Objetivos Entrada/Salida. Serialización de objetos.

Índice

1. Parte 1: Entrada / Salida 1

2. Parte 2: Serialización de objetos 5

1. Parte 1: Entrada / Salida

Se pide realizar una serie de clases para una aplicación de procesado (muy trivial) de imágenes.

En el caso que nos ocupa, trabajaremos con una imagen en escala de grises y por lo tanto la imagen estará representada mediante un array de datos con valores enteros en el rango $\{0, 1, \cdots, 254, 255\}$. El tamaño de la imagen es de 512×512 pixels.

La figura 1 muestra un aumento de una imagen en la que se ha superpuesto el valor del nivel de gris.

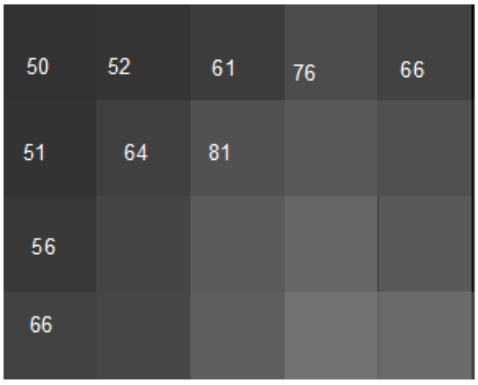
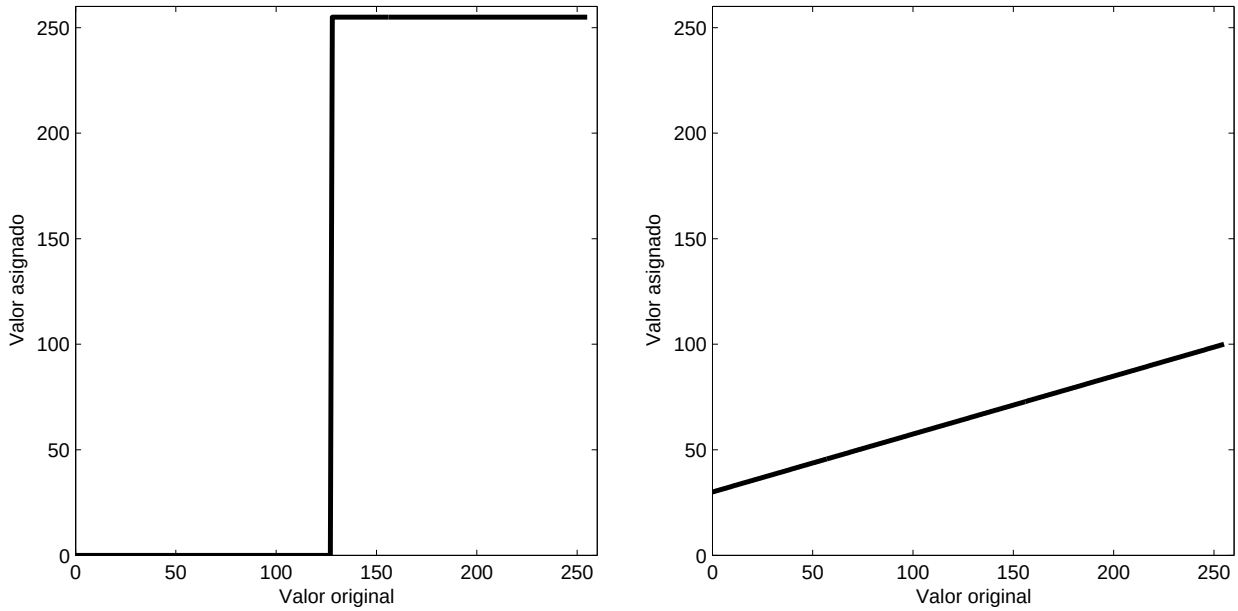


Figura 1: Aumento de una parte de una imagen en la que se pueden ver los pixels y sus niveles de gris

