



Manual técnico

Práctica 2 IPC1

Damian Ignacio Peña Afre
202110568

Contenido

Paquete lógica..... 2

 Clase DatosGrafica 2

 Clase DatosFila 4

 Clase Cronometro 5

 Clase InformacionOrdenamiento..... 7

 Clase Ordenamiento 8

 Clase Reporte 13

Paquete Vista 17

 Frame Principal 17

 Frame simulación 21

Paquete lógica

Clase DatosGrafica

Logica;

```
import org.jfree.data.category.CategoryDataset;  
import org.jfree.data.category.DefaultCategoryDataset;
```

Inclusión de JFreeChart

```
/**  
 *  
 * @author damia  
 */
```

```
public class DatosGrafica {
```

```
    public DatosGrafica() {  
  
    }
```

Constructor vacío

```
    private String datosSinFormato;  
    public String tituloNumeracion;  
    public String tituloBarras;  
    public DatosFila[] datosFilas;  
    private int tipo;
```

Propiedades esenciales
para el gráfico

```
    public DatosGrafica(String datos, int ti) {  
        datosSinFormato = datos;  
        tipo = ti;  
        formatearDatos();  
    }
```

Recibe una String con los datos del csv
separados por “,” y un salto de línea,
llama a un método que formatea los
datos convirtiéndolos en un arreglo de un
objeto DatosFila

```
    private void formatearDatos() {  
        String[] arregloAux1 = datosSinFormato.split("\n");  
  
        datosFilas = new DatosFila[arregloAux1.length - 1];
```

Crea un arreglo apartir
de la String sin
formatear, separando
por salto de linea

```
        int contadorAux = 0;  
        boolean encontrado = false;  
        for (String fila : arregloAux1) {
```

Establece un estado inicial para
verificar si se encontraron los
encabezados

```
            String[] arregloAux2 = fila.split(",");
```

Crea un arreglo separando los datos
de una String con “,”

```
            if (!encontrado) {
```

```
                //encabezados  
                tituloBarras = arregloAux2[0].trim();  
                tituloNumeracion = arregloAux2[1].trim();
```

Guarda el título que tendrán las
barras y la numeración del gráfico,
estableciendo que ya se encontraron
los encabezados

```
                encontrado = true;
```

```

    } else {

        String tema = arregloAux2[0].trim();
        int nota = Integer.parseInt(arregloAux2[1].trim());

        // clase - valor
        datosFilas[contadorAux] = new DatosFila(tema, nota);
        contadorAux++;
    }
}

}

public static CategoryDataset crearDataSet(DatosGrafica datos) {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();

    String descripcion = "";

    if (datos.tipo==0) {
        descripcion = "Ascendente";
    }else{
        descripcion = "Descendente";
    }

    for (DatosFila datoColumna : datos.datosFilas) {
        dataset.addValue(datoColumna.getNota(),
            datoColumna.getTitulo(), descripcion);
    }

    return dataset;
}
}

```

Crea un objeto tipo DatosFila a partir del resto de filas del csv

Recibe un objeto tipo DatosGrafica y regresa un conjunto de datos para generar la gráfica.

Determina el tipo de ordenamiento

Por cada fila dentro de este objeto es añadido al conjunto de datos

```
package Logica;

/**
 *
 * @author damia
 */

public class DatosFila{

    public String titulo;
    public int nota;

    public DatosFila(){}

    public DatosFila(String titulo, int nota) {
        this.titulo = titulo;
        this.nota = nota;
    }

    public String getTitulo() {
        return titulo;
    }

    public int getNota() {
        return nota;
    }

    public String getNotaAsString() {
        return nota+"";
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public void setNota(int nota) {
        this.nota = nota;
    }

}
```

Propiedades de la fila del .csv

Constructores, uno de ellos define las propiedades iniciales.

