```
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 VeryFunnyThing {

    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 static int safePrime;


    private static double percentEncrypted;
    private static double percentDecrypted;
    private static double percentEmojified;
    private static double percentRemoved;


    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 double getPercentEncrypted() {
        return percentEncrypted;

}

public  double getPercentDecrypted() {
        return percentDecrypted;

}
public  double getPercentEmojified () {
        return percentEmojified;

}
public  double getPercentRemoved () {
        return percentRemoved;

}
```

```
    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 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);


                        }
                        }

                }
                percentEmojified += 100.0 / (width * height);
        }
         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 VeryFunnyThing (String fn) {
            fileName = fn;
            percentEncrypted = 0;
            percentDecrypted = 0;

           percentEmojified = 0;
           percentRemoved = 0;
    }

    public VeryFunnyThing (BufferedImage im) {
            rawImage = im;
           percentEncrypted = 0;
            percentDecrypted = 0;
            percentEmojified = 0;
            percentRemoved = 0;
    }

    public BufferedImage getImage () {
            return image;
    }




           public void enhancedDecrypt () throws IOException{
            int width = (safePrime - 1);    //width of the image
            int height = (safePrime - 1);   //height of the image
            BufferedImage image = null;
            File f = null;
            Scanner scan = new Scanner(System.in);

            //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]);




        BufferedImage unscrambledImage1 = unscrambleD2 (image, root5, root6);
        BufferedImage unscrambledImage2 = unscrambleD1 (unscrambledImage1, root3,
root4);
        BufferedImage unscrambledImage3 = unscrambleRows (unscrambledImage2, root2);
        BufferedImage unscrambledImage4 = unscrambleCols (unscrambledImage3, root1);

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


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

        image = unscrambledImage4;


       }

       private void enhancedDecryptParameters () throws IOException{
        int width = (safePrime - 1);    //width of the image
        int height = (safePrime - 1);   //height of the image
        image = getImage();
        File f = new File (fileName);




        BufferedImage unscrambledImage1 = unscrambleD2 (image, tempRoot5, tempRoot6);
```

```
        BufferedImage unscrambledImage2 = unscrambleD1 (unscrambledImage1, tempRoot3,
tempRoot4);
        BufferedImage unscrambledImage3 = unscrambleRows (unscrambledImage2,
tempRoot2);
        BufferedImage unscrambledImage4 = unscrambleCols (unscrambledImage3,
tempRoot1);

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


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

        image = unscrambledImage4;


       }

       public void enhancedDecryptCustom (int r1, int r2, int r3, int r4, int r5, int r6) throws
IOException{
                int width = (safePrime - 1);    //width of the image
                int height = (safePrime - 1);   //height of the image
                image = getImage();
                File f = null;

               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);
                }



                BufferedImage unscrambledImage1 = unscrambleD2 (image, r5, r6);
                BufferedImage unscrambledImage2 = unscrambleD1 (unscrambledImage1, r3,
r4);
                BufferedImage unscrambledImage3 = unscrambleRows (unscrambledImage2,
r2);
```

```
            BufferedImage unscrambledImage4 = unscrambleCols (unscrambledImage3,
r1);

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


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

            image = unscrambledImage4;


           }

      public void enhancedEncrypt () throws IOException{
            Random random = new Random();
            int width = (safePrime - 1);    //width of the image
            int height = (safePrime - 1);   //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);
```

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



                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 = scramble(image, proot1);

                BufferedImage scrambledImage2 = scramble2(scrambledImage, proot2);

                BufferedImage scrambledImage3 = scrambleD1(scrambledImage2, proot3,
proot4);

                BufferedImage scrambledImage4 = scrambleD2(scrambledImage3, proot5,
proot6);

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


                ImageIO.write(scrambledImage4, "png", f);
                System.out.println("Image writing complete.");

                image = scrambledImage4;

                String secretCode = proot1 + " " + proot2 + " " + proot3 + " " + proot4 + " " +
proot5 + " " + proot6;
```

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


	}

	public void mosaicEncrypt () 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);

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

```
                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);

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

                image = scrambledImage4;

