G-SORT 770

PRÁCTICA 2

MANUAL TÉCNICO

Ing. Moisés Velázquez

Aux. José Hernández

Daniel Estuardo Cuque Ruíz
202112145

Índice

Inicio

Botón Buscar

```
private void botonBuscarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    fileChooser = new JFileChooser();
    fileChooser.setCurrentDirectory(new java.io.File("/Users/daniel/Desktop/USAC"
            + " PRIMER SEMESTRE 2022/IPC 1/LABORATORIO/PRACTICAS/Practica_2"));

    FileNameExtensionFilter filtro = new FileNameExtensionFilter("*.CSV","csv");

    fileChooser.setFileFilter(filtro);
    int seleccion = fileChooser.showOpenDialog(this);

    if(seleccion == JFileChooser.APPROVE_OPTION){
        fichero = fileChooser.getSelectedFile();

        this.cajaRutaArchivos.setText(fichero.getAbsolutePath());
    }
}
```

Con la propiedad *setCurrentDirectory*, se colocará la ruta absoluta por defecto en donde se abrirá el objeto de la clase JFileChooser(). También se agregará un filtro para que solo muestre los archivos con la extensión .CSV, posteriormente se mostrará la ruta absoluta dentro de un *JTextField*.

Botón Cargar

Dentro del botón de cargar, verificamos que el usuario ya haya ingresado una ruta para el archivo .csv, y que también haya ingresado el nombre de la gráfica. Luego creamos un objeto de la clase BufferedReader para leer el texto que contenga el archivo .csv, también creamos un String para que se concatene el texto del archivo. Luego del ciclo while, se crea un arreglo para guardar los datos temporalmente y separamos los datos por cada salto de línea (“\n”). Después guardamos los títulos y llenamos un arreglo de la clase Datos. Y mostramos la gráfica.

Crear gráfico

```
private void crearGrafico(){
        for(Datos dato: arregloDatos){
            if(dato != null){
                dataSet.addValue(dato.getDatoNumerico(), dato.getDatoTexto(),dato.getDatoTexto());
            }
        }
        chart = ChartFactory.createBarChart(tituloGrafica, titulos[0],
                titulos[1], dataSet, PlotOrientation.VERTICAL, true,true,false);

        panelDatos = new ChartPanel(chart);
        panelDatos.setMouseWheelEnabled(true);
        contenedorGrafica.add(panelDatos, BorderLayout.CENTER);
        pack();
        repaint();
}
```

Luego de haber llenado el arreglo de tipo Dato, creamos un objeto dataset para añadir valores, luego creamos un chart que creará la gráfica. Posteriormente creamos un panel donde se almacenará el chart y lo añadimos al panel principal.

Reiniciar dataset

Con este método limpiamos los datos del objeto dataset por si por algún motivo deseamos cambiar de archivo .csv

```
private void reiniciarGrafica(){
    dataSet.clear();
    repaint();
    pack();
}
```

Botón Ejecutar

```
private void botonEjecutarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String tipoAlgoritmo = (String) listaAlgoritmos.getSelectedItem();
    String tipoOrden = (String) listaTipo.getSelectedItem();
    String tipoVelocidad = (String) listaVelocidad.getSelectedItem();
    new Ejecucion(tipoAlgoritmo,tipoOrden, tipoVelocidad).setVisible(true);
    this.dispose();
}
```

Con este botón capturamos los datos de los *JComboBox* y los creamos un objeto de la clase *Ejecución* con los parámetros seleccionados.

Datos

Creamos una clase para generar objetos que contengan los datos que se ingresen con el .csv, tendrá como propiedad un tipo String que contiene el dato de texto, y otro de tipo int que contendrá los datos numéricos para crear las barras.

```
public class Datos {

    //Atributos de la clase
    private String datoTexto;
    private int datoNumerico;

    public Datos(String datoTexto, int datoNumerico){
        this.datoTexto = datoTexto;
        this.datoNumerico = datoNumerico;
    }

    public Datos(){

    }
    public String getDatoTexto() {
        return datoTexto;
    }

    public void setColumna1(String columna1) {
        this.datoTexto = columna1;
    }

    public int getDatoNumerico() {
        return datoNumerico;
    }

    public void setDatoNumerico(int datoNumerico) {
        this.datoNumerico = datoNumerico;
    }
}
```

