

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*.

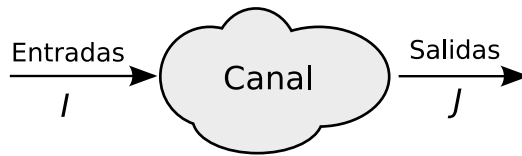


Figura 1: Representación visual de un canal

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

Definición 1 (Canal Binario Simétrico) *Un canal binario simétrico (BSC) corresponde a una 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

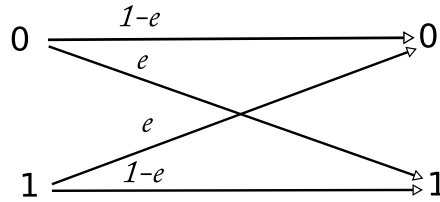


Figura 2: El canal binario simétrico con probabilidad de error de bit e

información que recibe de la librería 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 mostrar 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*.

```

1  private static void compara (String ori,String mod){
2
3      CompararImagenes comparar = new CompararImagenes(ori,mod);
4      int dif=comparar.diferencias2Imagenes();
5      int dif3canales=comparar.diferencias2Imagenes3canales();
6      int dif3canalesBit=comparar.diferenciasBit2Imagenes3canales();
7
8      System.out.println("Diferencia imagenes: "+dif+" Diferencias 3 canales: "+dif3canales);
9      System.out.println("Diferencias 3 canales BIT: "+dif3canalesBit);
10     System.out.println("*****");
11 }
12
13 public static void main(String[] args) {
14
15     String imagenNormal ="android.bmp";
16     Visual normalV = new Visual ();
17     normalV.load(imagenNormal);
18     Canal c = new Canal (0.01);
19     imagenCanal= c.canal(imagenNormal);
20     Visual canalV = new Visual ();
21     canalV.load(imagenCanal);
22     compara(imagenNormal,imagenCanal);
23 }

```

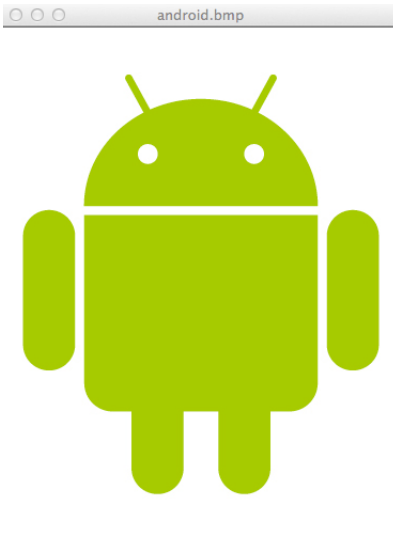


Figura 3: Imagen original utilizada en las prácticas.

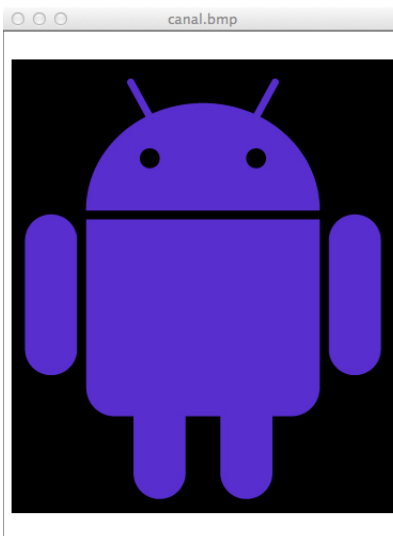


Figura 4: Imagen resultante tras pasar por un canal con tasa de error 1.



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.

```
1 package Practica4;
2
3 import java.awt.*;
4 import java.awt.image.MemoryImageSource;
5 import java.awt.image.PixelGrabber;
6 import java.io.FileInputStream;
7 import java.io.FileOutputStream;
8
9 public class BMPHandler {
10
11     private final static int BITMAPFILEHEADER_SIZE = 14;
12     private final static int BITMAPINFOHEADER_SIZE = 40;
13
14     //--- Bitmap file header
```

```

15     // private byte bitmapFileHeader [] = new byte [14];
16     private byte bfType [] = {(byte)'B', (byte)'M'};
17     private int bfSize = 0;
18     private int bfReserved1 = 0;
19     private int bfReserved2 = 0;
20     private int bfOffBits = BITMAPFILEHEADER_SIZE + BITMAPINFOHEADER_SIZE;
21
22     //--- Bitmap info header
23     //private byte bitmapInfoHeader [] = new byte [40];
24     private int biSize = BITMAPINFOHEADER_SIZE;
25     private int biWidth = 0;
26     private int biHeight = 0;
27     private int biPlanes = 1;
28     private int biBitCount = 24;
29     private int biCompression = 0;
30     private int biSizeImage = 0x030000;
31     private int biXPelsPerMeter = 0x0;
32     private int biYPelsPerMeter = 0x0;
33     private int biClrUsed = 0;
34     private int biClrImportant = 0;
35
36     //--- Bitmap raw data
37     private int bitmap [];
38
39     //--- File section
40     private FileOutputStream fo;
41
42
43     /*****
44     *
45     * @param openName
46     * @return
47     */
48
49     public static int[][][] loadBMP(String openName) {
50         Image img = (Image)null; // create a null Image
51
52
53         img = loadbitmap("./", openName);
54
55         // If the image did not load, print an explanatory
56         // message to the user and ask him/her to try again.
57         if (img == null) {
58             System.out.println("Could not read an image from file "+openName);
59             System.out.println("Make sure that you supply the name of an image file, \nand that you include
60             return null;
61         } // if (img==null)
62
63
64         // Translate from Image img to a 3D array "imagePixels".
65         // Using this 3D array, imagePixels[r][c][w] gives the value
66         // of row r, column c, colour w.
67         int[][][] imagePixels = getImagePixels(img);
68         return imagePixels;
69     } // end of method loadImage
70
71     /*****

