Teoría de la Información y Codificación Práctica 4: Simulación de un canal de comunicación con errores

José A. Montenegro Montes

26 de septiembre de 2014

1. Enunciado

Esta práctica tiene como objetivo simular un canal de comunicaciones binario simétrico. El canal de comunicaciones será definido mediante una probabilidad de error e, por tanto, la probabilidad de error tendrá que ser el parámetro del constructor de la clase *Canal*.

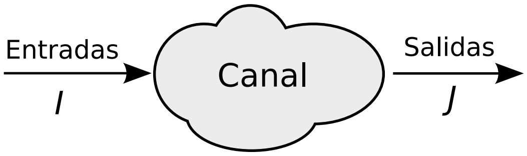

Figura 1: Representación visual de un canal

El canal sera definido siguiendo el modelo de canal simétrico binario explicado en clase:

Definición 1 (Canal Binario Simétrico) *Un canal binario simétrico (BSC) corresponde a un matriz de la forma*

$$\Gamma = \begin{pmatrix} \Gamma_{00} & \Gamma_{01} \\ \Gamma_{10} & \Gamma_{11} \end{pmatrix} = \begin{pmatrix} 1-e & e \\ e & 1-e \end{pmatrix}$$

Para tener una constancia visual de los errores introducidos en el canal, transmitiremos por el canal una imagen (véase figura 3). El formato escogido es *bmp* sin comprimir, debido a que las modificaciones introducidas por el canal son visualizadas directamente en la imagen.

Para facilitar el manejo de archivos bmp se aporta las clases necesarias para el manejo de archivos *bmp* (*BMPHandler*). El alumno tendrá que adaptar la

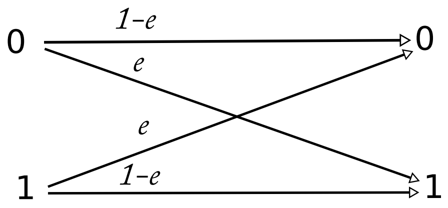

Figura 2: El canal binario simétrico con probabilidad de error de bit e

información que recibe de la libería de bmp y convertir a un flujo de bits que son introducidos en el canal.

Como resultado, visualizaremos una imagen con ruido, que dependerá del error definido en el canal. Por ejemplo, la imagen 4 es la imagen resultante con un canal de error 1, y la figura 5 en el caso que el error sea 0.01. La practica deberá mostar la imagen original y la imagen resultante, para ello se podrá hacer uso de la clase *Visual* aportada, como por ejemplo las figuras 3 4 y 5.

Además es necesario calcular la **entropía** de la imagen origen y destino para evaluar como modifica la **entropía** al paso de los datos por un canal con ruido.

Por ejemplo, el código que mostramos a continuación puede ser parte de la clase de esta práctica, donde se visualizan una imagen, es enviada por el canal con una tasa de error, p.ej. 0.01, y finalmente visualizamos la imagen que pasa por el canal. Además el método *compara* nos permite obtener las diferencias entre dos imágenes *bmp*.

```
  private static void compara (String ori,String mod){

        CompararImagenes comparar = new CompararImagenes(ori,mod);
         int dif=comparar.diferencias2Imagenes();
         int dif3canales=comparar.diferencias2Imagenes3canales();
         int dif3canalesBit=comparar.diferenciasBit2Imagenes3canales();

         System.out.println("Diferencia imagenes: "+dif+" Diferencias 3 canales: "+dif3canales);
         System.out.println("Diferencias 3 canales BIT: "+dif3canalesBit);
         System.out.println("************************");
  }

public static void main(String[] args) {

  String imagenNormal ="android.bmp";
  Visual normalV = new Visual ();
  normalV.load(imagenNormal);
  Canal c = new Canal (0.01);
  imagenCanal= c.canal(imagenNormal);
  Visual canalV = new Visual ();
  canalV.load(imagenCanal);
  compara(imagenNormal,imagenCanal);
}
```

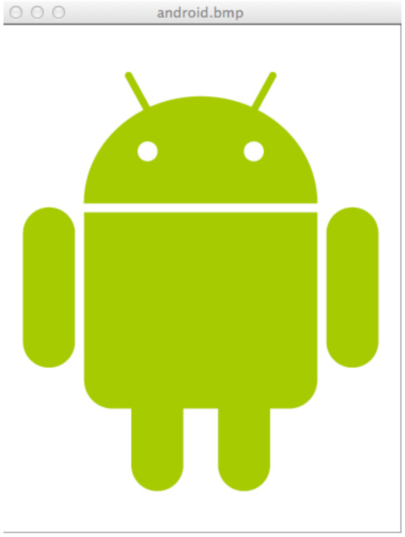

Figura 3: Imagen original utilizada en las prácticas.