Cronómetro

Run Cronometro

```
*/
public class Cronometro extends Thread{

    private static int  minutos = 0, segundos = 0, milisegundos = 0;
    String ceroSegundos = "0";
    String ceroMinutos = "0";

    public Cronometro(){

    }

    public void run(){
        try{
            while(Ejecucion.verificarOrdenamiento){
                comenzarCronometro();
                Thread.sleep(1,777777);
            }
            Reportes.generarDocHTML();
            abrirDocumento();

        }catch(Exception e){
            System.out.println("Problemas con el cronómetro "+e.getMessage());
        }
    }
```

Creamos las variables estáticas de tipo int para almacenar los valores de minutos, segundos y milisegundos dentro del cronómetro. Con el método run() se iniciará el hilo de la clase Cronómetro, y se ejecutará el método de comenzarCronometro() con un retaro de 1ms con 777777 nanosegundos.

Comenzar cronómetro

Con este método aumentamos los milisegundos de 2 en 2, posteriormente, los segundos tienen 1000 milisegundos, por lo que cuando el valor de milisegundos sea mayor a 999ms, se reiniciarán los ms y se aumentará 1 segundo, cuando los segundos sean mayores a 59, se reiniciará y se aumentará 1 minuto.

```
private void comenzarCronometro(){
    //Comenzamos a aumentar los milisegundos
    milisegundos+=2;

    /*
        Un segundo tiene 1000 ms por lo que
        cuando sea 1000, regresamos los ms a
        0 y sumamos 1 unidad a los segundos
    */
    if(milisegundos > 999){
        milisegundos = 0;
        segundos++;

        /*1 minuto tiene 59 s por lo que
         cuando los segundos lleguen a 59
         los reiniciamos y aumentamos 1
         unidad a los minutos
        */
        if(segundos  > 59){
            segundos = 0;
            minutos++;
        }
        if(segundos > 9){
            ceroSegundos = "";
        }
        if(minutos > 9){
            ceroMinutos = "";
        }
    }
    String reloj = ceroMinutos + minutos + ":" + ceroSegundos+segundos + ":" + milisegundos;
    Ejecucion.etiquetaTiempo.setText(reloj);
}
```

Por estética revisamos que cuando los minutos o segundos sean menores a 9, entonces se agregará un 0 al costado. Por ejemplo “00:09:200”, y cada vez que se llame a este método se mostrará el valor dentro de un *JLabel* en la ventana de Ejecución.

Abrir documento HTML

Cuando el bucle del hilo finalice, se abrirá el documento HTML generado, para ello se necesita de la case Desktop, le colocamos una ruta absoluta de donde se encuentre el documento y con la propiedad open, nos abrirá el documento dentro de nuestro navegador.

```
private void abrirDocumento(){
    if(Desktop.isDesktopSupported()){
        Desktop desktop = Desktop.getDesktop();
        try{
            desktop.open(new File("/Users/daniel/Desktop/USAC PRIMER SEMESTRE 2022/IPC 1/LABORATORIO/PRACTICAS/Practica_2/Reporte-de-ejecucion.html"));
        }catch(Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

Ordenamiento

Run Ordenamiento

Se ejecutará el método de ordenamiento dependiendo del tipo de algoritmo que se encuentre dentro del constructo de la clase.

```
@Override
public void run(){
    try{
        switch(algoritmo){
            case burbuja:
                metodoBurbuja();
                break;
            case seleccion:
                metodoSeleccion();
                break;
            case insercion:
                metodoInsercion();
                break;
        }
    }catch(Exception e){
        System.out.println("Problemas con el ordenamiento "+e.getMessage());
    }
}
```

Crear gráfica de ordenamiento

Se creará los dataset, chart y chartpanel para mostar la gráfica del ordenamiento, además se guardará el gráfico como una imagen si presenta el estado inicial o final del ordenamiento. Con la propiedad repaint() refrescamos el panel.

