Práctica 2

Manual Tecnico

Kelly Mischel Herrera Espino
UNIVERSIDAD SAN CARLOS DE GUATEMALA

Manual Técnico

Introducción:

Para la realización del proyecto fue necesaria la creación de trece clases y la utilización de Programación Orientada a Objetos. Así como el uso de memoria estática para poder guardar los datos que el usuario carga por medio de un archivo csv. Se hace el uso de arreglos, interfaz grafica y el uso de hilos para poder realizar mas de un proceso al mismo tiempo como por ejemplo el tiempo y la presentación de cada cambio que tiene la grafica al aplicar un método de ordenamiento. Al finalizar el proceso de ordenamiento de los datos se genera un archivo html el cual es un reporte con la grafica desordenada y la grafica ordenada. Para ello se utilizo CSS en conjunto con html.

Requisitos:

Para el uso de la aplicación se necesita que la computadora tenga una memoria RAM de 2.00 GB y un sistema operativo de 64 bits.

Métodos

VentanPrincipal(): Es un constructor de la clase VentanaPrincipa el cual hace instancia del JFrame.

```
52      public VentanaPrincipal(){
53          super();
54          this.setSize(600,700);
55          this.setTitle("Ventana Principal");
56          this.setDefaultCloseOperation(EXIT_ON_CLOSE);
57          this.setLayout(null);
58          this.setLocationRelativeTo(null);
            agregarBotones();
```

agregarBotones(): Es un método en el cual se encuentran los componentes del JFrame como botones, JTextField y JLabels. En este método se instancias estos componentes asignándoles un tamaño, posición, color y algún texto en específico.

```
public void agregarBotones(){
    botonBuscar= new JButton("Buscar");
    botonBuscar.setBounds(363,80 ,220, 45);

    this.add(botonBuscar);
//
    buscar= new JTextField();
    buscar.setBounds(10,80,350, 45);
    this.add(buscar);
    //

    titulo= new JTextField();
    titulo.setBounds(10,160, 350, 45);
    this.add(titulo);

   botonAcepatar= new JButton("Aceptar");
   botonAcepatar.setBounds(363,160 ,220, 45);
    this.add(botonAcepatar);



    ordenar= new JButton("ordenar");
   ordenar.setBounds(363,208 ,100, 45);
    this.add(ordenar);

    iniciarBotones();
}
```

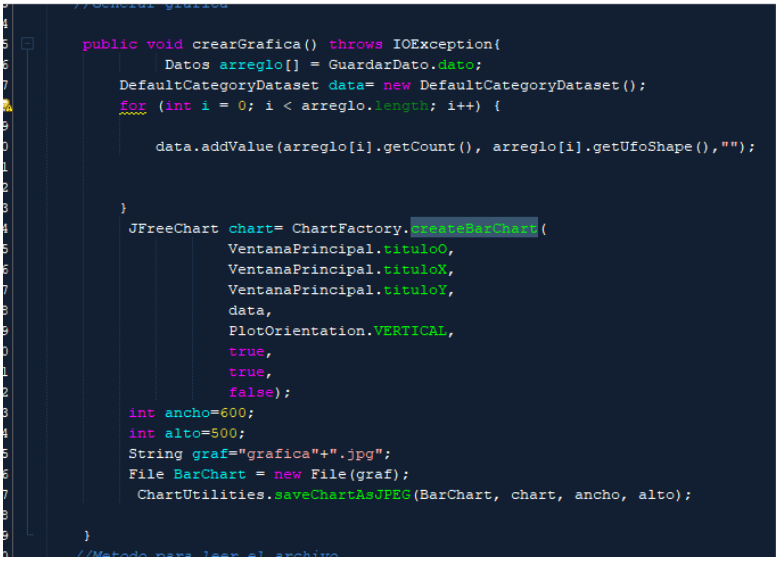
iniciarBotones(): Contiene los eventos de los botones aceptar, buscar y ordenar los cuales al pulsar realizan inician el proceso que contienen adentro o de los métodos escritos adentro del evento.

```
public void iniciarBotones(){
    botonBuscar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {
           abrirTexto();
           buscar.setText(texto);
        }
    });
    //Evento del boton aceptar

    botonAcepatar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {
          texto2=buscar.getText();
            leerRuta();
            titulo0=titulo.getText();
            JOptionPane.showMessageDialog(null, "Listo");

        }
    });

    ordenar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {


           Opciones op= new Opciones();

           op.show();
```