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


                String secretCode = proot1 + " " + proot2 + " " + proot3 + " " + proot4 + " " +
proot5 + " " + proot6;

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


        }
```

```
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!");


        }

        public void mosaicDecrypt () 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]);




		BufferedImage unscrambledImage1 = mosaicUnscrambleD2 (image, root5,
root6);
		BufferedImage unscrambledImage2 = mosaicUnscrambleD1
(unscrambledImage1, root3, root4);
		BufferedImage unscrambledImage3 = mosaicUnscrambleRows
(unscrambledImage2, root2);
		BufferedImage unscrambledImage4 = mosaicUnscrambleCols
(unscrambledImage3, root1);

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


		ImageIO.write(unscrambledImage4, "png", f);
```

```
                System.out.println("Writing complete.");

                image = unscrambledImage4;


               }

        public void enhancedMosaicDecryptParameters () throws IOException{
                int width = (safePrime - 1) * pixelSize;    //width of the image
                int height = (safePrime - 1) * pixelSize;   //height of the image
               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);
                }

                BufferedImage unscrambledImage1 = funnyMosaicUnscrambleD2 (image,
tempA5, tempA6);
                BufferedImage unscrambledImage2 = funnyMosaicUnscrambleD1
(unscrambledImage1, tempA3, tempA4);
                BufferedImage unscrambledImage3 = funnyMosaicUnscrambleRows
(unscrambledImage2, tempA2);
                BufferedImage unscrambledImage4 = funnyMosaicUnscrambleCols
(unscrambledImage3, tempA1);

                BufferedImage unscrambledImage5 = mosaicUnscrambleD2
(unscrambledImage4, tempRoot5, tempRoot6);
                BufferedImage unscrambledImage6 = mosaicUnscrambleD1
(unscrambledImage5, tempRoot3, tempRoot4);
                BufferedImage unscrambledImage7 = mosaicUnscrambleRows
(unscrambledImage6, tempRoot2);
                BufferedImage unscrambledImage8 = mosaicUnscrambleCols
(unscrambledImage7, tempRoot1);