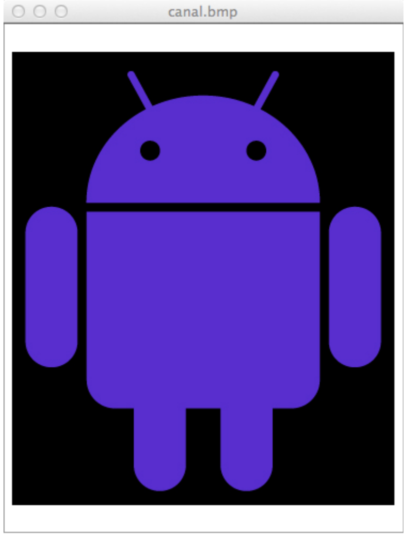

Figura 4: Imagen resultante tras pasar por un canal con tasa de error 1.

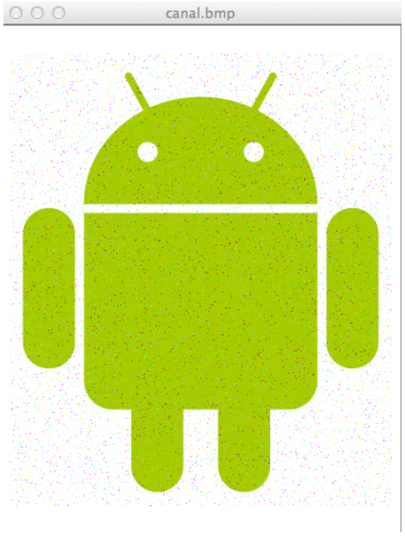

Figura 5: Imagen resultante tras pasar por un canal con tasa de error 0.01.

2. Conclusiones

La práctica simula mediante software un canal de comunicaciones con errores. Este será el primer paso para establecer el entorno necesario que nos permita aplicar los algoritmos de corrección de errores.

Además de crear el canal, tenemos las herramientas necesarias para comparar imágenes que nos serán de utilidad en posteriores prácticas, de la misma forma que hicimos uso de la librería *diffutils* en la práctica anterior.

El alumno observará como el paso de la información por el canal influye en la entropía de la información que es enviada por el canal.