```

```

72      *
73      * @param imagePixels
74      * @param saveName
75      */
76
77      public void saveImage(String saveName, int[] [] imagePixels){
78          int height = imagePixels.length;
79          int width = imagePixels[0].length;
80          int[] [] flat = new int[width*height] [4];
81
82
83          // If saveName does not already end in .bmp, then add .bmp to saveName.
84          saveName=bmpTack(saveName);
85
86          // Flatten the image into a 2D array.
87          int index=0;
88          for(int row=0; row<height; row++) {
89              for(int col=0; col<width; col++) {
90                  for(int rgbo=0; rgbo<4; rgbo++) {
91                      flat[index][rgbo]=imagePixels[row][col][rgbo];
92                  }
93                  index++;
94              } // for col
95          } // for row
96
97          // Combine the 8-bit red, green, blue, offset values into 32-bit words.
98          int[] outPixels = new int[flat.length];
99          for(int j=0; j<flat.length; j++) {
100              outPixels[j] = ((flat[j][0]&0xff)<<16) | ((flat[j][1]&0xff)<<8)
101                  | (flat[j][2]&0xff) | ((flat[j][3]&0xff)<<24);
102          } // for j
103
104          // Write the data out to file with the name given by string saveName.
105          saveBitmap(saveName, outPixels, width, height);
106
107      } // end of method saveImage
108
109  /*****
110  *
111  * @param img
112  * @return
113  */
114  private static int[] [] [] getImagePixels(Image img) {
115
116      // Get the raw pixel data
117      iObserver observer = new iObserver();
118      int width1 = img.getWidth(observer);
119      int height1 = img.getHeight(observer);
120      int[] rawPixels = getPixels(img,width1,height1);
121
122      // Each pixel is represented by 32 bits. Separate the th32 bits into
123      // four 8-bit values (red, green, blue, offset).
124      int[] [] rgbPixels = new int[rawPixels.length] [4];
125      for(int j=0; j<rawPixels.length; j++) {
126          rgbPixels[j][0] = ((rawPixels[j]>>16)&0xff);
127          rgbPixels[j][1] = ((rawPixels[j]>>8)&0xff);
128          rgbPixels[j][2] = (rawPixels[j]&0xff);

```

```

129         rgbPixels[j][3] =((rawPixels[j]>>24)&0xff);
130     } // for j
131
132     // Arrange the data by rows and columns
133     int[][][] imagePixels = new int[height1][width1][4];
134     int index=0;
135     for(int row=0; row<imagePixels.length; row++) {
136         for(int col=0; col<imagePixels[0].length; col++) {
137             for(int rgbo=0; rgbo<4; rgbo++) {
138                 imagePixels[row][col][rgbo]=rgbPixels[index][rgbo];
139             } // for
140             index++;
141         } // for col
142     } // for row
143     return imagePixels;
144 } // end of method getImagePixels
145
146
147
148
149
150 /*****
151  *
152  * @param name
153  * @return
154  */
155
156
157     private static String bmpTack(String name) {
158         if (name.endsWith(".bmp"))
159             return name;
160         else
161             return name+".bmp";
162     } // end of method bmpTack
163
164 /*****
165  *
166  * @param sdir
167  * @param sfile
168  * @return
169  */
170     public static Image loadbitmap (String sdir, String sfile){
171
172         Image image;
173
174         try
175         {
176             FileInputStream fs;
177             if(sdir.equals("./")){//a bit of a hack
178                 fs=new FileInputStream(sfile);
179             }
180             else{
181                 fs=new FileInputStream(sdir+sfile);
182             }
183
184             int bflen=14; // 14 byte BITMAPFILEHEADER
185             byte bf[]=new byte[bflen];