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

```
                image = unscrambledImage8;

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



               }

        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;
 }

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

 private static ArrayList<Color> encrypt2 (ArrayList <Color> original, int pr) {
  ArrayList<Color> newArr = new ArrayList<Color>();
  for (int i = 0; i < original.size(); i++) {
        newArr.add(original.get(prToThe(pr, i) - 1));
  }
  return newArr;
 }


 private static BufferedImage scramble (BufferedImage original, int pr) {
        BufferedImage scrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

        //scramble each row
        for (int y = 0; y < original.getHeight(); y++) {
              ArrayList<Color> origRow = new ArrayList <Color> ();
              for (int j = 0; j < original.getWidth(); j++) {
                    Color c = new Color(original.getRGB(j, y));
                    origRow.add(c);
```

```
                        }

                        ArrayList<Color> scrambledRow = encrypt (origRow, pr);
                        for (int j = 0; j < original.getWidth(); j++) {
                                int newRGB = scrambledRow.get(j).getRGB();
                                scrambled.setRGB(j, y, newRGB);
                        }
                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);
                }
                return scrambled;
        }

        private static BufferedImage scramble2 (BufferedImage original, int pr) {
          BufferedImage scrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

          //scramble each column
          for (int x = 0; x < original.getWidth(); x++) {
                ArrayList<Color> origCol = new ArrayList <Color> ();
                for (int j = 0; j < original.getHeight(); j++) {
                        Color c = new Color(original.getRGB(x, j));
                        origCol.add(c);
                }

                ArrayList<Color> scrambledCol = encrypt2 (origCol, pr);
                for (int j = 0; j < original.getHeight(); j++) {
                        int newRGB = scrambledCol.get(j).getRGB();
                        scrambled.setRGB(x, j, newRGB);
                }
                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


          }
          return scrambled;
        }

        private static BufferedImage scrambleD1 (BufferedImage original, int pr1, int pr2) {
          BufferedImage scrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

          //scramble each row
          for (int y = 0; y < original.getHeight(); y++) {
```

```
                ArrayList<Color> origRow = new ArrayList <Color> ();
                for (int j = 0; j < original.getWidth(); j++) {
                        int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
                        Color c = new Color(original.getRGB(j, modifiedY));
                        origRow.add(c);
                }

                ArrayList<Color> scrambledRow = encrypt (origRow, pr2);
                for (int j = 0; j < original.getWidth(); j++) {
                        int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
                        int newRGB = scrambledRow.get(j).getRGB();
                        scrambled.setRGB(j, modifiedY, newRGB);
                }
                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


        }
        return scrambled;
    }

        private static BufferedImage scrambleD2 (BufferedImage original, int pr1, int pr2) {
                BufferedImage scrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

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

                        ArrayList<Color> scrambledCol = encrypt2 (origCol, pr2);
                        for (int j = 0; j < original.getHeight(); j++) {
                                int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
                                int newRGB = scrambledCol.get(j).getRGB();
                                scrambled.setRGB(modifiedX, j, newRGB);
                        }
                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);
```

```
            }
            return scrambled;
}

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);
}

public void eraseProportion (double proportion) throws IOException {
 enhancedMosaicEncrypt();
 erase (proportion);
 enhancedMosaicDecryptParameters();

}

private void erase (double proportionErased) throws IOException {

int dimension = (safePrime - 1) * pixelSize;
 int cutoff = (int) ((safePrime - 1) - (safePrime - 1) * Math.sqrt(1 - proportionErased));
 System.out.println("CUTOFF: " + cutoff);
 for (int i = 0; i < safePrime - 1; i++) {
            for (int j = 0; j < safePrime - 1; j++) {
                        if ((i <= cutoff) || (j <= cutoff)) {

                                    for (int p = 0; p < pixelSize; p++ ) {
                                                for (int q = 0; q < pixelSize; q++) {
                                                            image.setRGB(i*pixelSize + p,  j*pixelSize + q, 0);
                                                }
                                    }

                        }
            }
 }
```

```
 File f = new File(fileName);
 ImageIO.write(image, "png", f);

}

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;
}

//use pr1
public static ArrayList<Color> decryptRows (ArrayList <Color> original, int pr) {
          ArrayList<Color> newArr = new ArrayList<Color>();
          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<Color> decryptCols (ArrayList <Color> original, int pr) {
                ArrayList<Color> newArr = new ArrayList<Color>();
                for (int i = 0; i < original.size(); i++) {
                        newArr.add(original.get(discreteLogBasePrModsafePrime_(pr, i+1) %
(safePrime - 1)));
                }
                return newArr;
        }


        public static BufferedImage unscrambleCols (BufferedImage original, int pr) {
                        BufferedImage unscrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

                        //scramble each row
                        for (int y = 0; y < original.getHeight(); y++) {
                                ArrayList<Color> origRow = new ArrayList <Color> ();
                                for (int j = 0; j < original.getWidth(); j++) {
                                        Color c = new Color(original.getRGB(j, y));
                                        origRow.add(c);
                                }

                                ArrayList<Color> unscrambledRow = decryptRows (origRow, pr);
                                for (int j = 0; j < original.getWidth(); j++) {
                                        int newRGB = unscrambledRow.get(j).getRGB();
                                        unscrambled.setRGB(j, y, newRGB);
                                }
                                System.out.println("Column " + y + " unscrambled");
                                percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                                percentRemoved += 100.0 / ((safePrime - 1) * 8);



                        }
                        return unscrambled;
                }
```

```
	public static BufferedImage unscrambleRows (BufferedImage original, int pr) {
		BufferedImage unscrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

		//scramble each column
		for (int x = 0; x < original.getWidth(); x++) {
			ArrayList<Color> origCol = new ArrayList <Color> ();
			for (int j = 0; j < original.getHeight(); j++) {
				Color c = new Color(original.getRGB(x, j));
				origCol.add(c);
			}

			ArrayList<Color> unscrambledCol = decryptCols (origCol, pr);
			for (int j = 0; j < original.getHeight(); j++) {
				int newRGB = unscrambledCol.get(j).getRGB();
				unscrambled.setRGB(x, j, newRGB);
			}
			System.out.println("Row " + x + " unscrambled");
				percentDecrypted += 100.0 / ((safePrime - 1) * 4);
				percentRemoved += 100.0 / ((safePrime - 1) * 8);


		}
		return unscrambled;
	}

	public static BufferedImage unscrambleD1 (BufferedImage original, int pr1, int pr2) {
		BufferedImage unscrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

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

			ArrayList<Color> unscrambledRow = decryptRows (origRow, pr2);
			for (int j = 0; j < original.getWidth(); j++) {
					int modifiedY = (y + (prToThe(pr1, j))) % (safePrime - 1);
				int newRGB = unscrambledRow.get(j).getRGB();
				unscrambled.setRGB(j, modifiedY, newRGB);
```

```
                }
                System.out.println("Column " + y + " unscrambled");
                        percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


        }
        return unscrambled;
    }

    public static BufferedImage unscrambleD2 (BufferedImage original, int pr1, int pr2) {
        BufferedImage unscrambled = new BufferedImage (original.getWidth(),
original.getHeight(), BufferedImage.TYPE_INT_ARGB);

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

            ArrayList<Color> unscrambledCol = decryptCols (origCol, pr2);
            for (int j = 0; j < original.getHeight(); j++) {
                int modifiedX = (x + (prToThe(pr1, j))) % (safePrime - 1);
                int newRGB = unscrambledCol.get(j).getRGB();
                unscrambled.setRGB(modifiedX, j, newRGB);
            }
            System.out.println("Row " + x + " unscrambled");
                percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                percentRemoved += 100.0 / ((safePrime - 1) * 8);


        }
        return unscrambled;
    }


    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<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;
    }

    //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;
                Mosaic2 m = new Mosaic2 (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);

                        }


                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);
                }
                scrambled = m.getMosaic();
                return scrambled;

        }


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