3. Código

Clase BMPHandler.

```
package Practica4;

import java.awt.*;
import java.awt.image.MemoryImageSource;
import java.awt.image.PixelGrabber;
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class BMPHandler {

        private final static int BITMAPFILEHEADER_SIZE = 14;
        private final static int BITMAPINFOHEADER_SIZE = 40;

        //--- Bitmap file header
        // private byte bitmapFileHeader [] = new byte [14];
         private byte bfType [] = {(byte)'B', (byte)'M'};
         private int bfSize = 0;
         private int bfReserved1 = 0;
         private int bfReserved2 = 0;
         private int bfOffBits = BITMAPFILEHEADER_SIZE + BITMAPINFOHEADER_SIZE;

         //--- Bitmap info header
         //private byte bitmapInfoHeader [] = new byte [40];
         private int biSize = BITMAPINFOHEADER_SIZE;
         private int biWidth = 0;
         private int biHeight = 0;
         private int biPlanes = 1;
         private int biBitCount = 24;
         private int biCompression = 0;
         private int biSizeImage = 0x030000;
         private int biXPelsPerMeter = 0x0;
         private int biYPelsPerMeter = 0x0;
         private int biClrUsed = 0;
         private int biClrImportant = 0;

         //--- Bitmap raw data
         private int bitmap [];

         //--- File section
         private FileOutputStream fo;


         /**********
          *
          * @param openName
          * @return
          */

 public static  int[][][] loadBMP(String openName) {
     Image img = (Image)null;  // create a null Image


             img = loadbitmap("./", openName);

             // If the image did not load, print an explanatory
             // message to the user and ask him/her to try again.
             if (img == null) {
                 System.out.println("Could not read an image from file "+openName);
                 System.out.println("Make sure that you supply the name of an image file, \nand that you include
                 return null;
             }  // if (img==null)


     // Translate from Image img to a 3D array "imagePixels".
     // Using this 3D array, imagePixels[r][c][w] gives the value
     // of row r, column c, colour w.
     int[][][] imagePixels = getImagePixels(img);
     return imagePixels;
} // end of method loadImage

          /*****************
```

```
           *
           * @param imagePixels
           * @param saveName
           */

              public void saveImage(String saveName, int[][][] imagePixels){
                  int height = imagePixels.length;
                  int width = imagePixels[0].length;
                  int[][] flat = new int[width*height][4];


                  // If saveName does not already end in .bmp, then add .bmp to saveName.
                  saveName=bmpTack(saveName);

                  // Flatten the image into a 2D array.
                  int index=0;
                  for(int row=0; row<height; row++) {
                      for(int col=0; col<width; col++) {
                          for(int rgbo=0; rgbo<4; rgbo++) {
                              flat[index][rgbo]=imagePixels[row][col][rgbo];
                          }
                          index++;
                      }  // for col
                  }  // for row

                  // Combine the 8-bit red, green, blue, offset values into 32-bit words.
                  int[] outPixels = new int[flat.length];
                  for(int j=0; j<flat.length; j++) {
                      outPixels[j] = ((flat[j][0]&0xff)<<16) | ((flat[j][1]&0xff)<<8)
                                      | (flat[j][2]&0xff) | ((flat[j][3]&0xff)<<24);
                  } // for j

                  // Write the data out to file with the name given by string saveName.
                  saveBitmap(saveName, outPixels, width, height);

              }  // end of method saveImage

/*********
 *
 * @param img
 * @return
 */
    private static  int[][][] getImagePixels(Image img) {

        // Get the raw pixel data
        iObserver observer = new iObserver();
        int width1 = img.getWidth(observer);
        int height1 = img.getHeight(observer);
        int[] rawPixels = getPixels(img,width1,height1);

        // Each pixel is represented by 32 bits.  Separate the tH32 bits into
        // four 8-bit values (red, green, blue, offset).
        int[][] rgbPixels = new int[rawPixels.length][4];
        for(int j=0; j<rawPixels.length; j++) {
            rgbPixels[j][0] = ((rawPixels[j]>>16)&0xff);
            rgbPixels[j][1] = ((rawPixels[j]>>8)&0xff);
            rgbPixels[j][2] = (rawPixels[j]&0xff);
            rgbPixels[j][3] =((rawPixels[j]>>24)&0xff);
        }  // for j

        // Arrange the data by rows and columns
        int[][][] imagePixels = new int[height1][width1][4];
        int index=0;
        for(int row=0; row<imagePixels.length; row++) {
            for(int col=0; col<imagePixels[0].length; col++) {
                for(int rgbo=0; rgbo<4; rgbo++) {
                    imagePixels[row][col][rgbo]=rgbPixels[index][rgbo];
                } // for
                index++;
            } // for col
        }  // for row
        return imagePixels;
    } // end of method getImagePixels





/*************
 *
 * @param name
 * @return
 */


    private static String bmpTack(String name) {
        if (name.endsWith(".bmp"))
            return name;
        else
            return name+".bmp";
    }  // end of method bmpTack

 /***************
  *
  * @param sdir
  * @param sfile
  * @return
  */
    public static  Image loadbitmap (String sdir, String sfile){

            Image image;

            try
            {
                FileInputStream fs;
                if(sdir.equals("./")){//a bit of a hack
                    fs=new FileInputStream(sfile);
                }
                else{
                    fs=new FileInputStream(sdir+sfile);
                }

                int bflen=14; // 14 byte BITMAPFILEHEADER
                byte bf[]=new byte[bflen];
```

```java
fs.read(bf,0,bflen);
int bilen=40; // 40-byte BITMAPINFOHEADER
byte bi[]=new byte[bilen];
fs.read(bi,0,bilen);

// Interperet data.
int nsize = (((int)bf[5]&0xff)<<24)
    | (((int)bf[4]&0xff)<<16)
    | (((int)bf[3]&0xff)<<8)
    | (int)bf[2]&0xff;
//System.out.println("File type is :"+(char)bf[0]+(char)bf[1]);
//System.out.println("Size of file is :"+nsize);

int nbisize = (((int)bi[3]&0xff)<<24)
    | (((int)bi[2]&0xff)<<16)
    | (((int)bi[1]&0xff)<<8)
    | (int)bi[0]&0xff;
//System.out.println("Size of bitmapinfoheader is :"+nbisize);

int nwidth = (((int)bi[7]&0xff)<<24)
    | (((int)bi[6]&0xff)<<16)
    | (((int)bi[5]&0xff)<<8)
    | (int)bi[4]&0xff;
//System.out.println("Width is :"+nwidth);

int nheight = (((int)bi[11]&0xff)<<24)
    | (((int)bi[10]&0xff)<<16)
    | (((int)bi[9]&0xff)<<8)
    | (int)bi[8]&0xff;
//System.out.println("Height is :"+nheight);


int nplanes = (((int)bi[13]&0xff)<<8) | (int)bi[12]&0xff;
//System.out.println("Planes is :"+nplanes);

int nbitcount = (((int)bi[15]&0xff)<<8) | (int)bi[14]&0xff;
//System.out.println("BitCount is :"+nbitcount);

// Look for non-zero values to indicate compression
int ncompression = (((int)bi[19])<<24)
    | (((int)bi[18])<<16)
    | (((int)bi[17])<<8)
    | (int)bi[16];
//System.out.println("Compression is :"+ncompression);

int nsizeimage = (((int)bi[23]&0xff)<<24)
    | (((int)bi[22]&0xff)<<16)
    | (((int)bi[21]&0xff)<<8)
    | (int)bi[20]&0xff;
//System.out.println("SizeImage is :"+nsizeimage);

int nxpm = (((int)bi[27]&0xff)<<24)
    | (((int)bi[26]&0xff)<<16)
    | (((int)bi[25]&0xff)<<8)
    | (int)bi[24]&0xff;
//System.out.println("X-Pixels per meter is :"+nxpm);

