getMosaicImages() {
		return mosaicImages;
	}

	public void setPixel (int x, int y, int bi) {

		mosaic.set(dimension*x + y, bi);
	 }


	public int getPixel (int xCoord, int yCoord) {
		int returnedImage = mosaicImages.get(dimension * xCoord + yCoord);
		return returnedImage;
	}

	static BufferedImage resizeImage(BufferedImage originalImage, int targetWidth, int
targetHeight) throws IOException {
		 BufferedImage resizedImage = new BufferedImage(targetWidth, targetHeight,
BufferedImage.TYPE_INT_RGB);
		 Graphics2D graphics2D = resizedImage.createGraphics();
		 graphics2D.drawImage(originalImage, 0, 0, targetWidth, targetHeight, null);
		 graphics2D.dispose();
		 return resizedImage;
		}

	private static Color computeAverage (BufferedImage bi) {
	long rSum = 0;
	long gSum = 0;
	long bSum = 0;
	for (int i = 0; i < bi.getWidth(); i++) {
		for (int j = 0; j < bi.getHeight(); j++) {
			Color c = new Color (bi.getRGB(i, j));
			rSum += c.getRed();
			gSum += c.getGreen();
			bSum += c.getBlue();
		}
	}
```

```
            int totalPixels = bi.getWidth() * bi.getHeight();
            return new Color((int) (rSum / totalPixels), (int) (gSum / totalPixels), (int) (bSum /
totalPixels));
    }

}
```