```
public void crearGrafico(int i){
    String titulo = "";
    DefaultCategoryDataset dataSet = new DefaultCategoryDataset();
    JFreeChart chart;
    rellenarDataSet(dataSet);
    chart = ChartFactory.createBarChart(Inicio.tituloGrafica, Inicio.titulos[0], Inicio.titulos[1], dataSet, PlotOrientation.VERTICAL, true,true,false);
    if(i == 0 || i == 1){
        if(i == 0){
            titulo = "imagenInicial.png";
        } else if(i == 1){
            titulo = "imagenFinal.png";
        }
        try{
        final ChartRenderingInfo info = new ChartRenderingInfo(new StandardEntityCollection());
        final File imagen = new File("/Users/daniel/Desktop/USAC PRIMER SEMESTRE 2022/IPC 1/LABORATORIO/PRACTICAS/Practica_2/Imagenes/"+titulo);
        ChartUtilities.saveChartAsPNG(imagen, chart, 600, 700,info);
        }catch(Exception e){
        }
    }
    ChartPanel panelDatos = new ChartPanel(chart);
    panelDatos.setMouseWheelEnabled(true);
    Ejecucion.panelGrafica.add(panelDatos,BorderLayout.CENTER);
    new Ejecucion().repaint();
    new Ejecucion().pack();
}
```

Rellenar dataset

Dentro de este método rellenamos el dataset para usarlo dentro del objeto chart para la gráfica, dependiendo si la gráfica se ordena de manera ascendente o descendente.

```
public void rellenarDataSet(DefaultCategoryDataset dataSet){
    switch(tipo){
        case descendente:
            for (int i = (arregloDatos.length - 1); i >= 0; i--) {
                if(arregloDatos[i] != null){
                    dataSet.addValue(arregloDatos[i].getDatoNumerico(), arregloDatos[i].getDatoTexto(), arregloDatos[i].getDatoTexto());
                }
            }
            break;
        case ascendente:
            for (int i = 0; i < arregloDatos.length; i ++) {
                if(arregloDatos[i] != null){
                    dataSet.addValue(arregloDatos[i].getDatoNumerico(), arregloDatos[i].getDatoTexto(), arregloDatos[i].getDatoTexto());
                }
            }
            break;
    }
}
```

Establecer velocidad

```
private void establecerVelocidad(){
    switch(velocidad){
        case strRapido:
            nVelocidad = 500;
            break;
        case strMedio:
            nVelocidad = 1000;
            break;
        case strLento:
            nVelocidad = 1700;
            break;
    }
}
```

Con este método establecemos las distintas velocidades para el ordenamiento del arreglo.

Metodo Burbuja

Con el primer ciclo for recorremos todo el arreglo, con el segundo ciclo for hacemos el intercambio de elementos. Para ello nos apoyamos de un objeto pivote, o auxiliar, si el dato numérico en la posición j es mayor al de la posición j+1 entonces el arreglo auxiliar va a tomar el valor del arreglo de la posición j temporalmente, luego el objeto de la posición j tomará el valor de la posición j+1 y luego la posición j+1 tomará el valor del objeto auxiliar, todo esto con la finalidad de hacer el intercambio de los objetos dentro del arreglo.

```
public void metodoBurbuja(){
    crearGrafico(0);
    Reportes.generarEstadoIncial();
    for(int i = 0; i < arregloDatos.length; i++){
        for(int j = 0; j < (arregloDatos.length - 1); j++){
                    mostrarPasos();
                    try {
                        Thread.sleep(nVelocidad);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(Ordenamiento.class.getName()).log(Level.SEVERE, null, ex);
                    }

                    if(arregloDatos[j].getDatoNumerico() > arregloDatos[j+1].getDatoNumerico()){
                            auxiliar = arregloDatos[j];
                            arregloDatos[j] = arregloDatos[j+1];
                            arregloDatos[j+1] = auxiliar;
                    }
                    crearGrafico(2);
        }
    }
    crearGrafico(1);
    Ejecucion.verificarOrdenamiento = false;
}
```

Metodo por selección

```
public void metodoSeleccion(){
    crearGrafico(0);
    Reportes.generarEstadoIncial();
    int i,j,pos;
    Datos tmp;
    for(i = 0; i < (arregloDatos.length-1); i++){
            pos = i;
            for(j = (i+1); j < arregloDatos.length; j++){
                if(arregloDatos[j].getDatoNumerico() < arregloDatos[pos].getDatoNumerico()){
                    pos = j;
                }

                try {
                    Thread.sleep(nVelocidad);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Ordenamiento.class.getName()).log(Level.SEVERE, null, ex);
                }
            mostrarPasos();
            crearGrafico(2);
            }
            tmp = arregloDatos[i];
            arregloDatos[i] = arregloDatos[pos];
            arregloDatos[pos] = tmp;
    }
    crearGrafico(1);
    Ejecucion.verificarOrdenamiento = false;
}
```

Con este nos apoyamos de un pivote para los índices, y un pivote para los datos, recorremos todo el arreglo, el objeto verifica si el dato del arreglo en la posición j es menor al de la posición pos, si es verdadero, entonces el pivote de posición toma el valor del j que se está verificando, luego con el pivote auxiliar, luego ese pivote tomara el valor del objeto en la posición i, el arreglo de la posición i tomará el valor de la posición pos, y luego el objeto en [pos] tomará el valor del pivote para los datos.

Método por inserción