```

```
                int nypm = (((int)bi[31]&0xff)<<24)
                    | (((int)bi[30]&0xff)<<16)
                    | (((int)bi[29]&0xff)<<8)
                    | (int)bi[28]&0xff;
                //System.out.println("Y-Pixels per meter is :"+nypm);

                int nclrused = (((int)bi[35]&0xff)<<24)
                    | (((int)bi[34]&0xff)<<16)
                    | (((int)bi[33]&0xff)<<8)
                    | (int)bi[32]&0xff;
                //System.out.println("Colors used are :"+nclrused);

                int nclrimp = (((int)bi[39]&0xff)<<24)
                    | (((int)bi[38]&0xff)<<16)
                    | (((int)bi[37]&0xff)<<8)
                    | (int)bi[36]&0xff;
                //System.out.println("Colors important are :"+nclrimp);

                if (nbitcount==24)
                    {
                        // No Palatte data for 24-bit format but scan lines are
                        // padded out to even 4-byte boundaries.
                        int npad = (nsizeimage / nheight) - nwidth * 3;
                        //added for Bug correction
                        if(npad == 4){
                            npad=0;
                        }
                        int ndata[] = new int [nheight * nwidth];
                        byte brgb[] = new byte [( nwidth + npad) * 3 * nheight];

                        fs.read (brgb, 0, (nwidth + npad) * 3 * nheight);
                        int nindex = 0;
                        for (int j = 0; j < nheight; j++)
                            {
                                for (int i = 0; i < nwidth; i++)
                                    {
                                        ndata [nwidth * (nheight - j - 1) + i] =

                                            (255&0xff)<<24
                                            | (((int)brgb[nindex+2]&0xff)<<16)
                                            | (((int)brgb[nindex+1]&0xff)<<8)
                                            | (int)brgb[nindex]&0xff;
                                        nindex += 3;
                                    }
                                nindex += npad;
                            }

                        image = Toolkit.getDefaultToolkit().createImage( new MemoryImageSource (nwidth, nheight,n
                    }
                else if (nbitcount == 8)
                    {
                        // Have to determine the number of colors, the clrsused
                        // parameter is dominant if it is greater than zero. If
                        // zero, calculate colors based on bitsperpixel.
                        int nNumColors = 0;
                        if (nclrused > 0)
                            {
```

```
                        nNumColors = nclrused;
                    }
                else
                    {
                        nNumColors = (1&0xff)<<nbitcount;
                    }
                //System.out.println("The number of Colors is"+nNumColors);

                // Some bitmaps do not have the sizeimage field calculated
                // Ferret out these cases and fix 'em.
                if (nsizeimage == 0)
                    {
                        nsizeimage = ((((nwidth*nbitcount)+31) & ~31 ) >> 3);
                        nsizeimage *= nheight;
                        //System.out.println("nsizeimage (backup) is"+nsizeimage);
                    }

                // Read the palatte colors.
                int npalette[] = new int [nNumColors];
                byte bpalette[] = new byte [nNumColors*4];
                fs.read (bpalette, 0, nNumColors*4);
                int nindex8 = 0;
                for (int n = 0; n < nNumColors; n++)
                    {
                        npalette[n] = (255&0xff)<<24
                            | (((int)bpalette[nindex8+2]&0xff)<<16)
                            | (((int)bpalette[nindex8+1]&0xff)<<8)
                            | (int)bpalette[nindex8]&0xff;
                        nindex8 += 4;
                    }
                // Read the image data (actually indices into the palette)
                // Scan lines are still padded out to even 4-byte
                // boundaries.
                int npad8 = (nsizeimage / nheight) - nwidth;
                //System.out.println("nPad is:"+npad8);

                int ndata8[] = new int [nwidth*nheight];
                byte bdata[] = new byte [(nwidth+npad8)*nheight];
                fs.read (bdata, 0, (nwidth+npad8)*nheight);
                nindex8 = 0;
                for (int j8 = 0; j8 < nheight; j8++)
                    {
                        for (int i8 = 0; i8 < nwidth; i8++)
                            {
                                ndata8 [nwidth*(nheight-j8-1)+i8] =
                                    npalette [((int)bdata[nindex8]&0xff)];
                                nindex8++;
                            }
                        nindex8 += npad8;
                    }

                image = Toolkit.getDefaultToolkit().createImage( new MemoryImageSource (nwidth, nheight,n
            }
        else if (nbitcount == 1) {