```

```

186     fs.read(bf,0,bflen);
187     int bilen=40; // 40-byte BITMAPINFOHEADER
188     byte bi[]=new byte[bilen];
189     fs.read(bi,0,bilen);
190
191     // Interpret data.
192     int nsize = (((int)bf[5]&0xff)<<24)
193         | (((int)bf[4]&0xff)<<16)
194         | (((int)bf[3]&0xff)<<8)
195         | (int)bf[2]&0xff;
196     //System.out.println("File type is :"+(char)bf[0]+(char)bf[1]);
197     //System.out.println("Size of file is :"+nsize);
198
199     int nbisize = (((int)bi[3]&0xff)<<24)
200         | (((int)bi[2]&0xff)<<16)
201         | (((int)bi[1]&0xff)<<8)
202         | (int)bi[0]&0xff;
203     //System.out.println("Size of bitmapinfoheader is :"+nbisize);
204
205     int nwidth = (((int)bi[7]&0xff)<<24)
206         | (((int)bi[6]&0xff)<<16)
207         | (((int)bi[5]&0xff)<<8)
208         | (int)bi[4]&0xff;
209     //System.out.println("Width is :"+nwidth);
210
211     int nheight = (((int)bi[11]&0xff)<<24)
212         | (((int)bi[10]&0xff)<<16)
213         | (((int)bi[9]&0xff)<<8)
214         | (int)bi[8]&0xff;
215     //System.out.println("Height is :"+nheight);
216
217
218     int nplanes = (((int)bi[13]&0xff)<<8) | (int)bi[12]&0xff;
219     //System.out.println("Planes is :"+nplanes);
220
221     int nbitcount = (((int)bi[15]&0xff)<<8) | (int)bi[14]&0xff;
222     //System.out.println("BitCount is :"+nbitcount);
223
224     // Look for non-zero values to indicate compression
225     int ncompression = (((int)bi[19])<<24)
226         | (((int)bi[18])<<16)
227         | (((int)bi[17])<<8)
228         | (int)bi[16];
229     //System.out.println("Compression is :"+ncompression);
230
231     int nsizeimage = (((int)bi[23]&0xff)<<24)
232         | (((int)bi[22]&0xff)<<16)
233         | (((int)bi[21]&0xff)<<8)
234         | (int)bi[20]&0xff;
235     //System.out.println("SizeImage is :"+nsizeimage);
236
237     int nxpm = (((int)bi[27]&0xff)<<24)
238         | (((int)bi[26]&0xff)<<16)
239         | (((int)bi[25]&0xff)<<8)
240         | (int)bi[24]&0xff;
241     //System.out.println("X-Pixels per meter is :"+nxpm);
242

```



```

243     int nypm = (((int)bi[31]&0xff)<<24)
244             | (((int)bi[30]&0xff)<<16)
245             | (((int)bi[29]&0xff)<<8)
246             | (int)bi[28]&0xff;
247     //System.out.println("Y-Pixels per meter is :"+nypm);
248
249     int nclrused = (((int)bi[35]&0xff)<<24)
250                  | (((int)bi[34]&0xff)<<16)
251                  | (((int)bi[33]&0xff)<<8)
252                  | (int)bi[32]&0xff;
253     //System.out.println("Colors used are :"+nclrused);
254
255     int nclrimp = (((int)bi[39]&0xff)<<24)
256                  | (((int)bi[38]&0xff)<<16)
257                  | (((int)bi[37]&0xff)<<8)
258                  | (int)bi[36]&0xff;
259     //System.out.println("Colors important are :"+nclrimp);
260
261     if (nbitcount==24)
262     {
263         // No Palette data for 24-bit format but scan lines are
264         // padded out to even 4-byte boundaries.
265         int npad = (nsizeimage / nheight) - nwidth * 3;
266         //added for Bug correction
267         if(npad == 4){
268             npad=0;
269         }
270         int ndata[] = new int [nheight * nwidth];
271         byte brgb[] = new byte [(nwidth + npad) * 3 * nheight];
272
273         fs.read (brgb, 0, (nwidth + npad) * 3 * nheight);
274         int nindex = 0;
275         for (int j = 0; j < nheight; j++)
276         {
277             for (int i = 0; i < nwidth; i++)
278             {
279                 ndata [nwidth * (nheight - j - 1) + i] =
280
281                     (255&0xff)<<24
282                     | (((int)brgb[nindex+2]&0xff)<<16)
283                     | (((int)brgb[nindex+1]&0xff)<<8)
284                     | (int)brgb[nindex]&0xff;
285                 nindex += 3;
286             }
287             nindex += npad;
288         }
289
290         image = Toolkit.getDefaultToolkit().createImage( new MemoryImageSource (nwidth, nheight,n
291     }
292     else if (nbitcount == 8)
293     {
294         // Have to determine the number of colors, the clrused
295         // parameter is dominant if it is greater than zero. If
296         // zero, calculate colors based on bitsperpixel.
297         int nNumColors = 0;
298         if (nclrused > 0)
299         {