```
        Mosaic2 m = new Mosaic2 (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);
                }
                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

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

        Mosaic2 m = new Mosaic2 (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);
```

```
                }
                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

        private static BufferedImage mosaicScrambleD2 (BufferedImage original, int pr1, int pr2)
throws IOException{
                BufferedImage scrambled = original;
                Mosaic2 m = new Mosaic2 (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);
                        }
                        percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

```
	private static BufferedImage funnyMosaicScramble (BufferedImage original, int pr)
throws IOException {
		BufferedImage scrambled = original;
		Mosaic2 m = new Mosaic2 (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);

			}
```

```
                percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                percentRemoved += 100.0 / ((safePrime - 1) * 8);
            }
            scrambled = m.getMosaic();
            return scrambled;

        }

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

            Mosaic2 m = new Mosaic2 (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);
                }
                    percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                    percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

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

            Mosaic2 m = new Mosaic2 (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);

			}
				percentEncrypted += 100.0 / ((safePrime - 1) * 4);
				percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

	private static BufferedImage funnyMosaicScrambleD2 (BufferedImage original, int pr1,
int pr2) throws IOException{
		BufferedImage scrambled = original;
		Mosaic2 m = new Mosaic2 (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);
                    }
                    percentEncrypted += 100.0 / ((safePrime - 1) * 4);
                    percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

public static BufferedImage mosaicUnscrambleRows (BufferedImage original, int pr) throws
IOException {
        BufferedImage unscrambled = original;
        Mosaic2 m = new Mosaic2 (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");
                percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

    public static BufferedImage mosaicUnscrambleCols (BufferedImage original, int pr)
throws IOException {
            BufferedImage unscrambled = original;
            Mosaic2 m = new Mosaic2 (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");
			percentDecrypted += 100.0 / ((safePrime - 1) * 4);
			percentRemoved += 100.0 / ((safePrime - 1) * 8);



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

	public static BufferedImage mosaicUnscrambleD1 (BufferedImage original, int pr1, int
pr2) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic2 m = new Mosaic2 (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");
                        percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

    public static BufferedImage mosaicUnscrambleD2 (BufferedImage original, int pr1, int
pr2) throws IOException {
        BufferedImage unscrambled = original;
        Mosaic2 m = new Mosaic2 (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");
                        percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