            int npad1 = (nsizeimage / nheight) - nwidth/8;
            byte bdata[] = new byte [(nwidth+npad1)*nheight];
```

```
                    fs.read (bdata, 0, 8);
                    fs.read (bdata, 0, (nwidth+npad1)*nheight);
                    int ndata1[] = new int [nwidth*nheight];
                    int nindex1 = 0 ;

                    int max = 0 ;

                    for (int j1 = 0 ; j1 < nheight ; j1++) {
                        int iindex ;
                        iindex = nindex1 ;
                        for (int i1 = 0 ; i1 <= nwidth/8 ; i1++) {
                            int ib1 = 0 ;
                            if (i1*8 < nwidth) {
                                for (int b1 = 128 ; b1 > 0 ; b1 = b1 / 2) {
                                    ndata1 [nwidth*(nheight-j1-1)+i1*8+ib1] = ((b1 & bdata[iindex]) > 0) ? 255+(2
                                    ib1++ ;
                                    if (i1*8+ib1 >= nwidth) {
                                        b1 = 0 ;
                                    }
                                }
                            }
                            max = i1 * 8 + ib1 ;
                            iindex++ ;
                        }
                        nindex1 += (nsizeimage / nheight) ;
                    }

                    image = Toolkit.getDefaultToolkit().createImage( new MemoryImageSource (nwidth, nheight,ndata
                }
                else
                    {
                        System.out.println ("Not a 24-bit or 8-bit or 1-bit Windows Bitmap, aborting...");
                        image = (Image)null;
                    }

                fs.close();
                return image;

            }
        catch (Exception e)
            {
                System.out.println("Caught exception in loadbitmap!");
            }
        return (Image)null;
    }

/**************
 *
 * @param parImage
 * @param parWidth
 * @param parHeight
 * @return
 */
    private static  int[] getPixels(Image parImage, int parWidth, int parHeight) {
        int[] bitmap = new int [parWidth * parHeight];
        PixelGrabber pg = new PixelGrabber (parImage, 0, 0, parWidth, parHeight,
                                            bitmap, 0, parWidth);
        try {
            pg.grabPixels ();
        }
        catch (InterruptedException e) {
            e.printStackTrace ();
        }
        return bitmap;
    }

 /***************
  *
  * @param parFilename
  * @param imagePix
  * @param parWidth
  * @param parHeight
  */
    private void saveBitmap (String parFilename, int[] imagePix, int
            parWidth, int parHeight) {

            try {
                            fo = new FileOutputStream (parFilename);
                            save (imagePix, parWidth, parHeight);
                            fo.close ();
            }
            catch (Exception saveEx) {
                    saveEx.printStackTrace ();
            }

}


/*
* The saveMethod is the main method of the process. This method
* will call the convertImage method to convert the memory image to
* a byte array; method writeBitmapFileHeader creates and writes
* the bitmap file header; writeBitmapInfoHeader creates the
* information header; and writeBitmap writes the image.
*
*/
private void save (int[] imagePix, int parWidth, int parHeight) {

        try {
                convertImage (imagePix, parWidth, parHeight);
                writeBitmapFileHeader ();
                writeBitmapInfoHeader ();
                writeBitmap ();
        }
        catch (Exception saveEx) {
        saveEx.printStackTrace ();
        }
}


/*
* convertImage converts the memory image to the bitmap format (BRG).
* It also computes some information for the bitmap info header.
*
```

```
*/
private boolean convertImage (int[] imagePix, int parWidth, int parHeight) {

        int pad;
        bitmap = imagePix;


        pad = (4 - ((parWidth * 3) % 4)) * parHeight;

        if (4 - ((parWidth * 3) % 4) == 4) pad = 0 ;

        biSizeImage = ((parWidth * parHeight) * 3) + pad;
        bfSize = biSizeImage + BITMAPFILEHEADER_SIZE +
        BITMAPINFOHEADER_SIZE;
        biWidth = parWidth;
        biHeight = parHeight;

        return (true);
}

/*
* writeBitmap converts the image returned from the pixel grabber to
* the format required. Remember: scan lines are inverted in
* a bitmap file!
*
* Each scan line must be padded to an even 4-byte boundary.
*/
private void writeBitmap () {

        int size;
        int value;
        int j;
        int i;
        int rowCount;
        int rowIndex;
        int lastRowIndex;
        int pad;
        int padCount;
        byte rgb [] = new byte [3];


        size = (biWidth * biHeight) - 1;
        pad = 4 - ((biWidth * 3) % 4);

        //The following bug correction will cause the bitmap to be unreadable by
        //GIMP.  It must be there for the bitmap to be readable by most other
        //graphics packages.
        if (pad == 4){ // <==== Bug correction
        pad = 0;
        }// <==== Bug correction


        rowCount = 1;
        padCount = 0;
        rowIndex = size - biWidth;
        lastRowIndex = rowIndex;