```

```

300         nNumColors = nclrused;
301     }
302     else
303     {
304         nNumColors = (1&0xff)<<nbitcount;
305     }
306     //System.out.println("The number of Colors is"+nNumColors);
307
308     // Some bitmaps do not have the sizeimage field calculated
309     // Ferret out these cases and fix 'em.
310     if (nsizeimage == 0)
311     {
312         nsizeimage = (((nwidth*nbitcount)+31) & ~31 ) >> 3);
313         nsizeimage *= nheight;
314         //System.out.println("nsizeimage (backup) is"+nsizeimage);
315     }
316
317     // Read the palatte colors.
318     int npalette[] = new int [nNumColors];
319     byte bpalette[] = new byte [nNumColors*4];
320     fs.read (bpalette, 0, nNumColors*4);
321     int nindex8 = 0;
322     for (int n = 0; n < nNumColors; n++)
323     {
324         npalette[n] = (255&0xff)<<24
325             | (((int)bpalette[nindex8+2]&0xff)<<16)
326             | (((int)bpalette[nindex8+1]&0xff)<<8)
327             | (int)bpalette[nindex8]&0xff;
328         nindex8 += 4;
329     }
330     // Read the image data (actually indices into the palette)
331     // Scan lines are still padded out to even 4-byte
332     // boundaries.
333     int npad8 = (nsizeimage / nheight) - nwidth;
334     //System.out.println("nPad is:"+npad8);
335
336     int ndata8[] = new int [nwidth*nheight];
337     byte bdata[] = new byte [(nwidth+npad8)*nheight];
338     fs.read (bdata, 0, (nwidth+npad8)*nheight);
339     nindex8 = 0;
340     for (int j8 = 0; j8 < nheight; j8++)
341     {
342         for (int i8 = 0; i8 < nwidth; i8++)
343         {
344             ndata8 [nwidth*(nheight-j8-1)+i8] =
345                 npalette [((int)bdata[nindex8]&0xff)];
346             nindex8++;
347         }
348         nindex8 += npad8;
349     }
350
351     image = Toolkit.getDefaultToolkit().createImage( new MemoryImageSource (nwidth, nheight,n
352 }
353 else if (nbitcount == 1) {
354
355     int npad1 = (nsizeimage / nheight) - nwidth/8;
356     byte bdata[] = new byte [(nwidth+npad1)*nheight];

```

```

357         fs.read (bdata, 0, 8);
358         fs.read (bdata, 0, (nwidth+npad1)*nheight);
359         int ndata1[] = new int [nwidth*nheight];
360         int nindex1 = 0 ;
361
362         int max = 0 ;
363
364         for (int j1 = 0 ; j1 < nheight ; j1++) {
365             int iindex ;
366             iindex = nindex1 ;
367             for (int i1 = 0 ; i1 <= nwidth/8 ; i1++) {
368                 int ib1 = 0 ;
369                 if (i1*8 < nwidth) {
370                     for (int b1 = 128 ; b1 > 0 ; b1 = b1 / 2) {
371                         ndata1 [nwidth*(nheight-j1-1)+i1*8+ib1] = ((b1 & bdata[iindex]) > 0) ? 255+(
372                             ib1++ ;
373                             if (i1*8+ib1 >= nwidth) {
374                                 b1 = 0 ;
375                             }
376                         }
377                     }
378                     max = i1 * 8 + ib1 ;
379                     iindex++ ;
380                 }
381                 nindex1 += (nsizeimage / nheight) ;
382             }
383
384             image = Toolkit.getDefaultToolkit().createImage( new MemoryImageSource (nwidth, nheight,ndata1
385         )
386         else
387         {
388             System.out.println ("Not a 24-bit or 8-bit or 1-bit Windows Bitmap, aborting...");
389             image = (Image)null;
390         }
391
392         fs.close();
393         return image;
394     }
395     catch (Exception e)
396     {
397         System.out.println("Caught exception in loadbitmap!");
398     }
399     return (Image)null;
400 }
401
402
403 /*****
404  *
405  * @param parImage
406  * @param parWidth
407  * @param parHeight
408  * @return
409  */
410 private static int[] getPixels(Image parImage, int parWidth, int parHeight) {
411     int [] bitmap = new int [parWidth * parHeight];
412     PixelGrabber pg = new PixelGrabber (parImage, 0, 0, parWidth, parHeight,
413     bitmap, 0, parWidth);