El procesado que se va a realizar consiste en la sustitución de los valores de nivel de gris de la imagen por otros según una determinada tabla. La figura 1 muestra dos ejemplos de sustitución de

valores.



a) b)

Figura 2: Ejemplo de tablas de correspondencia: a) Umbral a 128 y b) Escala lineal entre 30 y 100

Se proporciona una clase `TablasCorrespondencia` que define 2 métodos estáticos que devuelven tablas de transformación:

1. `public static int[] tablaUmbral(int corte)` Devuelve una tabla para realizar una umbralización en un determinado nivel (todos los valores por debajo del valor pasado como argumento se ponen a 0, correspondiente al negro, y todos los valores por encima se ponen a 255, correspondiente al blanco).
2. `public static int[] tablaEscala(int min,int max)` Devuelve una tabla para realizar un cambio de escala. Al valor 0 le hace corresponder *min* y valor 255 le hace corresponder `max`.

El código correspondiente a esta clase es:

```
/** Clase TablasCorrespondencia
 * @author Lenguajes de Programacion
 */
public class TablasCorrespondencia {

   /** Devuelve una tabla para umbralizar los datos
    * @param corte valor de umbralizacion
    * @return la tabla como un vector de enteros
    */
   public static int[] tablaUmbral(int corte) {
      int[] b = new int[256];
      for (int i = 0; i < 256; i++)
         if (i <= corte)
            b[i] = 0;
         else
            b[i] = 255;
      return b;
```

```
    }

    /** Devuelve una tabla para realizar un escalado
     * de los datos
     * @param min valor al que corresponde el 0
     * @param max valor al que corresponde el 255
     * @return la tabla como un vector de enteros
     */
    public static int[] tablaEscala(int min, int max) {
        int[] b = new int[256];
        for (int i = 0; i < 255; i++)
            b[i] = (((max - min) * i) / 255 + min);
        return b;
    }
}
```

Otra de las clases que se proporciona es la clase `MuestraImagen` que representa a una ventana con una serie de componentes donde se visualizará la imagen (esta clase sirve como ejemplo de AWT y como ejemplo de cómo se tratan eventos mediante clases anónimas).

```
import java.awt.Frame;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.MemoryImageSource;

/** Clase que representa a una ventana.
 * @author Lenguajes de Programacion
 */
public class MuestraImagen extends Frame {
    Image img;
    private TextField c;
    private Button b;
    private Lienzo li;

    /** Clase que extiende a Panel
     * Se sobre-escribe el metodo paint de Panel
     * para que muestre la imagen
     */
    class Lienzo extends Panel{
        Lienzo(){
            super();
        }
        public void paint(Graphics g){
            if (img!=null)
                g.drawImage(img,0,0,this);
        }
    }

    /** Constructor
     */
    public MuestraImagen(){

        Panel p = new Panel();
        p.setLayout(new FlowLayout());
        c = new TextField("                ");
        p.add(new Label("Fichero: "));
        p.add(c);

        b = new Button("Mostrar");
        p.add(b);

        add(p,BorderLayout.NORTH);
```

```
        li = new Lienzo();
        add(li,BorderLayout.CENTER);

        /* Se registra un objeto que extiende a MouseAdapter
           para tratar los eventos de raton en el boton.
           Lo que se hace es cargar y mostrar la imagen
           que se ha indicado en el campo de texto enviando
           un mensaje a repaint de Lienzo que encola un evento
           de pintado que llamara a paint de Lienzo (el metodo
           que se ha sobre-escrito */

        b.addMouseListener(new MouseAdapter(){
           public void mousePressed(MouseEvent e){
              try{
                 InputStream is = new BufferedInputStream(new FileInputStream(c.getText()));
                 int dato;
                 int cont=0;
                 int[] imagen = new int[512*512];

                 while ((dato=is.read())!=-1){
                    int r = dato & 0xff;
                    int g = dato & 0xff;
                    int b = dato & 0xff;
                    imagen[cont] = (255 << 24) | (r << 16) | (g << 8) | b;
                    cont++;
                 }
                 is.close();
                 img = createImage(new MemoryImageSource(512,512,imagen,0,512));
                 li.repaint();

              }catch(IOException ex){
                 System.out.println("Error de lectura del fichero");
                 ex.printStackTrace();
              }

           }
        });

        /* Se registra un objeto que extiende a WindowAdapter
           para tratar los eventos sobre la ventana.
           Cuando se pulse para cerrar la ventana se
           finaliza la aplicacion */

        addWindowListener(new WindowAdapter(){
           public void windowClosing(WindowEvent e){
              System.exit(0);
           }
        });

        setSize(700,700);
        show();

     }
}
```