```

```
try {
// The following three lines of code are a correction supplied
// by Alin Arsu, Feb 2003.  The original code set the top-right
// pixel in the image to black, and also shifted the bottom row
// of the image by one pixel.
// The original code was the following two lines:
// for (j = 0; j < size; j++) {
//     value = bitmap [rowIndex];
// This is replaced by the three lines that appear next.
for (j = 0; j < size+1; j++) {
if (j<biWidth) { value = bitmap [rowIndex+1]; }
else { value = bitmap [rowIndex]; }

rgb [0] = (byte) (value & 0xFF);
rgb [1] = (byte) ((value >> 8) & 0xFF);
rgb [2] = (byte) ((value >> 16) & 0xFF);
fo.write (rgb);
if (rowCount == biWidth) {
    padCount += pad;
    for (i = 1; i <= pad; i++) {
        fo.write (0x00);
    }
    rowCount = 1;
    rowIndex = lastRowIndex - biWidth;
    lastRowIndex = rowIndex;
}
else
    rowCount++;
rowIndex++;
}

//--- Update the size of the file
bfSize += padCount - pad;
biSizeImage += padCount - pad;
}
catch (Exception wb) {
wb.printStackTrace ();
}

}

/*
* writeBitmapFileHeader writes the bitmap file header to the file.
*
*/
private void writeBitmapFileHeader () {

try {
fo.write (bfType);
fo.write (intToDWord (bfSize));
fo.write (intToWord (bfReserved1));
fo.write (intToWord (bfReserved2));
fo.write (intToDWord (bfOffBits));

}
catch (Exception wbfh) {
wbfh.printStackTrace ();
```

```
        }

        }

        /*
        *
        * writeBitmapInfoHeader writes the bitmap information header
        * to the file.
        *
        */

        private void writeBitmapInfoHeader () {

                try {
                        fo.write (intToDWord (biSize));
                        fo.write (intToDWord (biWidth));
                        fo.write (intToDWord (biHeight));
                        fo.write (intToWord (biPlanes));
                        fo.write (intToWord (biBitCount));
                        fo.write (intToDWord (biCompression));
                        fo.write (intToDWord (biSizeImage));
                        fo.write (intToDWord (biXPelsPerMeter));
                        fo.write (intToDWord (biYPelsPerMeter));
                        fo.write (intToDWord (biClrUsed));
                        fo.write (intToDWord (biClrImportant));
                }
                catch (Exception wbih) {
                        wbih.printStackTrace ();
                }

        }




/****************************
 * Metodo privado que convierte un int a word y devuelve el valor
 * en un array de 2 bytes.
 *
 * Utilizado en writeBitmapInfoHeader y writeBitmapFileHeader
 *
 * @param parValue
 * @return
 */
        private byte [] intToWord (int parValue) {

                byte retValue [] = new byte [2];

                retValue [0] = (byte) (parValue & 0x00FF);
                retValue [1] = (byte) ((parValue >> 8) & 0x00FF);

                return (retValue);

        }


/****************************
```

```
 * Metodo privado que convierte un int a double word y lo
 * devulve en un array de 4 bytes.
 *
 * Utilizado en writeBitmapInfoHeader
 *
 * @param parValue
 * @return
 */

        private byte [] intToDWord (int parValue) {

                byte retValue [] = new byte [4];
                retValue [0] = (byte) (parValue & 0x00FF);
                retValue [1] = (byte) ((parValue >> 8) & 0x000000FF);
                retValue [2] = (byte) ((parValue >> 16) & 0x000000FF);
                retValue [3] = (byte) ((parValue >> 24) & 0x000000FF);

                return (retValue);

        }

}
```