```

```

414     try {
415         pg.grabPixels ();
416     }
417     catch (InterruptedException e) {
418         e.printStackTrace ();
419     }
420     return bitmap;
421 }
422
423 /*****
424  *
425  * @param parFilename
426  * @param imagePix
427  * @param parWidth
428  * @param parHeight
429  */
430 private void saveBitmap (String parFilename, int[] imagePix, int
431     parWidth, int parHeight) {
432
433     try {
434
435         fo = new FileOutputStream (parFilename);
436         save (imagePix, parWidth, parHeight);
437         fo.close ();
438     }
439     catch (Exception saveEx) {
440         saveEx.printStackTrace ();
441     }
442 }
443
444
445 /*
446 * The saveMethod is the main method of the process. This method
447 * will call the convertImage method to convert the memory image to
448 * a byte array; method writeBitmapFileHeader creates and writes
449 * the bitmap file header; writeBitmapInfoHeader creates the
450 * information header; and writeBitmap writes the image.
451 *
452 */
453 private void save (int[] imagePix, int parWidth, int parHeight) {
454
455     try {
456         convertImage (imagePix, parWidth, parHeight);
457         writeBitmapFileHeader ();
458         writeBitmapInfoHeader ();
459         writeBitmap ();
460     }
461     catch (Exception saveEx) {
462         saveEx.printStackTrace ();
463     }
464 }
465
466
467 /*
468 * convertImage converts the memory image to the bitmap format (BRG).
469 * It also computes some information for the bitmap info header.
470 *

```

```

471 */
472 private boolean convertImage (int[] imagePix, int parWidth, int parHeight) {
473
474     int pad;
475     bitmap = imagePix;
476
477
478     pad = (4 - ((parWidth * 3) % 4)) * parHeight;
479
480     if (4 - ((parWidth * 3) % 4) == 4) pad = 0 ;
481
482     biSizeImage = ((parWidth * parHeight) * 3) + pad;
483     bfSize = biSizeImage + BITMAPFILEHEADER_SIZE +
484     BITMAPINFOHEADER_SIZE;
485     biWidth = parWidth;
486     biHeight = parHeight;
487
488     return (true);
489 }
490
491 /*
492  * writeBitmap converts the image returned from the pixel grabber to
493  * the format required. Remember: scan lines are inverted in
494  * a bitmap file!
495  *
496  * Each scan line must be padded to an even 4-byte boundary.
497  */
498 private void writeBitmap () {
499
500     int size;
501     int value;
502     int j;
503     int i;
504     int rowCount;
505     int rowIndex;
506     int lastRowIndex;
507     int pad;
508     int padCount;
509     byte rgb [] = new byte [3];
510
511
512     size = (biWidth * biHeight) - 1;
513     pad = 4 - ((biWidth * 3) % 4);
514
515     //The following bug correction will cause the bitmap to be unreadable by
516     //GIMP. It must be there for the bitmap to be readable by most other
517     //graphics packages.
518     if (pad == 4){ // <==== Bug correction
519         pad = 0;
520     }// <==== Bug correction
521
522
523     rowCount = 1;
524     padCount = 0;
525     rowIndex = size - biWidth;
526     lastRowIndex = rowIndex;
527

```

```

528     try {
529         // The following three lines of code are a correction supplied
530         // by Alin Arsu, Feb 2003. The original code set the top-right
531         // pixel in the image to black, and also shifted the bottom row
532         // of the image by one pixel.
533         // The original code was the following two lines:
534         // for (j = 0; j < size; j++) {
535         //     value = bitmap [rowIndex];
536         // This is replaced by the three lines that appear next.
537         for (j = 0; j < size+1; j++) {
538             if (j<biWidth) { value = bitmap [rowIndex+1]; }
539             else { value = bitmap [rowIndex]; }
540
541             rgb [0] = (byte) (value & 0xFF);
542             rgb [1] = (byte) ((value >> 8) & 0xFF);
543             rgb [2] = (byte) ((value >> 16) & 0xFF);
544             fo.write (rgb);
545             if (rowCount == biWidth) {
546                 padCount += pad;
547                 for (i = 1; i <= pad; i++) {
548                     fo.write (0x00);
549                 }
550                 rowCount = 1;
551                 rowIndex = lastRowIndex - biWidth;
552                 lastRowIndex = rowIndex;
553             }
554             else
555                 rowCount++;
556             rowIndex++;
557         }
558
559         //--- Update the size of the file
560         bfSize += padCount - pad;
561         biSizeImage += padCount - pad;
562     }
563     catch (Exception wb) {
564         wb.printStackTrace ();
565     }
566
567 }
568
569 /*
570 * writeBitmapFileHeader writes the bitmap file header to the file.
571 *
572 */
573 private void writeBitmapFileHeader () {
574
575     try {
576         fo.write (bfType);
577         fo.write (intToDWord (bfSize));
578         fo.write (intToWord (bfReserved1));
579         fo.write (intToWord (bfReserved2));
580         fo.write (intToDWord (bfOffBits));
581
582     }
583     catch (Exception wbfh) {
584         wbfh.printStackTrace ();