Finalmente se proporciona la clase `ProcesaImagen` que contiene el `main(.)`.

Para esta parte de la práctica se deben realizar dos clases:

1. Una clase `FilterOutputImagen` que extienda a `FilterOutputStream` de forma que esta clase se pueda encadenar siguiendo el patrón decorador. Esta clase se encarga de almacenar en un fichero la imagen transformada según la tabla que se pase en el momento de su creación (ver el `main()`). Como se puede observar en el `main()` las operaciones de escritura se delegarán en el `BufferedOuputStream`. La ventaja de esta clase no es sólo ésta, sino que puesto que es un

`OutputStream` se podría por ejemplo pasar a un objeto del tipo `ZipOutputStream` creando de una vez un fichero comprimido.

> Esta clase `FilterOutputImagen` es similar a la clase `MayusculasWriter` que se comentó en teoría.

2. Una clase `OutputImagen` Esta clase se encarga también de almacenar en un fichero la imagen transformada según la tabla de transformación que le se pase en el momento de su creación (ver el `main()`). En este caso no se utilizará ninguna clase con *buffer*. El esqueleto de esta clase es:

```
public class OutputImagen {

    private OutputStream os;
    private int[] lookupTable;

    public OutputImagen(OutputStream o,int[] table){}

    public void guarda(byte b) throws IOException{}

    public void cierra() throws IOException{}
}
```

En el `main(.)` se mide el tiempo que se tarda en realizar la escritura utilizando cada una de estas clases.

2. Parte 2: Serialización de objetos

En la segunda parte de la práctica se propone trabajar con la serialización de objetos.

Se dispone de la siguiente clase:

```
import java.math.BigDecimal;

public class Comprador {
   private String login;
   private String password;
   private boolean permiteConexion = false;
   private BigDecimal cesta = new BigDecimal(0);

   public Comprador(String usuario){
      login = usuario;
      password = "CMPRDR" + login;
   }

   public void conecta(String pass){
      if (password.compareTo(pass)==0){
         permiteConexion = true;
         System.out.println("Conectado");
      }
      else
         permiteConexion = false;
   }
```

```
    public void sumaCesta(double c) throws Exception{
        if (!permiteConexion)
            throw new Exception();
        else
            cesta = cesta.add(new BigDecimal(c));
    }

    public double consultaCesta() throws Exception{
        if (!permiteConexion)
            throw new Exception();
        else
            return cesta.doubleValue();
    }
}
```

Modificarla para poder serializar objetos de este tipo. No se debe almacenar el `password` durante la serialización.

Realizar otra clase que contenga el `main(.)` en la que se hagan este tipo de operaciones con un objeto de la clase anterior:

1. Crear un objeto del tipo `Comprador`.
2. Conectar.
3. Sumar a la cesta una determinada cantidad.
4. Mostrar el contenido de la cesta.
5. Serializar el objeto.
6. Leer el objeto serializado.
7. Conectar.
8. Sumar a la cesta una determinada cantidad.
9. Mostrar el contenido de la cesta (que deberá tener la suma de las dos cantidades).

Un requisito que se debe cumplir siempre es que no se puede realizar ninguna operación si no se ha realizado una conexión esto se debe controlar desde la clase `Comprador`.