leerRuta(): Este método tiene como proceso leer la ruta de un archivo csv y poder obtener la ruta de este y guardarlo en una variable de tipo String para poder mostrarlo en el JTextField.

```
public void leerRuta(){
     String cadena2="";
     int cont=0;
     System.out.println(texto2);
  try{


          FileReader archivo= new FileReader(texto2);
          BufferedReader leer= new BufferedReader(archivo);
          while((cadena2=leer.readLine())!=null){
              texto3+=cadena2+"\n";
              cont++;


      }
          leer.close();
  }catch(IOException b){
      JOptionPane.showMessageDialog(null, b+" "+"\n No se encontro el archivo f" ,
          "Adevertencia",JOptionPane.WARNING_MESSAGE);

  }
  tamaño=cont-1;
  System.out.println("cont"+tamaño);
        quitarComa();
        obtenerTitulo();
          graficar();


}
//Generar grafica
```

void crearGrafica(): Se encarga de generar una gráfica como imagen haciendo uso de ChartUtilities en el cual se le asignan los valores correspondientes.



```
    public void crearGrafica() throws IOException{
            Datos arreglo[] = GuardarDato.dato;
        DefaultCategoryDataset data= new DefaultCategoryDataset();
        for (int i = 0; i < arreglo.length; i++) {

            data.addValue(arreglo[i].getCount(), arreglo[i].getUfoShape(),"");


        }
         JFreeChart chart= ChartFactory.createBarChart(
                    VentanaPrincipal.tituloO,
                    VentanaPrincipal.tituloX,
                    VentanaPrincipal.tituloY,
                    data,
                    PlotOrientation.VERTICAL,
                    true,
                    true,
                    false);
        int ancho=600;
        int alto=500;
        String graf="grafica"+".jpg";
        File BarChart = new File(graf);
         ChartUtilities.saveChartAsJPEG(BarChart, chart, ancho, alto);

    }
```

abrirTexto(): Este método tiene como proceso abrir un JFileChooser para poder escoger algún archivo y luego leer la ruta.

```
    }
    //Metodo para leer el archivo
    public void abrirTexto(){


      try{
          JFileChooser file= new JFileChooser();
          file.showOpenDialog(this);
          File abrir=file.getSelectedFile();
          if (abrir!=null) {
              FileReader archivo = new FileReader(abrir);
              texto= String.valueOf(abrir);




          }
      }catch(Exception e){
          JOptionPane.showMessageDialog(null, e+" "+"\n No se encontro el archivo");
      }


    }
    //Grafica
     public void graficar(){
          Datos arreglo[] = GuardarDato.dato;
        DefaultCategoryDataset data= new DefaultCategoryDataset();
        for (int i = 0; i < arreglo.length; i++) {
```

graficar(): En este método de tipo void se encuentra el proceso para generar una grafica.

```
public void graficar(){
      Datos arreglo[] = GuardarDato.dato;
    DefaultCategoryDataset data= new DefaultCategoryDataset();
    for (int i = 0; i < arreglo.length; i++) {

        data.addValue(arreglo[i].getCount(), arreglo[i].getUfoShape(), "");


    }
     JFreeChart chart= ChartFactory.createBarChart(
                titulo.getText(),
                tituloX,
                tituloY,
                data,
                PlotOrientation.VERTICAL,
                true,
                true,
                false);


    ChartPanel panelBarra= new ChartPanel(chart);

//        panel.setLayout(null);
//        panel.setBounds(10,250 ,300,300);
//        panel.add(panelBarra);
//        panel.setVisible(true);
//        panel.setBackground(Color.yellow);
    panelBarra.setBounds(10, 250, 500, 350);
    panelBarra.setVisible(true);
```

obtenerTitulo(): Tiene como proceso obtener el titulo en el eje x y en eje y de los datos.

```
//Metodo para obtener el titulo de los ejes
public void obtenerTitulo(){
    System.out.println("----------------------");
    System.out.println("hola");
      GuardarDato gd= new GuardarDato();
     String fila[]= texto3.split("\n");
     String t="";
     int tamañoF=fila.length-(fila.length-1);
     for (int i = 0; i < tamañoF; i++) {
        String[]palabra= fila[i].split(",");
        t=palabra[0];
        tituloY= palabra[1];

     }
     System.out.println(t.substring(3,t.length()));
     tituloX=t.substring(4,t.length());
     System.out.println(tituloY);
}
//Metodo para quitar la cadena
```

quitarComa(): Tiene como proceso separa una cadena de texto por medio de las comas y saltos de líneas.

```java
//Metodo para quitar la cadena
public void quitarComa(){

    GuardarDato gd= new GuardarDato();
    String fila[]= texto3.split("\n");
    for (int i = 1; i < fila.length; i++) {
       String[]palabra= fila[i].split(",");
       String nombre=palabra[0];
       double numero= Double.parseDouble(palabra[1]);
       gd.guardarDatos(nombre, numero);

    }

}
```