```
public void metodoInsercion(){
    crearGrafico(0);
    Reportes.generarEstadoIncial();
    int i,j;
    for(i = 1; i < arregloDatos.length; i++){
            try {
                Thread.sleep(nVelocidad);
            } catch (InterruptedException ex) {
                Logger.getLogger(Ordenamiento.class.getName()).log(Level.SEVERE, null, ex);
            }
            auxiliar = arregloDatos[i];
            for (j = i; (j > 0) && (arregloDatos[j-1].getDatoNumerico() > auxiliar.getDatoNumerico()); j--) {
                arregloDatos[j] = arregloDatos[j-1];
            }
            arregloDatos[j] = auxiliar;
            crearGrafico(2);
            mostrarPasos();
    }
    crearGrafico(1);
    Ejecucion.verificarOrdenamiento = false;
}
```

Vamos a necesitar usar dos ciclos for, el primero es para recorrer todo el arreglo, el segundo es para ir ordenando los datos según el índice del primer ciclo. En la condición del ciclo, la primera posición será de verificar que j e i sean distintos de 0 porque en la segunda proposición verificamos la posición j-1, y si j fuera 0, entonces j-1 sería -1 y saltaría una excepción, una vez controlado eso, verificamos si el dato de la posición j-1 es mayor al dato auxiliar, si esta condición se cumple, entonces el ciclo se volverá a ejecutar, y el valor del arreglo en la posición j será el de posición j-1.

Metodo para mostrar los pasos

Este método aumenta el número de pasos y lo muestra en el objeto *JLabel* para indicar la cantidad de pasos que realizó el ordenamiento.

```
private void mostrarPasos(){
    contador++;
    Ejecucion.etiquetaPasos.setText(String.valueOf(contador));
}
```

Ejecución

Constructor de ejecución

```
public Ejecucion(String tipoAlgoritmo, String tipoOrdenamiento, String tipoVelocidad) {

    this.tipoAlgoritmo = tipoAlgoritmo;
    this.tipoOrdenamiento = tipoOrdenamiento;
    this.tipoVelocidad = tipoVelocidad;
    initComponents();
    iniciarCronometro();
    mostrarAtributosOrdenamiento();
    iniciarOrdenamiento();
    this.setLocationRelativeTo(null);
    this.setTitle("SIMULACIÓN");

}
```

Dentro del constructor de la clase, se requiere el tipo de algoritmo, tipo de ordenamiento, y la velocidad con la que se va a ejecutar el ordenamiento, todo eso proveniente de la clase de Inicio.

Mostar atributos

Con este método mostramos los atributos del ordenamiento dentro de los *JLabels*

```
private void mostrarAtributosOrdenamiento(){
    etiquetaAlgoritmo.setText(this.tipoAlgoritmo);
    etiquetaTipo.setText(this.tipoOrdenamiento);
    etiquetaVelocidad.setText(this.tipoVelocidad);
}
```

Iniciar ordenamiento

Con este método iniciamos el hilo del ordenamiento de la clase Ordenamiento con el metodo start().

```
private void iniciarOrdenamiento(){
    ordenamiento = new Ordenamiento(this.tipoAlgoritmo,this.tipoOrdenamiento,this.tipoVelocidad);
    ordenamiento.start();
}
```

Iniciar el cronómetro

Llamamos al hilo para iniciar el cronómetro de la clase Cronometro con la propiedad start()

```
private void iniciarCronometro(){
    Cronometro cronometro = new Cronometro();
    cronometro.start();
}
```

Reportes

Generar estado inicial