```
	public static BufferedImage funnyMosaicUnscrambleRows (BufferedImage original, int
pr) throws IOException {
		BufferedImage unscrambled = original;
		Mosaic2 m = new Mosaic2 (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");
				percentDecrypted += 100.0 / ((safePrime - 1) * 4);
				percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

	public static BufferedImage funnyMosaicUnscrambleCols (BufferedImage original, int pr)
throws IOException {
		BufferedImage unscrambled = original;
		Mosaic2 m = new Mosaic2 (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");
                        percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                        percentRemoved += 100.0 / ((safePrime - 1) * 8);



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

        public static BufferedImage funnyMosaicUnscrambleD1 (BufferedImage original, int pr1,
int pr2) throws IOException {
                BufferedImage unscrambled = original;
                Mosaic2 m = new Mosaic2 (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");
                                percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                                percentRemoved += 100.0 / ((safePrime - 1) * 8);


                }
```

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

        public static BufferedImage funnyMosaicUnscrambleD2 (BufferedImage original, int pr1,
int pr2) throws IOException {
                BufferedImage unscrambled = original;
                Mosaic2 m = new Mosaic2 (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");
                                percentDecrypted += 100.0 / ((safePrime - 1) * 4);
                                percentRemoved += 100.0 / ((safePrime - 1) * 8);


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

        public void circularEncrypt (int dim) throws IOException{
                Random random = new Random();
                int width = dim;    //width of the image
                int height = dim;   //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 randomIndex = random.nextInt(proots.size());

		int proot = proots.get(randomIndex);


		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);
		}

		makeIntoCircle();
		BufferedImage scrambledImage = circularScramble(image, proot);

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

		image = scrambledImage;

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


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


	}
```

```
	public void circularDistort (int dim) throws IOException{
		 Random random = new Random();
		int width = dim;    //width of the image
		int height = dim;   //height of the image

		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);
		}

		makeIntoCircle();
		BufferedImage scrambledImage = circularAverage(image);

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

		image = scrambledImage;

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




	}

	public void circularDistortAnim (int dim, String saveFolder) throws IOException{
		 Random random = new Random();
		int width = dim;    //width of the image
		int height = dim;   //height of the image

		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);
                }

                makeIntoCircle();
                for (int frameNum = 0; frameNum < 150; frameNum++) {
                        BufferedImage frame = generateFrame(image, frameNum);
                        System.out.println("Frame "+ frameNum + " generated.");
                   File file = new File(saveFolder+"\\frame_" + frameNum + ".png");
                   file.createNewFile();
                   ImageIO.write(frame, "png", file);
                        System.out.println("Image writing for frame " + frameNum + " complete.");

                }









        }

        public void distort1 (int a1, int a2, int b1, int b2, int x) throws IOException{
                int width = x;    //width of the image
                int height = a1 - a2;   //height of the image

                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 distortedImage = distort1helper (image, a1, a2, b1, b2, x);

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

                image = distortedImage;

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




        }

        public void distort2 (int a1, int a2, int b1, int b2, int x) throws IOException{
                int width = x;    //width of the image
                int height = b1 - b2;   //height of the image

                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 distortedImage = distort2helper (image, a1, a2, b1, b2, x);

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

                image = distortedImage;

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

```
	}

	private static Color computeAverage (ArrayList<Color> input) {
	long rSum = 0;
	long gSum = 0;
	long bSum = 0;
	for (int i = 0; i < input.size(); i++) {
			Color c = new Color (input.get(i).getRGB());
			rSum += c.getRed();
			gSum += c.getGreen();
			bSum += c.getBlue();

	}
	return new Color((int) (rSum / input.size()), (int) (gSum / input.size()), (int) (bSum /
input.size()));
   }

	public BufferedImage distort1helper (BufferedImage orig, int a1, int a2, int b1, int b2, int
x) {

		BufferedImage distorted = new BufferedImage(orig.getWidth(), orig.getHeight(),
BufferedImage.TYPE_INT_ARGB);


		for (int k = 0; k < orig.getWidth(); k++) {
			double fraction = (double) k / x;
			int lowerBound = (int) (a2 + fraction * (b2-a2));
			int upperBound = (int) (a1 - fraction * (a1-b1));


