MANUAL TÉCNICO

Clase Coordenada

Se construyeron setters y getters para sus variables.

```
package Datos;
public class Coordenada {
	private String titulo;
	private int valores;
	public Coordenada(String titulo, int valores) {
		this.setTitulo(titulo);
		this.setValores(valores);
	}
	public String getTitulo() {
		return titulo;
	}
	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}
	public int getValores() {
		return valores;
	}
	public void setValores(int valores) {
		this.valores = valores;
	}
}
```

Clase Ploteo

Método “leer”

Se encarga de leer el archivo csv, además de guardar los nombres de los ejes. De igual manera envía los datos para insertarlos en el arreglo.

```
public static void leer(String ruta) {
	try {
		BufferedReader archivo = new BufferedReader(new
		InputStreamReader(new FileInputStream(ruta),"utf-8"));
		Scanner sc = new Scanner(archivo);
		String[] actual;
		Coordenada cord;
		actual = sc.nextLine().split(",");
		ejes[0] = actual[0];
		ejes[1] = actual[1];
		nDatos = 0;
		while(sc.hasNextLine()) {
			try {
				actual = sc.nextLine().split(",");
				cord = new
				Coordenada(actual[0],Integer.parseInt(actual[1]);
				insertar(cord);
			}catch(Exception e) {
			}
		}
		sc.close();
	}catch(Exception e) {
		JOptionPane.showMessageDialog(null, "Error de lectura");
	}
}
```

Función “insertar”

Se encarga de agregar los datos leídos al arreglo coordenadas.

```
public static boolean insertar(Coordenada cord) {
	for(int i = 0; i < coordenadas.length; i++) {
		if(coordenadas[i] == null) {
			coordenadas[i] = cord;
			nDatos ++;
			return true;
		}
	}
	return false;
}
```

Función “limpiar”

Se encarga de asignarle un valor NULL a las posiciones ocupadas en el arreglo coordenadas.

```
public static boolean limpiar() {
	for(int i = 0; i < coordenadas.length; i++) {
		if(coordenadas[i] != null) {
			coordenadas[i] = null;
		}else {
			return true;
		}
	}
	return true;
}
```

Clase Grafico

Función “generarGrafica”

Se encarga de generar la gráfica con los datos requeridos (títulos de ejes, información para plotear y la orientación de la gráfica).

```
ChartPanel generarGrafica() {
	chart = ChartFactory.createBarChart(
		titulo,
		ejes[0],
		ejes[1],
		dataset(),
		PlotOrientation.VERTICAL, true, true, false);
	ChartPanel chartPanel = new ChartPanel(chart);
	chartPanel.setPreferredSize(new java.awt.Dimension(getWidth(),getHeight()));
	return chartPanel;
}
```

Función “dataset”

Se encarga de obtener los datos del arreglo coordenadas para ser graficados.

```
DefaultCategoryDataset dataset() {
	DefaultCategoryDataset dataset = new DefaultCategoryDataset();
	for (Coordenada coord : coordenadas) {
		if (coord != null) {
			dataset.setValue(coord.getValores(), coord.getTitulo(), "");
		}
	}
	return dataset;
}
```

Método “saveImg”

Se encarga de guardar una imagen de la gráfica generada en un directorio específico.

```
public static void saveImg(String nombre) {
	int width = 640;
	int height = 480;
	File lineChart = new File("Reporte/images/" + nombre + ".png");
	try {
		ChartUtils.saveChartAsPNG(lineChart, chart, width, height);
	} catch (Exception e) {
	}
}
```

Método “generarReporte”

Se encarga de concatenar cada una de las partes que componen el reporte en **HTML** y escribirlo en el archivo correspondiente.

```
void generarReporte() {
	String reporteHtml = inicioHtml;
	reporteHtml += tabMiInfo;
	reporteHtml += infoOrd;
	reporteHtml += divCuadrar;
	reporteHtml += titulosT;
	reporteHtml += tablaInicio;
	reporteHtml += tablaFin;
	reporteHtml += imagenes;
	reporteHtml += cDivCuadrar;
	reporteHtml += finHtml;
	try {
		PrintWriter reporte = new PrintWriter(new File("Reporte.html"), "UTF-8");
		reporte.write(reporteHtml);
		reporte.close();
	} catch (Exception e) {
	}
}
```

Método “animar”