```

```

585     }
586
587     }
588
589     /*
590     *
591     * writeBitmapInfoHeader writes the bitmap information header
592     * to the file.
593     *
594     */
595
596     private void writeBitmapInfoHeader () {
597
598         try {
599             fo.write (intToDWord (biSize));
600             fo.write (intToDWord (biWidth));
601             fo.write (intToDWord (biHeight));
602             fo.write (intToDWord (biPlanes));
603             fo.write (intToDWord (biBitCount));
604             fo.write (intToDWord (biCompression));
605             fo.write (intToDWord (biSizeImage));
606             fo.write (intToDWord (biXPelsPerMeter));
607             fo.write (intToDWord (biYPelsPerMeter));
608             fo.write (intToDWord (biClrUsed));
609             fo.write (intToDWord (biClrImportant));
610         }
611         catch (Exception wbih) {
612             wbih.printStackTrace ();
613         }
614     }
615
616
617
618
619
620     /*****
621     * Metodo privado que convierte un int a word y devuelve el valor
622     * en un array de 2 bytes.
623     *
624     * Utilizado en writeBitmapInfoHeader y writeBitmapFileHeader
625     *
626     * @param parValue
627     * @return
628     */
629     private byte [] intToWord (int parValue) {
630
631         byte retValue [] = new byte [2];
632
633         retValue [0] = (byte) (parValue & 0x00FF);
634         retValue [1] = (byte) ((parValue >> 8) & 0x00FF);
635
636         return (retValue);
637     }
638
639
640
641     /*****

```

```

642 * Metodo privado que convierte un int a double word y lo
643 * devuelve en un array de 4 bytes.
644 *
645 * Utilizado en writeBitmapInfoHeader
646 *
647 * @param parValue
648 * @return
649 */
650
651     private byte [] intToDWord (int parValue) {
652
653         byte retVal [] = new byte [4];
654         retVal [0] = (byte) (parValue & 0x00FF);
655         retVal [1] = (byte) ((parValue >> 8) & 0x000000FF);
656         retVal [2] = (byte) ((parValue >> 16) & 0x000000FF);
657         retVal [3] = (byte) ((parValue >> 24) & 0x000000FF);
658
659         return (retVal);
660
661     }
662
663 }

```

Clase Visual.

```

1 package Practica4;
2
3
4 import java.awt.*;
5 import java.awt.event.*;
6 import java.io.*;
7
8 import javax.swing.Box;
9 import javax.swing.ImageIcon;
10 import javax.swing.JFrame;
11 import javax.swing.JLabel;
12 import javax.swing.JScrollPane;
13
14 /*****GRAVE(Graphics Viewer for Everywhere)*****/
15 *****GRAVE(Graphics Viewer for Everywhere)*****
16 *****GRAVE(Graphics Viewer for Everywhere)*****
17 *Public Class Grave written by Jeb Thorley Aug. 2000
18 *Grave is a simple platform independent viewer
19 *capable of displaying gif, jpeg and bitmap
20 *files. Viewer requires FileMenuControl.class to control
21 *its file menu, and utils.class to open bitmaps. The
22 *BMPHandler Class was originally written
23 *by Jeff West and published at
24 *http://www.javaworld.com/javaworld/javatips/jw-javatip43.html
25 *****/
26
27 public class Visual extends JFrame{
28     private JLabel labell1;
29     private Box layoutBox = Box.createVerticalBox();
30     private JFrame resultFrame = new JFrame("GRAVE (GRAphics Viewer for Everywhere)");

```



```

31     private Container contentPane = resultFrame.getContentPane();
32     private JScrollPane imageScroller;
33
34
35
36
37     /*****
38         *
39         *
40         * @param file
41         */
42     public void load(String file){
43
44
45         //initialize label1, and load first image if one is provided.
46         ImageIcon ic = new ImageIcon();
47         ic = openFile(file);
48         label1 = new JLabel(ic);
49
50         //Set resultFrame's size
51         resultFrame.setSize(375,500);
52         resultFrame.setTitle(file);
53
54         //Make Viewer let go of its resources and relinquish control if
55         //its window is closed.
56         resultFrame.addWindowListener(new WindowAdapter(){
57             public void windowClosing(WindowEvent e){
58                 resultFrame.dispose();
59                 System.exit(0);
60             }
61         });
62
63         /**Create a Scrollable area in which to display the image**/
64         imageScroller = new JScrollPane(label1);
65         imageScroller.setPreferredSize(new Dimension(580,380));
66         imageScroller.setMinimumSize(new Dimension(580,380));
67         *****
68
69         //Add the scrollable are to the layout box
70         layoutBox.add(imageScroller);
71
72         //display the layout box
73         contentPane.add(layoutBox);
74
75         //show the JFrame
76         resultFrame.show();
77     }
78
79
80     *****
81     *openFile returns an ImageIcon representation of a graphics
82     *file. It has been overloaded, so it may take one, or two
83     *input parameters. If supplied with a single string, the string
84     *should be the name of the file to opened. If supplied with two
85     *arguments, they should be the path to the file, and the file
86     *name, respectively.
87     *****