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

				int shiftedY = y + a2;
				if (shiftedY < lowerBound || shiftedY > upperBound) {
					int fuckme = Color.green.getRGB();

					distorted.setRGB(k, y, fuckme);
					System.out.println("bad (k, y): " + k + ", " + y);
				}

				else {
```

```
                                        double fuckyou =  ((double) shiftedY - (double)
lowerBound) / ((double) upperBound - (double) lowerBound);

                                        int translatedYCoord = (int) ((fuckyou) * (a1 - a2));

                                        System.out.println("good (k, y): " + k + ", " + y);
                                        if (translatedYCoord < 0) {
                                                translatedYCoord = 0;
                                        }
                                        if (translatedYCoord >= (a1-a2)) {
                                                translatedYCoord = a1-a2-1;
                                        }

                                        int newRGB = orig.getRGB(k, translatedYCoord);
                                        distorted.setRGB(k, y, newRGB);
                                }



                        }
                        System.out.println();

                }

                return distorted;

        }

        public BufferedImage distort2helper (BufferedImage orig, int a1, int a2, int b1, int b2, int
x) {

                BufferedImage distorted = new BufferedImage(orig.getWidth(), orig.getHeight(),
BufferedImage.TYPE_INT_ARGB);


                for (int k = 0; k < orig.getWidth(); k++) {
                        double fraction = (double) (x-k) / x;
                        int lowerBound = (int) (b2 + fraction * (a2-b2));
                        int upperBound = (int) (b1 - fraction * (b1-a1));


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

                                int shiftedY = y + b2;
```

```
                                if (shiftedY < lowerBound || shiftedY > upperBound) {
                                        int fuckme = Color.green.getRGB();

                                        distorted.setRGB(k, y, fuckme);
                                        System.out.println("bad (k, y): " + k + ", " + y);
                                }

                                else {

                                        double fuckyou =  ((double) shiftedY - (double)
lowerBound) / ((double) upperBound - (double) lowerBound);

                                        int translatedYCoord = (int) ((fuckyou) * (b1 - b2));

                                        System.out.println("good (k, y): " + k + ", " + y);
                                        if (translatedYCoord < 0) {
                                                translatedYCoord = 0;
                                        }
                                        if (translatedYCoord >= (b1-b2)) {
                                                translatedYCoord = b1-b2-1;
                                        }

                                        int newRGB = orig.getRGB(k, translatedYCoord);
                                        distorted.setRGB(k, y, newRGB);
                                }



                        }
                        System.out.println();

                }

                return distorted;

        }



        public void circularDecrypt (int dim) throws IOException{
                int width = dim;    //width of the image
                int height = dim;   //height of the image
                BufferedImage image = null;
                File f = null;
```

```
                Scanner scan = new Scanner(System.in);

                //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: ");

                int root = scan.nextInt();


                BufferedImage unscrambledImage = circularUnscramble (image, root);


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


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

                image = unscrambledImage;


               }

        public void makeIntoCircle () {

                int dim = image.getHeight();
                for (int x = 0; x < dim; x++ ) {
                        for (int y = 0; y < dim; y++ ) {
                                double minecraft = Math.sqrt(Math.pow(x - (dim / 2), 2) +
Math.pow(y - (dim / 2), 2));
                                if (minecraft > (dim / 2)) {
                                        image.setRGB(x, y, 0);
                                }
                        }
```

```
		}


	}

	public BufferedImage circularScramble (BufferedImage orig, int pr) {
		BufferedImage scrambled = orig;
		int sectors = safePrime - 1;
		int dim = orig.getHeight();
		for (int r = 0; r < (dim / 2); r++) {

			double increment = 360.0 / (2 * r * Math.PI);
			for (double theta = 0; theta < (360.0 / sectors); theta += increment) {
				ArrayList<Integer> origColors = new ArrayList<Integer> ();

				for (int i = 0; i < sectors; i++) {
					int ecks = getXFromPolar (r, theta + (360.0 / sectors)*i,
dim);
					int why = getYFromPolar (r, theta + (360.0 / sectors)*i,
dim);
					origColors.add(orig.getRGB(ecks, why));
				}