Getters y Setters

Clase Cronometro

```
package Logica;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JLabel;

/**
 *
 * @author damia
 */
public class Cronometro implements Runnable {

    private Thread tCronometro;

    public static int milisegundos, segundos, minutos;

    JLabel label;

    public Cronometro(JLabel lbl) {
        label = lbl;
        tCronometro = new Thread(this);
        tCronometro.start();
        milisegundos = 0;
        segundos = 0;
        minutos = 0;
    }

    private String formatearTiempo() {
        String fecha = "";
        milisegundos+=2;
        if (milisegundos > 999) {
            milisegundos = 0;
            segundos++;
        }

        if (segundos > 59) {
            segundos = 0;
            minutos++;
        }

        fecha = minutos + ":" + segundos + ":" + milisegundos;

        return fecha;
    }

    public void detener(){
```

Implementa la interfaz runnable

Hilo del cronometro

Propiedades del cronometro

Recibe un elemento tipo JLabel donde se establecerá el tiempo de ejecución y reinicia

Le da un formato al tiempo y gestiona el incremento de cada contador

```

        tCronometro.interrupt();
    }

    @Override
    public void run() {
        try {
            while (!tCronometro.isInterrupted()){
                label.setText(formatearTiempo());
                Thread.sleep(1);
            }
        } catch (InterruptedException ex) {
        }
    }
}

```

Le da un formato al tiempo y gestiona el incremento de cada contador

```
package Logica;

/**
 *
 * @author damia
 */
public class InformacionOrdenamiento {
    public String algoritmo;
    public String velocidad;
    public String tipo;
    public int duracion;

    public InformacionOrdenamiento(int algoritmo, int velocidad, int tipo) {
        //Informacion de la ejecucion
        if (algoritmo == 0) {
            this.algoritmo = "Burbuja";
        } else if (algoritmo == 1) {
            this.algoritmo = "Seleccion";
        } else if (algoritmo == 2) {
            this.algoritmo = "Insercion";
        }

        if (tipo == 0) {
            this.tipo = "Ascedente";
        } else if (tipo == 1) {
            this.tipo = "Descentende";
        }

        if (velocidad == 0) {
            this.velocidad = "Rapida";
            duracion = 100;
        } else if (velocidad == 1) {
            this.velocidad = "Media";
            duracion = 700;
        } else if (velocidad == 2) {
            this.velocidad = "Lenta";
            duracion = 1500;
        }
    }
}
```

Propiedades del ordenamiento

Formateo de las propiedades


```

public class Ordenamiento implements Runnable {

    //Informacion del csv
    public DatosGrafica datos;

    //Informacion de la simulacion
    public InformacionOrdenamiento info;
    public int tipo, velocidad, algoritmo;
    public String tituloGrafica;
    public int pasos = 0;
    //tiempo
    int minutos, segundos, milisegundos;

    //Thread del ordenamiento
    private Thread tOrdenamiento;

    //Archivo de imagen
    public File imagenGrafica;
    JFreeChart graficaBarrasFinal;

    //Elementos graficos de la simulacion
    public JPanel panelContenedorGrafica;
    public JLabel labelPasos;

    public Ordenamiento(DatosGrafica datos, int algoritmo, int tipo, int velocidad, String
    titulo) {
        this.datos = datos;
        this.tipo = tipo;
        this.algoritmo = algoritmo;
        this.tituloGrafica = (titulo.equals("")) ? "Grafica" : titulo;

        info = new InformacionOrdenamiento(algoritmo, velocidad, tipo);

    }

    public void empezarSimulacion(JPanel panel, JLabel label) {
        panelContenedorGrafica = panel;
        labelPasos = label;
        tOrdenamiento = new Thread(this);
        tOrdenamiento.start();
    }
}

```

Propiedades que guardan información del ordenamiento

Hilo de la simulación y Objetos para manejar la grafica y su imagen.