```

```

88 public ImageIcon openFile(String dir, String filename){
89     //File f is used only to get the file separator string. This
90     //is done to ensure platform independence.
91     File f = new File(filename);
92     String s = f.separator;
93     if(dir.endsWith(s)!=true){
94         dir = dir+s;
95     }
96     Image i;
97     ImageIcon oic = new ImageIcon();
98     if(filename.endsWith(".gif")||filename.endsWith(".jpg")||filename.endsWith(".jpeg")){
99         i=Toolkit.getDefaultToolkit().getImage(dir+filename);
100         oic = new ImageIcon(i);
101     }
102     else if(filename.endsWith(".bmp")){
103         i = BMPHandler.loadbitmap(dir,filename);
104         oic = new ImageIcon(i);
105     }
106     else{
107         System.out.println("Unable to open " + filename+". File must end in bmp, gif, jpg or jpeg.");
108         System.exit(1);
109     }
110     return oic;
111 }
112
113 /*****
114  *
115  *
116  * @param filename
117  * @return
118  */
119 public static ImageIcon openFile(String filename){
120     Image i;
121     ImageIcon ic = new ImageIcon();
122     if(filename.endsWith(".gif")||filename.endsWith(".jpg")||filename.endsWith(".jpeg")){
123         i=Toolkit.getDefaultToolkit().getImage(filename);
124         ic = new ImageIcon(i);
125     }
126     else if(filename.endsWith(".bmp")){
127         i = BMPHandler.loadbitmap("./",filename);
128         ic = new ImageIcon(i);
129     }
130     else{
131         System.out.println("Unable to open " + filename+". File must end in bmp, gif, jpg or jpeg.");
132         System.exit(1);
133     }
134     return ic;
135 }
136
137 /*****
138  *showNew is a public method to update the image displayed
139  *in Viewer. It takes the the ImageIcon that is to replace
140  *the current image as a parameter.
141  *****/
142 public void showNew(ImageIcon newIcon){
143     //Remove everything from each level of container.
144     //This seems to be required to have changes show.

```

```

145     contentPane.removeAll();
146     layoutBox.removeAll();
147     imageScroller.removeAll();
148
149     //Change the image and put everything back together
150     label1 = new JLabel(newIcon);
151     imageScroller = new JScrollPane(label1);
152     layoutBox.add(imageScroller);
153     contentPane.add(layoutBox);
154
155     //Show your changes
156     resultFrame.repaint();
157     resultFrame.show();
158 }
159 }

```

Class iObserver.

```

1 package Practica4;
2
3 import java.awt.image.ImageObserver;
4 import java.awt.*;
5
6 public class iObserver implements ImageObserver {
7
8     public boolean imageUpdate (Image img, int infoflags,
9         int x, int y, int width, int height) {
10         return true;
11     }
12
13 }

```

Class CompararImagenes.

```

1 /*
2  * To change this template, choose Tools | Templates
3  * and open the template in the editor.
4  */
5
6 package Practica4;
7
8 /**
9  *
10 * @author monte
11 */
12 public class CompararImagenes {
13
14     // to change the value of the offset bits.
15     final static int RED     = 0;
16     final static int GREEN   = 1;
17     final static int BLUE    = 2;
18     final static int OFFSET  = 3; // ignore offset; use only red, green, blue
19     final static int ERRORFILELOAD =-2;

```

```

20     final static int ERRORFILESIZE =-1;
21
22     int [] [] [] ImageOriginal;
23     final int MAXROWSOriginal;
24     final int MAXCOLSOriginal;
25
26     int [] [] [] ImageModificada;
27     final int MAXROWSModificada;
28     final int MAXCOLSModificada;
29
30     /*****
31     *
32     * @param Original
33     * @param Modificada
34     */
35     public CompararImagenes (String Original, String Modificada){
36
37         ImageOriginal = BMPHandler.loadBMP(Original);
38         MAXROWSOriginal = ImageOriginal.length;
39         MAXCOLSOriginal = ImageOriginal[0].length;
40
41         ImageModificada = BMPHandler.loadBMP(Modificada);
42         MAXROWSModificada = ImageOriginal.length;
43         MAXCOLSModificada = ImageOriginal[0].length;
44
45     }
46
47     /*****
48     *
49     * @return
50     */
51     public int diferencias2Imagenes3canales (){
52
53         int dif=0;
54
55         if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;
56
57         if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
58         // No tienen mismo tamaño
59
60         for (int row=0; row<MAXROWSOriginal; row++) {
61             for (int col=0; col<MAXCOLSOriginal; col++) {
62                 if (ImageOriginal[row][col][RED] != ImageModificada[row][col][RED]) dif++;
63                 if (ImageOriginal[row][col][GREEN] != ImageModificada[row][col][GREEN]) dif++;
64                 if (ImageOriginal[row][col][BLUE] != ImageModificada[row][col][BLUE]) dif++;
65             } // for col
66         } // for row
67
68         return dif;
69
70     }
71
72     /*****
73     *
74     * @return
75     */
76     public int diferencias2Imagenes4canales (){