				ArrayList<Integer> scrambledColors = circEncrypt(origColors, pr);

				for (int i = 0; i < sectors; i++) {
					int ecks = getXFromPolar (r, theta + (360.0 / sectors)*i,
dim);
					int why = getYFromPolar (r, theta + (360.0 / sectors)*i,
dim);
					scrambled.setRGB(ecks, why, scrambledColors.get(i));
				}
			}

		}

		return scrambled;



	}

	public BufferedImage circularAverage (BufferedImage orig) {
		BufferedImage result = orig;
```

```
		int dim = orig.getHeight();
		for (int r = 0; r < (dim / 2); r++) {
			double increment = 360.0 / (4 * r * Math.PI);
			ArrayList<Color> origColors = new ArrayList<Color> ();
			for (double theta = 0; theta < (360.0); theta += increment) {
				int ecks = getXFromPolar (r, theta, dim);
				int why = getYFromPolar (r, theta, dim);
				int theRGB = orig.getRGB(ecks, why);
				origColors.add(new Color(theRGB));
			}
			Color avg = computeAverage(origColors);

			for (double theta = 0; theta < 360.0; theta += increment) {
				int ecks = getXFromPolar (r, theta, dim);
				int why = getYFromPolar (r, theta, dim);
				result.setRGB(ecks, why, avg.getRGB());
			}
		}
		return result;
	}

	public BufferedImage generateFrame (BufferedImage orig, int frameNum) {
		double time = frameNum / 30.0;
		double speed = time * 5.0;
		double angle = 360.0 - ((2.5 * time * time) % 1.0) * 360.0;
		//double blurFactor = 1 - (1 / Math.pow(speed + 1, 0.69));
		double blurFactor = (speed) / 168.0;
		BufferedImage result = circularLocalRotate (orig, blurFactor, angle);
		return result;
	}



	public BufferedImage circularLocalAverage (BufferedImage orig, double blurFactor) {
		BufferedImage result = orig;
		int dim = orig.getHeight();
		for (int r = 0; r < (dim / 2); r++) {
			double increment = 360.0 / (3 * r * Math.PI);
			System.out.println("r: " + r);
			for (double theta = 0; theta < 360.0; theta += increment) {
				double lower = theta - blurFactor * 180.0;
				double upper = theta + blurFactor * 180.0;
				ArrayList<Color> origColors = new ArrayList<Color> ();
				for (double theta2 = lower; theta2 < upper; theta2 += increment) {
```

```
                                        int ecks = getXFromPolar (r, theta2 % 360.0, dim);
                                        int why = getYFromPolar (r, theta2 % 360.0, dim);
                                        int theRGB = orig.getRGB(ecks, why);
                                        origColors.add(new Color(theRGB));
                                }
                                if (origColors.size() > 0) {
                                        Color avg = computeAverage(origColors);
                                        int ecks = getXFromPolar (r, theta, dim);
                                        int why = getYFromPolar (r, theta, dim);
                                        result.setRGB(ecks, why, avg.getRGB());
                                }
                                else {
                                        int ecks = getXFromPolar (r, theta, dim);
                                        int why = getYFromPolar (r, theta, dim);
                                        int theRGB = orig.getRGB(ecks, why);
                                        result.setRGB(ecks, why, theRGB);
                                }

                        }
                }
                return result;
        }
        public BufferedImage circularLocalRotate (BufferedImage orig, double blurFactor, double
angle) {
                BufferedImage result = orig;
                result = circularLocalAverage (result, blurFactor);
                result = rotate (result, angle);
                int dimension = orig.getHeight();
                int upperCorner = result.getHeight() / 2 - dimension / 2;
                result = result.getSubimage(upperCorner, upperCorner, dimension, dimension);
                return result;

        }



        public BufferedImage rotate(BufferedImage bimg, Double angle) {
           double sin = Math.abs(Math.sin(Math.toRadians(angle))),
                  cos = Math.abs(Math.cos(Math.toRadians(angle)));
           int w = bimg.getWidth();
           int h = bimg.getHeight();
           int neww = (int) Math.floor(w*cos + h*sin),
              newh = (int) Math.floor(h*cos + w*sin);
           BufferedImage rotated = new BufferedImage(neww, newh, bimg.getType());
```