Se encarga de iniciar el hilo para animar la gráfica mientras se ordenan los datos.

```
public void animar(String ordenamiento, String sentido) {
	this.ordenamiento = ordenamiento;
	this.sentido = sentido;
	Thread animacion = new Thread(this);
	animacion.start();
}
```

Método “run”

- Bubble Sort (Ascendente)
  El método realiza **n^2** iteraciones para **n** cantidad de datos. Si el dato actual es mayor (ascendente) o menor (descendente) que el siguiente se efectúa un intercambio de posiciones.
  Nota: para realizar el ordenamiento descendente se invierte el signo **>** en la condición, es decir se coloca el signo **<**.

```
for (int i = 0; i < nDatos - 1; i++) {
        for (int x = 0; x < nDatos - i - 1; x++) {
                if (coordenadas[x].getValores() > coordenadas[x + 1].getValores()) {
                        Coordenada temporal = coordenadas[x];
                        coordenadas[x] = coordenadas[x + 1];
                        coordenadas[x + 1] = temporal;
                        removeAll();
                        add(generarGrafica());
                        revalidate();
                        repaint();
                        nPasos++;
                        numPasos.setText(String.valueOf(nPasos));
                        Thread.sleep(300);
                }
        }
}
```

- Insertion Sort (Ascendente)
  Se itera sobre el arreglo desde la segunda posición (índice 1) hasta la posición n (índice n - 1) y se itera hacia atrás el arreglo partiendo de la posición actual. Si la posición actual es menor (ascendente) o mayor (descendente) que la posición anterior se efectúa un intercambio.
  Nota: para realizar el ordenamiento descendente se invierte el signo **<** en la condición, es decir se coloca el signo **>**.

```
for (int i = 1; i < nDatos; i++) {
        for (int x = i; x > 0; x--) {
                if (coordenadas[x].getValores() < coordenadas[x - 1].getValores()) {
                        Coordenada temporal = coordenadas[x];
                        coordenadas[x] = coordenadas[x - 1];
                        coordenadas[x - 1] = temporal;
                        removeAll();
                        add(generarGrafica());
                        revalidate();
                        repaint();
                        nPasos++;
                        numPasos.setText(String.valueOf(nPasos));
                        Thread.sleep(300);
                }
        }
}
```

- Shell Sort (Ascendente)
  Nota: para realizar el ordenamiento descendente se invierte el signo **>**, es decir se coloca el signo **<**.

```
for (int gap = n / 2; gap > 0; gap /= 2) {
	for (int i = gap; i < n; i++) {
		Coordenada key = coordenadas[i];
		for (int j = i; j >= gap && coordenadas[j - gap].getValores() >
		key.getValores(); j -= gap) {
			coordenadas[j] = coordenadas[j - gap];
			coordenadas[j - gap] = key;
			removeAll();
			add(generarGrafica());
			revalidate();
			repaint();
			nPasos++;
			numPasos.setText(String.valueOf(nPasos));
			Thread.sleep(300);
		}
	}
}
```

Clase Cronometro

Método “run”

Mientras **iniciar** sea verdadero el tiempo seguirá aumentando, cuando el valor de milésimas llega a 1000 aumenta en 1 el valor de segundos. Cuando el valor de segundos llega 60 aumenta en 1 el valor de minutos.

```
public void run() {
	Integer minutos = 0, segundos = 0, milesimas = 0;
	String min = "", seg = "", mil = "";
	try {
		while(iniciar) {
		milesimas += 4;
		if(milesimas == 1000) {
			milesimas = 0;
			segundos += 1;
			if(segundos == 60) {
				segundos = 0;
				minutos++;
			}
		}
		if(minutos < 10) min = "0" + minutos;
		else min = minutos.toString();
		if(segundos < 10) seg = "0" + segundos;
		else seg = segundos.toString();

		if(milesimas < 10) mil = "00" + milesimas;
		else if( milesimas < 100 ) mil = "0" + milesimas;
		else mil = milesimas.toString();
		time.setText( min + ":" + seg + ":" + mil );
		Thread.sleep(4);
		}
	}catch(Exception e) {}
}
```

Método “iniciarCronometro”

Se encarga de iniciar el hilo.

```
public void iniciarCronometro() {
	iniciar = true;
	hilo = new Thread(this);
	hilo.start();
}
```

Método “pararCronometro”

Se encarga de detener el hilo.

```
public void pararCronometro() {
	iniciar = false;
	hilo.stop();
}
```