```
public static void generarEstadoIncial(){
    //Construimos la tabla inicial sin ordenar
    tablaInicial = "        <div class=\"estado-inicial\">\n" +
      <h1>ESTADO INICIAL</h1>\n" +
      <table>\n" +
        <tr>\n" +
          <th>"+Inicio.titulos[0]+"</th>\n";
    for (Datos arregloDato : Inicio.arregloDatos) {
      tablaInicial += "             <td>" + arregloDato.getDatoTexto() + "</td>\n";
    }

          tablaInicial +=
        </tr>\n" +
        <tr>\n" +
          <th>"+Inicio.titulos[1]+"</th>\n";

    for (Datos arregloDato : Inicio.arregloDatos) {
      tablaInicial += "             <td>" + arregloDato.getDatoNumerico() + "</td>\n";
    }
          tablaInicial +=
        </tr>\n" +
      </table>\n" +
      <div class=\"imagen\">\n" +
        <img\n" +
          src=\"/Users/daniel/Desktop/USAC PRIMER SEMESTRE 2022/IPC 1/LABORATORIO/PRACTICAS/Practica_2/Imagenes/imagenInicial.png\"\n" +
          alt=\"Grafica Inicial\"\n" +
        />\n" +
      </div>\n" +
      <span class=\"desc-grafica\">Gráfica desordenada</span>\n" +
    </div>";
}
```

Al crear el reporte se necesitan los datos del estado inicial de la gráfica antes del ordenamiento. Por lo que se crea el *String* con los valores iniciales del arreglo.

Generar codigoHTML

```
public static String generarCodigoHTML(){
    //Colocamos los datos del ordenamiento
    detalleOrdenamiento = "<div class=\"detalle-ordenamiento\">\n" +
      <h1>Detalle de ordenamiento</h1>\n" +
      <p>\n" +
        <span class=\"descripciones\">Algoritmo: </span>\n" +
        <span>"+Ejecucion.etiquetaAlgoritmo.getText()+"</span>\n" +
      </p>\n" +
      <p>\n" +
        <span class=\"descripciones\">Tipo: </span>\n" +
        <span>"+Ejecucion.etiquetaTipo.getText()+"</span>\n" +
      </p>\n" +
      <p>\n" +
        <span class=\"descripciones\">Velocidad: </span>\n" +
        <span>"+Ejecucion.etiquetaVelocidad.getText()+"</span>\n" +
      </p>\n" +
    </div>\n";

    //Colocamos el tiempo y los pasos
    detalleEjecucion = "        <div class=\"detalle-ejecucion\">\n" +
      <h1>Detalle de ejecución</h1>\n" +
      <p>\n" +
        <span class=\"descripciones\">Tiempo: </span>\n" +
        <span>"+Ejecucion.etiquetaTiempo.getText()+"</span>\n" +
      </p>\n" +
      <p>\n" +
        <span class=\"descripciones\">Pasos: </span>\n" +
        <span>"+Ejecucion.etiquetaPasos.getText()+"</span>\n" +
      </p>\n" +
    </div>";

          //Construimos la tabla final ordenada
          tablaFinal += "<div class=\"estado-final\">\n" +
      <h1>ESTADO FINAL</h1>\n" +
      <table>\n" +
        <tr>\n" +
          <th>"+Inicio.titulos[0]+"</th>\n";
          for (Datos arregloDato : Ordenamiento.arregloDatos) {
            tablaFinal += "             <td>" + arregloDato.getDatoTexto() + "</td>\n";
          }
        tablaFinal +=
        </tr>\n" +
        <tr>\n" +
          <th>"+Inicio.titulos[1]+"</th>\n";
    for (Datos arregloDato : Ordenamiento.arregloDatos) {
      tablaFinal += "             <td>" + arregloDato.getDatoNumerico() + "</td>\n";
    }
```

Con este método añadimos los valores de la tabla ordenada y los concatenamos al String de la tabla final, posteriormente concatenamos todas nuestras variables que contengan parte del documento HTML.

Generar documento HTML

Con el objeto de la clase *File* establecemos la ruta en donde queremos que se guarde el documento HTML, con el objeto FileWriter creamos el archivo HTML, con el objeto PrintWriter ingresamos el texto que se genera del método generarCodigoHTML, con la propiedad close() indicamos que deje de leer los archivos.

```
public static void generarDocHTML(){
    String texto = generarCodigoHTML();
     File f;
     FileWriter w;
     BufferedWriter bw;
     PrintWriter pw;

     try{
     f = new File("/Users/daniel/Desktop/USAC PRIMER SEMESTRE 2022/IPC 1/LABORATORIO/PRACTICAS/Practica_2/Reporte-de-ejecucion.html");
     w = new FileWriter(f);
     bw = new BufferedWriter(w);
     pw = new PrintWriter(bw);
     pw.write(texto);
     bw.close();
     pw.close();

     }catch(IOException e){
         System.out.println(e.getMessage());
     }
}
```