Clase Visual.

```
package Practica4;


import java.awt.*;
import java.awt.event.*;
import java.io.*;

import javax.swing.Box;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;

/*******************************************************
 ********GRAVE(GRAphics Viewer for Everywhere)**********
 *******************************************************
 *Public Class Grave written by Jeb Thorley Aug. 2000
 *Grave is a simple platform independent viewer
 *capable of displaying gif, jpeg and bitmap
 *files.  Viewer requires FileMenuControl.class to control
 *its file menu, and utils.class to open bitmaps.  The
 *BMPHandler Class was originally written
 *by Jeff West and published at
 *http://www.javaworld.com/javaworld/javatips/jw-javatip43.html
 ******************************************************/

public class Visual extends JFrame{
    private JLabel label1;
    private Box layoutBox = Box.createVerticalBox();
    private JFrame resultFrame = new JFrame("GRAVE (GRAphics Viewer for Everywhere)");
```

```
  private Container contentPane = resultFrame.getContentPane();
  private JScrollPane imageScroller;




/*****
  *
  *
  * @param file
  */
 public  void load(String file){


    //initialize label1, and load first image if one is provided.
    ImageIcon ic = new ImageIcon();
    ic = openFile(file);
    label1 = new JLabel(ic);

    //Set resultFrame's size
    resultFrame.setSize(375,500);
    resultFrame.setTitle(file);

    //Make Viewer let go of its resources and relinquish control if
    //its window is closed.
    resultFrame.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          resultFrame.dispose();
          System.exit(0);
        }
      });

    /**Create a Scrollable area in which to display the image**/
    imageScroller = new JScrollPane(label1);
    imageScroller.setPreferredSize(new Dimension(580,380));
    imageScroller.setMinimumSize(new Dimension(580,380));
    /*****************************************************/

    //Add the scrollable are to the layout box
    layoutBox.add(imageScroller);

    //display the layout box
    contentPane.add(layoutBox);

    //show the JFrame
    resultFrame.show();
 }


 /*****************************************************
  *openFile returns an ImageIcon representation of a graphics
  *file.  It has been overloaded, so it may take one, or two
  *input parameters.  If supplied with a single string, the string
  *should be the name of the file to opened.  If supplied with two
  *arguments, they should be the path to the file, and the file
  *name, respectively.
  ****************************************************/
```

```java
public  ImageIcon openFile(String dir, String filename){
      //File f is used only to get the file separator string.  This
      //is done to ensure platform independence.
      File f = new File(filename);
      String s = f.separator;
      if(dir.endsWith(s)!=true){
          dir = dir+s;
      }
      Image i;
      ImageIcon oic = new ImageIcon();
      if(filename.endsWith(".gif")||filename.endsWith(".jpg")||filename.endsWith(".jpeg")){
          i=Toolkit.getDefaultToolkit().getImage(dir+filename);
          oic = new ImageIcon(i);
      }
      else if(filename.endsWith(".bmp")){
          i = BMPHandler.loadbitmap(dir,filename);
          oic = new ImageIcon(i);
      }
      else{
          System.out.println("Unable to open " + filename+".  File must end in bmp, gif, jpg or jpeg.");
          System.exit(1);
      }
      return oic;
 }

  /********************
   *
   *
   * @param filename
   * @return
   */
public static ImageIcon openFile(String filename){
      Image i;
      ImageIcon ic = new ImageIcon();
      if(filename.endsWith(".gif")||filename.endsWith(".jpg")||filename.endsWith(".jpeg")){
          i=Toolkit.getDefaultToolkit().getImage(filename);
          ic = new ImageIcon(i);
      }
      else if(filename.endsWith(".bmp")){
          i = BMPHandler.loadbitmap("./",filename);
          ic = new ImageIcon(i);
      }
      else{
          System.out.println("Unable to open " + filename+".  File must end in bmp, gif, jpg or jpeg.");
          System.exit(1);
      }
      return ic;
 }

  /**************************************************
   *showNew is a public method to update the image displayed
   *in Viewer.  It takes the the ImageIcon that is to replace
   *the current image as a parameter.
   **************************************************/
public void showNew(ImageIcon newIcon){
      //Remove everything from each level of container.
      //This seems to be required to have changes show.
```

```
        contentPane.removeAll();
        layoutBox.removeAll();
        imageScroller.removeAll();

        //Change the image and put everything back together
        label1 = new JLabel(newIcon);
        imageScroller = new JScrollPane(label1);
        layoutBox.add(imageScroller);
        contentPane.add(layoutBox);

        //Show your changes
        resultFrame.repaint();
        resultFrame.show();
    }
}
```