Opciones(): Constructor de la clase Opciones.

```java
public Opciones() {
    super();
    this.setSize(400, 490);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setBackground(Color.GRAY);
    this.setLayout(null);
    this.setLocationRelativeTo(null);
    agregarComponetes();
}
```

agregarComponetes(): Es un método en el cual se encuentran los componentes del JFrame como botones, JRadioButtons y JLabels.

```java
public void agregarComponetes() {
    //Encabezado
    JLabel etiqueta = new JLabel("Tipo de ordenamiento");
    etiqueta.setBounds(10, 10, 210, 45);
    etiqueta.setFont(new Font("arial", Font.CENTER_BASELINE, 17));
    this.add(etiqueta);

    //Opcion acendetne
    ascendente = new JRadioButton("Ascendente");
    ascendente.setBounds(10, 65, 150, 30);
    this.add(ascendente);
    //Opcion descendente
    descendente = new JRadioButton("Descendente");
    descendente.setBounds(200, 65, 150, 30);
    this.add(descendente);
    //Encabezado
    JLabel etiqueta3 = new JLabel("Velocidad de ordenamiento");
    etiqueta3.setBounds(10, 100, 210, 45);

    this.add(etiqueta3);
    //Bajo
    baja = new JRadioButton("Baja");
    baja.setBounds(10, 155, 150, 30);
    this.add(baja);
    //Medio
```

IniciarBoton(): Tiene como proceso agregarle un evento a los botones e iniciar lo que esta adentro del evento.

```java
    }
    public void  IniciarBoton(){
        ordenar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {

                try {
                    CrearGrafica();
                } catch (IOException ex) {
                    Logger.getLogger(Opciones.class.getName()).log(Level.SEVERE, null, ex);
                }


            }
        });
    }
```

CrearGrafica(): Este método tiene como proceso generar una grafica.

```
public void CrearGrafica() throws IOException{
    panel=new JPanel();
    JFrame frame2= new JFrame();
    frame2.setVisible(true);

    frame2.setLayout(null);
    frame2.setVisible(true);
    frame2.setBounds(250, 0, 700, 700);
    frame2.setBackground(Color.WHITE);
    milisigundos= new JLabel("00");
    milisigundos.setBounds(405, 10, 50, 45);
    milisigundos.setOpaque(true);
    frame2.add(milisigundos);

    //
    segundos= new JLabel("00:");
    segundos.setBounds(370, 10, 50, 45);
    segundos.setOpaque(false);
    frame2.add(segundos);

    minutos= new JLabel();
    minutos.setBounds(300, 10, 50, 45);
    panel.setBounds(10, 65, 600, 400);

    panel.setBackground(Color.BLACK);
    //
    algoritmo=new JLabel();
    algoritmo.setBounds(5, 5,250, 35);
```

agregarDimensionesDeImagenP(int ancho,int alto,String direccion): Es una función que retorna una ImageIcon y recibe como parámetro el ancho, alto y la ruta de la imagen.

Seleccionar(): Este es un constructor de la clase Seleccionar que recibe como parámetro 8 JRadioButton y 3 labels.

```
49      public Seleccionar(JRadioButton ascendente, JRadioButton descendete, JRadioButton baja, JRadioButton media, JRadioButton alta, JRadioButton bubble, JRa
50          this.ascendente = ascendente;
51          this.descendete = descendete;
52          this.baja = baja;
53          this.media = media;
54          this.alta = alta;
55          this.bubble = bubble;
56          this.quick = quick;
57          this.shell = shell;
58          this.label1 = label;
59          this.label2 = label2;
60          this.label3 = label3;
61
            seleccion();
63      }
```