Elementos gráficos a modificar

Inicialización de propiedades

Formatea el tipo de ordenamiento

Inicializa los objetos gráficos y empieza la ejecución del hilo

```

@Override
public void run() {

    boolean ordenado = false;

    while (!tOrdenamiento.isInterrupted()) {

        while (!ordenado) {

            if (algoritmo == 0) {
                //Burbuja
                ordenado = burbuja(tipo);
            } else if (algoritmo == 1) {
                //seleccion
                ordenado = seleccion(tipo);
            } else if (algoritmo == 2) {
                //insercion
                ordenado = insercion(tipo);
            }
        }

        Simulacion.cronometro.detener();
        detener();

        generarImagen();

        milisegundos = Cronometro.milisegundos;
        segundos = Cronometro.segundos;
        minutos = Cronometro.minutos;

        Reporte reporte = new Reporte(this);

        JOptionPane.showMessageDialog
(null, "Ordenamiento finalizado, generando reporte...");

    }

    private void generarImagen() {
        imagenGrafica =
new File("E:\\NetBeansProjects\\Practica2\\imagenesGeneradas\\" +
tituloGrafica + "Final.png");
        try {
            ChartUtils.
saveChartAsJPEG(imagenGrafica, graficaBarrasFinal, 600, 600);
        } catch (IOException ex) {
        }
    }
}

```

Selección del tipo de algoritmo

Detención del hilo

Captura los datos del cronometro

Generación de reporte

Generación de imagen

```

private void detener() {
    JFreeChart graficoBarras = ChartFactory.createBarChart(tituloGrafica,
datosGrafica.tituloBarras, datosGrafica.tituloNumeracion,
DatosGrafica.crearDataSet(datosGrafica), PlotOrientation.VERTICAL, true, true, false);
    ChartPanel panelGrafica = new ChartPanel(graficoBarras);
    panelGrafica.setPreferredSize(new Dimension(panelContenedorGrafica.getWidth(),
panelContenedorGrafica.getHeight()));

    graficaBarrasFinal = graficoBarras;

    panelContenedorGrafica.setLayout(new BorderLayout());
    panelContenedorGrafica.add(panelGrafica, BorderLayout.CENTER);
    panelContenedorGrafica.validate();

    tOrdenamiento.interrupt();
}

private void actualizarPanel() {
    JFreeChart graficoBarras = ChartFactory.createBarChart(tituloGrafica,
datosGrafica.tituloBarras, datosGrafica.tituloNumeracion,
DatosGrafica.crearDataSet(datosGrafica), PlotOrientation.VERTICAL, true, true, false);
    ChartPanel panelGrafica = new ChartPanel(graficoBarras);
    panelGrafica.setPreferredSize(new Dimension(panelContenedorGrafica.getWidth(),
panelContenedorGrafica.getHeight()));

    panelContenedorGrafica.setLayout(new BorderLayout());
    panelContenedorGrafica.add(panelGrafica, BorderLayout.CENTER);
    panelContenedorGrafica.validate();

    pasos++;
    labelPasos.setText("" + pasos);

    try {
        Thread.sleep(info.duracion);
    } catch (InterruptedException ex) {
    }
}

private boolean burbuja(int tipo) {
    for (int x = 0; x < datos.datosFilas.length; x++) {
        for (int y = 0; y < datos.datosFilas.length - 1; y++) {
            int numeroActual = datos.datosFilas[y].getNota();
            int numeroSiguiente = datos.datosFilas[y + 1].getNota();

            boolean clausula;

            if (tipo == 0) {

```

Detención del hilo y generación del ultimo estado de la grafica

Actualización de la grafica

Actualización de la grafica

Ordenamiento burbuja

```

        clausula = numeroActual > numeroSiguiete;
    } else {
        clausula = numeroActual < numeroSiguiete;

        if (clausula) {
            // Intercambiar
            DatosFila aux = datos.datosFilas[y];
            datos.datosFilas[y] = datos.datosFilas[y + 1];
            datos.datosFilas[y + 1] = aux;
        }

        actualizarPanel();
    }
}
return true;
}

private boolean seleccion(int tipo) {
    for (int i = 0; i < datos.datosFilas.length - 1; i++) {
        for (int j = i + 1; j < datos.datosFilas.length; j++) {
            boolean clausula;

            if (tipo == 0) {
                clausula = datos.datosFilas[i].getNota() > datos.datosFilas[j].getNota();
            } else {
                clausula = datos.datosFilas[i].getNota() < datos.datosFilas[j].getNota();
            }

            if (clausula) {
                DatosFila temporal = datos.datosFilas[i];
                datos.datosFilas[i] = datos.datosFilas[j];
                datos.datosFilas[j] = temporal;
            }

            actualizarPanel();
        }
    }

    return true;
}