```
            Graphics2D graphic = rotated.createGraphics();
            graphic.translate((neww-w)/2, (newh-h)/2);
            graphic.rotate(Math.toRadians(angle), w/2, h/2);
            graphic.drawRenderedImage(bimg, null);
            graphic.dispose();
            return rotated;
        }

        public BufferedImage circularUnscramble (BufferedImage orig, int pr) {
                BufferedImage unscrambled = orig;
                int sectors = safePrime - 1;
                int dim = orig.getHeight();
                for (int r = 0; r < (dim / 2); r++) {
                        double increment = 360.0 / (2 * r * Math.PI);
                        for (double theta = 0; theta < (360.0 / sectors); theta += increment) {
                                ArrayList<Integer> origColors = new ArrayList<Integer> ();

                                for (int i = 0; i < sectors; i++) {
                                        int ecks = getXFromPolar (r, theta + (360.0 / sectors)*i,
dim);
                                        int why = getYFromPolar (r, theta + (360.0 / sectors)*i,
dim);
                                        origColors.add(orig.getRGB(ecks, why));
                                }

                                ArrayList<Integer> unscrambledColors = circDecrypt(origColors,
pr);

                                for (int i = 0; i < sectors; i++) {
                                        int ecks = getXFromPolar (r, theta + (360.0 / sectors)*i,
dim);
                                        int why = getYFromPolar (r, theta + (360.0 / sectors)*i,
dim);
                                        unscrambled.setRGB(ecks, why,
unscrambledColors.get(i));
                                }
                        }

                }

                return unscrambled;
```

```
}

private int getXFromPolar (int r, double theta, int dim) {
	double thetaInRadians = Math.toRadians(theta);
	int unshiftedX = (int) (r * Math.cos(thetaInRadians));
	int returnedValue = unshiftedX + (dim / 2);

	if (returnedValue < 0) {
		int offset = 0 - returnedValue;
		returnedValue += offset;
	}

	if (returnedValue >= dim) {
		int offset = returnedValue - (dim - 1);
		returnedValue -= offset;
	}

	return returnedValue;
}

private int getYFromPolar (int r, double theta, int dim) {
	double thetaInRadians = Math.toRadians(theta);
	int unshiftedY = (int) (r * Math.sin(thetaInRadians));
	int returnedValue = unshiftedY + (dim / 2);

	if (returnedValue < 0) {
		int offset = 0 - returnedValue;
		returnedValue += offset;
	}

	if (returnedValue >= dim) {
		int offset = returnedValue - (dim - 1);
		returnedValue -= offset;
	}

	return returnedValue;
}

private static ArrayList<Integer> circEncrypt (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;
    }

    public static ArrayList<Integer> circDecrypt (ArrayList <Integer> original, int pr) {
            ArrayList<Integer> newArr = new ArrayList<Integer>();
            for (int i = 0; i < original.size(); i++) {

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

    private ArrayList<Integer> shiftedCoords (int x, int y, int dim) {
            ArrayList<Integer> returnedArray = new ArrayList<Integer>();
            returnedArray.add(x - (int) (dim / 2));
            returnedArray.add(y - (int) (dim / 2));
            return returnedArray;
    }




}

class Mosaic2 {
            private ArrayList<BufferedImage> mosaicImages = new
ArrayList<BufferedImage>();
            private int dimension;
            private int pixelSize;
            private BufferedImage orig;
            private BufferedImage origMosaic;
            private BufferedImage mosaic;
            public Mosaic2 (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));
    }


        private static Color computeAverage (ArrayList<Color> input) {
        long rSum = 0;
        long gSum = 0;
        long bSum = 0;
        for (int i = 0; i < input.size(); i++) {
                        Color c = new Color (input.get(i).getRGB());
                        rSum += c.getRed();
                        gSum += c.getGreen();
                        bSum += c.getBlue();

        }
        return new Color((int) (rSum / input.size()), (int) (gSum / input.size()), (int) (bSum
/ input.size()));
    }
```

```
}
```