Seleccion(): Tiene como proceso verificar que opción se escogió si ascendente o descendente.

```
}

public void seleccion() throws IOException {

    if (this.ascendente.isSelected() == true) {
        JOptionPane.showMessageDialog(null, "Ascendente");
        orden1="Ascendente";
        ascendente();


    } else if (this.descendete.isSelected() == true) {
        orden1="Descendente";
        JOptionPane.showMessageDialog(null, "Descendente");
        descendete();
    }

}
```

asignarVelocidad(): Tiene como proceso asignarle a la variable velocidad si la opción es baja 3000 , si es media 2000 y si es rápida 1000.

```
        }

        //Velocida
        public void asignarVelocidad() {
            velocida = 0;
            if (this.baja.isSelected() == true) {
                velocida = 3000;
                Opciones.velocidad.setText("Velocidad: baja");
                velocidadA="Baja";

            } else if (this.media.isSelected() == true) {
                velocida = 2000;
                Opciones.velocidad.setText("Velocidad: media");
                velocidadA="Media";


            } else if (this.alta.isSelected() == true) {
                velocida = 1000;
                Opciones.velocidad.setText("Velocidad: Alta");
                 velocidadA="Alta";
            }
        }
```

ascendente(): Tiene como proceso llamar a los métodos de ordenamiento BubbleSort, QuickSort y Shellsort si la opción fue ascendente.

```java
//Ascendente
public void ascendente() throws IOException {

    if (this.bubble.isSelected() == true) {
        bubbleCont=1;
        //Nombre
        Opciones.algoritmo.setText("Bubble Sort");
        nombreAlgo="Bubble Sort";
        //Velocidad
        Opciones.orden.setText("Orden: Ascendente");
        JOptionPane.showMessageDialog(null,"Algoritmo: "+"BubbleSort");
        //Llamar
        BubleSort sort = new BubleSort();
        sort.bublesSort(this.label1, this.label2, this.label3);
        asignarVelocidad();

    } else if (this.quick.isSelected() == true) {
        quicks1="quicksort";
        //Asignar nombre
           bubbleCont=1;
        Opciones.algoritmo.setText("QuickSort");
           Opciones.orden.setText("Orden: Ascendente");
           nombreAlgo="QuickSort";
           //Asignar velocidad
        asignarVelocidad();
        JOptionPane.showMessageDialog(null,"Algoritmo: "+"Quicksort");
        Quicksort quicksort= new Quicksort();
        //LLamar
           Datos quick2[]= GuardarDato.dato4;
```

descendete(): Tiene como proceso llamar a los métodos de ordenamiento descendentes BubbleSort, QuickSort y Shellsort si la opción fue descendiente.

```java
//Descendente
public void descendete() throws IOException {

    if (this.bubble.isSelected()==true) {
        bubbleCont=1;
        //Asignarle al algoritmo nombre
        Opciones.algoritmo.setText("Algoritmo: "+"Bubble Sort");
           Opciones.orden.setText("Orden: Descendente");
           nombreAlgo="Bubble Sort";
       BubleSort b= new BubleSort();
       //Asignar la velocidad
       asignarVelocidad();
       b.BubbleSortDescendente(this.label1, this.label2, this.label3);

    }else if (this.quick.isSelected()==true) {
           bubbleCont=1;
        //Asignarle al algoritmo nombre
         JOptionPane.showMessageDialog(null, "Algoritmo: "+"Quicksort");
          Opciones.orden.setText("Orden: Descendente");
             nombreAlgo="QuickSort";
       Quicksort b= new Quicksort();
     asignarVelocidad();
       //Llamar al ordenamiento
       Datos quickB[]= GuardarDato.dato5;
       b.iniciarContador();
       b.ordenarQuickSortDescendente(quickB,0,quickB.length-1);
       GuardarDato.ordenada=quickB;
       //
        mayor=quickB[0].getUfoShape();
```

bublesSort(): Tiene como proceso ordenar los datos de un arreglo de objetos por medio del metodo BubbloSort de forma ascendente.

```java
public class BubleSort {

    Datos arreglo[] = GuardarDato.dato2;
    Datos arreglo3[] = GuardarDato.dato3;

    public void bublesSort(JLabel label, JLabel label2, JLabel minutos) throws IOException {
        int v = arreglo.length;
        Seleccionar.cont = 0;
        for (int i = 0; i < v - 1; i++) {
            for (int j = 0; j < v - i - 1; j++) {
                if (arreglo[j].getCount() > arreglo[j + 1].getCount()) {

                    Datos temp = arreglo[j];
                    arreglo[j] = arreglo[j + 1];
                    arreglo[j + 1] = temp;
                    System.out.println("-----");
                    grafica(Seleccionar.cont, arreglo);
                    Seleccionar.cont++;

                }

            }

        }
```