```

Cambio según el tipo de ordenamiento (ascendente o descendente)

Algoritmo de selección

Cambio según el tipo de ordenamiento (ascendente o descendente)

Algoritmo de inserción

```
private boolean insercion(int tipo) {

    if (tipo == 0) {
        for (int j = 1; j < datos.datosFilas.length; j++) {

            DatosFila actual = datos.datosFilas[j];
            int i = j - 1;

            while ((i > -1) && (datos.datosFilas[i].getNota() > actual.getNota())) {
                datos.datosFilas[i + 1] = datos.datosFilas[i];
                i--;
            }
            datos.datosFilas[i + 1] = actual;
            actualizarPanel();
        }
    } else {
        for (int j = 1; j < datos.datosFilas.length; j++) {

            DatosFila actual = datos.datosFilas[j];
            int i = j - 1;

            while ((i > -1) && (datos.datosFilas[i].getNota() < actual.getNota())) {
                datos.datosFilas[i + 1] = datos.datosFilas[i];
                i--;
            }
            datos.datosFilas[i + 1] = actual;
            actualizarPanel();
        }
    }

    return true;
}

}
```

```
/**
 *
 * @author damia
 */
public class Reporte {

    private final String rutaGuardado = "E:\\NetBeansProjects\\Practica2\\reportesGenerados\\";
    private Ordenamiento ordenamiento;
    String reporte = "";

    public Reporte(Ordenamiento ord) {
        ordenamiento = ord;
        crearReporte();
    }

    private void crearReporte() {
        reporte += crearEncabezado("Reporte: " + ordenamiento.tituloGrafica);
        reporte += "<body>"
            + datosPersonales
            + generarDetalles()
            + generarEstadoInicial()
            + generarEstadoFinal()
            + "</body>\n"
            + "\n"
            + "</html>";

        generarArchivo();
    }

    private void generarArchivo() {
        try {

            String direccion = rutaGuardado + ordenamiento.tituloGrafica + "Reporte.html";

            File file = new File(direccion);
            if (!file.exists()) {
                file.createNewFile();
            }
            FileWriter fw = new FileWriter(file);
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write(reporte);
            bw.close();

            Desktop.getDesktop().open(file);
        }
    }
}
```

```

    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Datos del autor
private final String datosPersonales = "<div class=\"datosPersonales\">\n"
    + "    <h1>Damian Ignacio Pe\u00f1a Afre</h1>\n"
    + "    <h2>202110568</h2>\n"
    + "    </div>";

// función para generar tablas
private String crearTabla(String[] encabezados, DatosFila[] datosCuerpo) {
    String tabla = "\n<table>";

    //Encabezados
    String encabezadoTabla = "\n<t<thead>\n\t\t<tr>";
    for (String encabezado : encabezados) {
        encabezadoTabla = encabezadoTabla + "\n\t\t\t<th>" + encabezado + "</th>";
    }
    encabezadoTabla = encabezadoTabla + "\n\t\t</tr>\n\t</thead>";

    //cuerpo de tabla
    String cuerpoTabla = "\n\t\t<tbody>";
    for (int i = 0; i < datosCuerpo.length; i++) {
        String fila = "\n\t\t\t<tr>";
        fila = fila + "\n\t\t\t\t<td>" + datosCuerpo[i].getTitulo() + "</td>";
        fila = fila + "\n\t\t\t\t<td>" + datosCuerpo[i].getNota() + "</td>";
        fila = fila + "\n\t\t\t</tr>";
        cuerpoTabla = cuerpoTabla + fila;
    }
    cuerpoTabla = cuerpoTabla + "\n\t\t</tbody>";

    tabla = tabla + encabezadoTabla + cuerpoTabla + "\n</table>";

    return tabla;
}

private String generarDetalles() {
    return "    <div class=\"dobleColumna\">\n"
        + "        <div>\n"
        + "            <h2>Detalle Ordenamiento</h2>\n"
        + "            <p><span class=\"resaltar\">Algoritmo: </span> " +
ordenamiento.info.algoritmo + "</p>\n"
        + "            <p><span class=\"resaltar\">Tipo: </span> " +
ordenamiento.info.tipo + "</p>\n"
        + "            <p><span class=\"resaltar\">Velocidad: </span> " +
ordenamiento.info.velocidad + "</p>\n"
    + "        </div>\n"
    + "    </div>";
}