Clase iObserver.

```
package Practica4;

import java.awt.image.ImageObserver;
import java.awt.*;

public class iObserver implements ImageObserver {

    public boolean imageUpdate (Image img, int infoflags,
                     int x, int y, int width, int height) {
        return true;
    }

}
```

Clase CompararImagenes.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Practica4;

/**
 *
 * @author monte
 */
public class CompararImagenes {

      // to change the value of the offset bits.
    final static int RED    = 0;
    final static int GREEN  = 1;
    final static int BLUE   = 2;
    final static int OFFSET = 3;  // ignore offset; use only red, green, blue
    final static int ERRORFILELOAD =-2;
```

```java
    final static int ERRORFILESIZE =-1;

        int[][][] ImageOriginal;
        final int MAXROWSOriginal;
        final int MAXCOLSOriginal;

        int[][][] ImageModificada;
        final int MAXROWSModificada;
        final int MAXCOLSModificada;

/******
 *
 * @param Original
 * @param Modificada
 */
 public    CompararImagenes (String Original, String Modificada){

        ImageOriginal = BMPHandler.loadBMP(Original);
        MAXROWSOriginal = ImageOriginal.length;
        MAXCOLSOriginal = ImageOriginal[0].length;

        ImageModificada = BMPHandler.loadBMP(Modificada);
        MAXROWSModificada = ImageOriginal.length;
        MAXCOLSModificada = ImageOriginal[0].length;

     }

  /**********
   *
   * @return
   */
   public int diferencias2Imagenes3canales (){

        int dif=0;

        if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;

        if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
        // No tienen mismo tamanho

        for (int row=0; row<MAXROWSOriginal; row++) {
                    for (int col=0; col<MAXCOLSOriginal; col++) {
                       if (ImageOriginal[row][col][RED] != ImageModificada[row][col][RED]) dif++;
                       if (ImageOriginal[row][col][GREEN] != ImageModificada[row][col][GREEN])  dif++;
                       if (ImageOriginal[row][col][BLUE] != ImageModificada[row][col][BLUE])      dif++;
                    }  // for col
           }  // for row

        return dif;

    }

   /************
    *
    * @return
    */
public int diferencias2Imagenes4canales (){

        int dif=0;

        if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;

        if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
        // No tienen mismo tamanho

        for (int row=0; row<MAXROWSOriginal; row++) {
                    for (int col=0; col<MAXCOLSOriginal; col++) {
                       if (ImageOriginal[row][col][RED] != ImageModificada[row][col][RED]) dif++;
                       if (ImageOriginal[row][col][GREEN] != ImageModificada[row][col][GREEN])  dif++;
                       if (ImageOriginal[row][col][BLUE] != ImageModificada[row][col][BLUE])      dif++;
                       if (ImageOriginal[row][col][OFFSET] != ImageModificada[row][col][OFFSET])      dif++;
                    }  // for col
           }  // for row

        return dif;
}

/*********
 *
 * @return
 */
public int diferenciasBit2Imagenes3canales (){

        int dif=0;

        if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;

        if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
        // No tienen mismo tamanho

        for (int row=0; row<MAXROWSOriginal; row++) {
           for (int col=0; col<MAXCOLSOriginal; col++) {
                dif+=  Integer.bitCount(ImageOriginal[row][col][RED] ^ ImageModificada[row][col][RED]);
                dif+=  Integer.bitCount(ImageOriginal[row][col][GREEN] ^ ImageModificada[row][col][GREEN]);
                dif+=  Integer.bitCount(ImageOriginal[row][col][BLUE] ^ ImageModificada[row][col][BLUE]);
               }  // for col
           }  // for row

        return dif;
}

/*********
 *
 * @return
 */
public int diferenciasBit2Imagenes4canales (){

        int dif=0;

        if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;

        if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
        // No tienen mismo tamanho

        for (int row=0; row<MAXROWSOriginal; row++) {
           for (int col=0; col<MAXCOLSOriginal; col++) {
                dif+=  Integer.bitCount(ImageOriginal[row][col][RED] ^ ImageModificada[row][col][RED]);
                dif+=  Integer.bitCount(ImageOriginal[row][col][GREEN] ^ ImageModificada[row][col][GREEN]);
                dif+=  Integer.bitCount(ImageOriginal[row][col][BLUE] ^ ImageModificada[row][col][BLUE]);
                dif+=  Integer.bitCount(ImageOriginal[row][col][OFFSET] ^ ImageModificada[row][col][OFFSET]);
               }  // for col
           }  // for row

        return dif;
}

/**************
 *
 * @return
 */

    public int diferencias2Imagenes (){

        int dif=0;
        if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;

        if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
        // No tienen mismo tamanho

        for (int row=0; row<MAXROWSOriginal; row++) {
                    for (int col=0; col<MAXCOLSOriginal; col++) {
                       if (ImageOriginal[row][col][RED] != ImageModificada[row][col][RED]) dif++;
                       else{
                           if(ImageOriginal[row][col][GREEN] != ImageModificada[row][col][GREEN])  dif++;
                           else
                             if (ImageOriginal[row][col][BLUE] != ImageModificada[row][col][BLUE])  dif++;
                       }
                    }  // for col
                }  // for row


        return dif;

    }

}
```

Clase Canal.

```
package Practica4;

import java.util.Random;

/**
 *
 * @author monte
 */
public class Canal {

```

```java

    double probabError=0.01;
    Random rnd = new Random();

   /********
    *
    * @param probabErrorP
    */

    Canal (double probabErrorP){
        probabError=probabErrorP;
    }

    /*********************
     * img es el nombre del fichero bmp
     *
     * @param img
     */
     public String canal (String img){

     }

   /********************
    * Es utilizado por el metodo canal para intrepetar
    * bit a bit el canal
    *
    * @param b4
    * @return
    */

    public  int  canal(int b4) {
      int out=0;

      return out;
    }



}
```