```

```

77
78     int dif=0;
79
80     if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;
81
82     if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
83     // No tienen mismo tamanho
84
85     for (int row=0; row<MAXROWSOriginal; row++) {
86         for (int col=0; col<MAXCOLSOriginal; col++) {
87             if (ImageOriginal[row][col][RED] != ImageModificada[row][col][RED]) dif++;
88             if (ImageOriginal[row][col][GREEN] != ImageModificada[row][col][GREEN]) dif++;
89             if (ImageOriginal[row][col][BLUE] != ImageModificada[row][col][BLUE]) dif++;
90             if (ImageOriginal[row][col][OFFSET] != ImageModificada[row][col][OFFSET]) dif++;
91         } // for col
92     } // for row
93
94     return dif;
95 }
96
97 /*****
98  *
99  * @return
100  */
101 public int diferenciasBit2Imagenes3canales (){
102
103     int dif=0;
104
105     if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;
106
107     if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
108     // No tienen mismo tamanho
109
110     for (int row=0; row<MAXROWSOriginal; row++) {
111         for (int col=0; col<MAXCOLSOriginal; col++) {
112             dif+= Integer.bitCount(ImageOriginal[row][col][RED] ^ ImageModificada[row][col][RED]);
113             dif+= Integer.bitCount(ImageOriginal[row][col][GREEN] ^ ImageModificada[row][col][GREEN]);
114             dif+= Integer.bitCount(ImageOriginal[row][col][BLUE] ^ ImageModificada[row][col][BLUE]);
115         } // for col
116     } // for row
117
118     return dif;
119 }
120
121 /*****
122  *
123  * @return
124  */
125 public int diferenciasBit2Imagenes4canales (){
126
127     int dif=0;
128
129     if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;
130
131     if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
132     // No tienen mismo tamanho
133

```

```

134     for (int row=0; row<MAXROWSOriginal; row++) {
135         for (int col=0; col<MAXCOLSOriginal; col++) {
136             dif+= Integer.bitCount(ImageOriginal[row][col][RED] ^ ImageModificada[row][col][RED]);
137             dif+= Integer.bitCount(ImageOriginal[row][col][GREEN] ^ ImageModificada[row][col][GREEN]);
138             dif+= Integer.bitCount(ImageOriginal[row][col][BLUE] ^ ImageModificada[row][col][BLUE]);
139             dif+= Integer.bitCount(ImageOriginal[row][col][OFFSET] ^ ImageModificada[row][col][OFFSET]);
140         } // for col
141     } // for row
142
143     return dif;
144 }
145
146 /*****
147  *
148  * @return
149  */
150
151     public int diferencias2Imagenes (){
152
153         int dif=0;
154         if ((ImageOriginal==null) || (ImageModificada ==null)) return ERRORFILELOAD;
155
156         if ((MAXROWSOriginal!=MAXROWSModificada) || (MAXCOLSOriginal !=MAXCOLSModificada)) return ERRORFILESIZE;
157         // No tienen mismo tamaño
158
159         for (int row=0; row<MAXROWSOriginal; row++) {
160             for (int col=0; col<MAXCOLSOriginal; col++) {
161                 if (ImageOriginal[row][col][RED] != ImageModificada[row][col][RED]) dif++;
162                 else{
163                     if(ImageOriginal[row][col][GREEN] != ImageModificada[row][col][GREEN]) dif++;
164                     else
165                         if (ImageOriginal[row][col][BLUE] != ImageModificada[row][col][BLUE]) dif++;
166                 }
167             } // for col
168         } // for row
169
170
171         return dif;
172     }
173 }
174
175 }

```

Clase Canal.

```

1 package Practica4;
2
3 import java.util.Random;
4
5 /**
6  *
7  * @author monte
8  */
9 public class Canal {
10

```

```

11
12     double probabError=0.01;
13     Random rnd = new Random();
14
15     /*****
16     *
17     * @param probabErrorP
18     */
19
20     Canal (double probabErrorP){
21         probabError=probabErrorP;
22     }
23
24     /*****
25     * img es el nombre del fichero bmp
26     *
27     * @param img
28     */
29     public String canal (String img){
30
31     }
32
33     /*****
34     * Es utilizado por el metodo canal para intrepetar
35     * bit a bit el canal
36     *
37     * @param b4
38     * @return
39     */
40
41     public int canal(int b4) {
42         int out=0;
43
44         return out;
45     }
46
47
48
49 }

```