```

función para los detalles del ordenamiento

```

        + "                </div>\n"
        + "                <div>\n"
        + "                    <h2>Detalle de ejecucion</h2>\n"
        + "                    <p><span class=\"resaltar\">Tiempo: </span> " +
ordenamiento.minutos + ":" + ordenamiento.segundos + ":" + ordenamiento.milisegundos +
"</p>\n"
        + "                    <p><span class=\"resaltar\">Pasos: </span> " +
ordenamiento.pasos + "</p>\n"
        + "                </div>\n"
        + "            </div>";
    }

    private String generarEstadoInicial() {

        String[] encabezados = {ordenamiento.datos.tituloBarras,
ordenamiento.datos.tituloNumeracion};

        return "<div>\n"
            + "                <h2>Estado Inicial</h2>\n"
            + crearTabla(encabezados, Practica2.datosGraficaInicial.datosFilas)
            + "                <img src=\"../imagenesGeneradas/" + ordenamiento.tituloGrafica +
"Inicial.png\" alt=\"\">\n"
            + "            </div>";
    }

    private String generarEstadoFinal() {

        String[] encabezados = {ordenamiento.datos.tituloBarras,
ordenamiento.datos.tituloNumeracion};

        return "<div>\n"
            + "                <h2>Estado Final</h2>\n"
            + crearTabla(encabezados, ordenamiento.datos.datosFilas)
            + "                <img src=\"../imagenesGeneradas/" + ordenamiento.tituloGrafica +
"Final.png\" alt=\"\">\n"
            + "            </div>";
    }

    private String crearEncabezado(String titulo) {
        String encabezado = "<!DOCTYPE html>\n"
            + "<html lang=\"es\">\n"
            + "<head>\n"
            + "\t<meta charset=\"UTF-8\">\n"
            + "\t<meta http-equiv=\"X-UA-Compatible\" content=\"IE=edge\">\n"
            + "\t<meta name=\"viewport\" content=\"width=device-width, initial-
scale=1.0\">\n"
            + "\t<title> " + titulo + "</title>\n"
            + "\t<!-- Fuentes de google -->\n"

```



```

        + "\t<link rel=\"preconnect\" href=\"https://fonts.googleapis.com\">\n"
        + "\t<link rel=\"preconnect\" href=\"https://fonts.gstatic.com\"
crossorigin>\n"
        + "\t<link
href=\"https://fonts.googleapis.com/css2?family=Fredoka:wght@300&display=swap\"
rel=\"stylesheet\">\n"
        + "\t<link rel=\"stylesheet\" href=\"./css/estilos.css\">"
        + "</head>\n";
    return encabezado;
}
}

```

Paquete Vista

Frame Principal

```
private File archivo = null;    Archivo de la gráfica inicial y
private String titulo = null;   propiedades iniciales de la grafica

private String datos = "";
private JFreeChart graficaBarrasInicial;

public static DatosGrafica datosGrafica = null;
public static DatosGrafica datosGraficaInicial = null;    Propiedades estáticas del estado
                                                            inicial y final de la gráfica.

/**
 * Creates new form Practica2
 */
public Practica2() {
    initComponents();
    FileNameExtensionFilter filtro = new FileNameExtensionFilter("*.CSV", "csv");
    exploradorArchivos.setFileFilter(filtro);    Filtro para archivos .csv en el
                                                JFileChooser
}
```

```

private void btnBuscarActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_btnBuscarActionPerformed

    int estado = exploradorArchivos.showOpenDialog(this); Abre el explorador de archivos

    if (estado == JFileChooser.APPROVE_OPTION) {
        archivo = exploradorArchivos.getSelectedFile(); Guarda en un objeto el archivo
                                                    seleccionado
        this.textFielRuta.setText(archivo.getAbsolutePath());
                                                    Establece la dirección de dicho
                                                    archivo
    }
}

private void btnCargarActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_btnCargarActionPerformed
    //Resetea la informacion Resetea la información almacenada
    datos = ""; previamente
    datosGrafica = null;

    if (archivo != null) {
        try {
            BufferedReader bfr = new BufferedReader(new FileReader(archivo));

            String line = bfr.readLine();
            while (null != line) { Lee el archivo
                datos += line + "\n";
                line = bfr.readLine();
            }

            if (!datos.equals("")) {
                datosGrafica = new DatosGrafica(datos, comboBoxTipo.getSelectedIndex());
                datosGraficaInicial = new DatosGrafica(datos,
comboBoxTipo.getSelectedIndex());
                                                    Genera un objeto del tipo DatosGrafica
                                                    mandando el tipo de algoritmo seleccionado
                                                    en ese momento y la información del .csv en
                                                    forma de texto
            }

            bfr.close();

        } catch (FileNotFoundException ex) {
        } catch (IOException ex) {
        }
    }
}

```

```

    if (!textFieldTitulo.equals("")) {
        titulo = textFieldTitulo.getText();
    }

    if (titulo != null && datosGrafica != null) {

        JFreeChart graficoBarras = ChartFactory.createBarChart(titulo,
datosGrafica.tituloBarras, datosGrafica.tituloNumeracion,
DatosGrafica.crearDataSet(datosGrafica), PlotOrientation.VERTICAL, true, true, false);
        ChartPanel panelGrafica = new ChartPanel(graficoBarras);

        graficaBarrasInicial = graficoBarras;
        panelGrafica.setPreferredSize(new Dimension(380, 334));

        panelContenedorGrafica.setLayout(new BorderLayout());
        panelContenedorGrafica.add(panelGrafica, BorderLayout.CENTER);
        panelContenedorGrafica.validate();

    } else {
        JOptionPane.showMessageDialog(this, "Hubo algun problema");
    }

}

private void btnEjecutarActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_btnEjecutarActionPerformed
    // TODO add your handling code here:
    btnCargar.doClick();

    if (datosGrafica != null) {

        Ordenamiento ord = new Ordenamiento(datosGrafica,
comboBoxAlgoritmo.getSelectedIndex(), comboBoxTipo.getSelectedIndex(),
comboBoxVelocidad.getSelectedIndex(), titulo);

        Simulacion frame = new Simulacion(ord);

        String nombreArchivo = (titulo.equals("")) ? "Grafica" : titulo;

```

```
File imagenGrafica = new
File("E:\\NetBeansProjects\\Practica2\\imagenesGeneradas\\" + nombreArchivo + "Inicial.png");
try {
    ChartUtils.saveChartAsJPEG(imagenGrafica, graficaBarrasInicial, 600, 600);
} catch (IOException ex) {
}
}
}
```

Genera la imagen inicial de la grafica

```
private int pasos = 0;
private Ordenamiento ordenamiento;
public static Cronometro cronometro;

public Simulacion(Ordenamiento ord) {

    initComponents();

    ordenamiento = ord;

    //Informacion de la ejecucion
    labelAlgoritmo.setText(ordenamiento.info.algoritmo);
    labelTipo.setText(ordenamiento.info.tipo);
    labelVelocidad.setText(ordenamiento.info.velocidad);

    labelPasos.setText(pasos + "");
    labelTiempo.setText("00:00:00");

    cronometro = new Cronometro(labelTiempo);
    ordenamiento.empezarSimulacion(panelContenedorGrafica, labelPasos);
}
```

Objeto de ordenamiento y cronometro

Coloca la información del ordenamiento en su reespectivo label

Inicia un Cronometro nuevo y empieza la simulación