BubbleSortDescendente(): Tiene como proceso ordenar los datos de un arreglo de objetos por medio del metodo BubbloSort de forma descendente.

```java
    //Bubble sort Descendente
    public void BubbleSortDescendente(JLabel label, JLabel label2, JLabel minutos) throws IOException {
        int v = arreglo3.length;
        Seleccionar.cont = 1;
        for (int i = 0; i < v - 1; i++) {
            for (int j = 0; j < v - i - 1; j++) {
                if (arreglo3[j + 1].getCount() > arreglo3[j].getCount()) {

                    Datos temp = arreglo3[j + 1];
                    arreglo3[j + 1] = arreglo3[j];
                    arreglo3[j] = temp;
                    System.out.println("-----");
                    grafica(Seleccionar.cont, arreglo3);
                     Seleccionar.cont++;

                }

            }

        }
        GuardarDato.ordenada = arreglo;
        //Dato menor
        Seleccionar.mayor=arreglo3[0].getUfoShape();
            Seleccionar.mayorN=String.valueOf(arreglo3[0].getCount());
        //Dato mayor
        Seleccionar.menor = arreglo3[arreglo3.length- 1].getUfoShape();
        Seleccionar.menorN = String.valueOf(arreglo3[arreglo3.length - 1].getCount());
        //
        Hilo nuev = new Hilo(Seleccionar.cont, label, label2, minutos);
```

iniciarContador(): Tiene como proceso agregarle un valor de 0 a la variable static de la cont de la clase Opciones.

```
    //Metodo de ordenamiento Quicksort
    public void iniciarContador(){
        contador=0;

        Seleccionar.cont=contador;

    }
```

ordenarQuickSort(): Metodo recursivo que tiene como proceso ordenar los datos de un arreglo de objetos por medio del metodo QuickSort de forma ascendente.

```
}
public void ordenarQuickSort(int inferior, int superior) throws IOException {

    if (superior == -1) {
        superior = 0;
    }
    double piv = quickA[superior].getCount();

    int i = inferior;
    int j = superior - 1;
    int contador = 1;
    Datos aux;
    if (inferior >= superior) {

    } else {
        while (contador == 1) {

            while (quickA[i].getCount() < piv) {

                i++;

            }
            while (quickA[j].getCount() > piv && j > 0) {
                j--;

            }

            grafica(Seleccionar.cont,quickA);
```

ordenarQuickSortDescendente(): Método recursivo que tiene como proceso ordenar los datos de un arreglo de objetos por medio del método QuickSort de forma descendete.

```java
//Metodo quicksort descendente
public void ordenarQuickSortDescendente(Datos [] arrreglo,int inferior, int superior) throws IOException {

  if (superior == -1) {
      superior = 0;
   }

   double piv = quickB[superior].getCount();

   int i = inferior;
   int j = superior - 1;
   int contador =1;
   Datos aux;

   if (inferior >=superior) {

   } else {

       while (contador == 1) {

           while (quickB[i].getCount() > piv) {

               i++;

           }

           while (quickB[j].getCount() <piv && j > 0) {
               j--;

           }
```

shellSort1(): Tiene como proceso ordenar los datos de un arreglo de objetos por medio del método ShellSort de forma ascendente.

```java
 * @author Kelly
 */
public class OrdenamientoShellSort {


    public  void shellSort1(Datos [] arreglo,JLabel label1,JLabel label2,JLabel label3) throws IOException{
        Seleccionar.cont=0;
          int salto,i,j,k;
          Datos auxiliar;
          salto=arreglo.length/2;
          while(salto>0){
              for ( i = salto; i < arreglo.length; i++) {
                  j=i-salto;
                  while (j>=0) {
                     k=j+salto;

                      if (arreglo[j].getCount()<=arreglo[k].getCount()) {
                          j=-1;
                      }else{
                          auxiliar=arreglo[j];
                          arreglo[j]=arreglo[k];
                          arreglo[k]=auxiliar;
                          j=j-salto;

                           grafica(Seleccionar.cont,arreglo);
                           Seleccionar.cont++;
                      }

                  }
              }
```

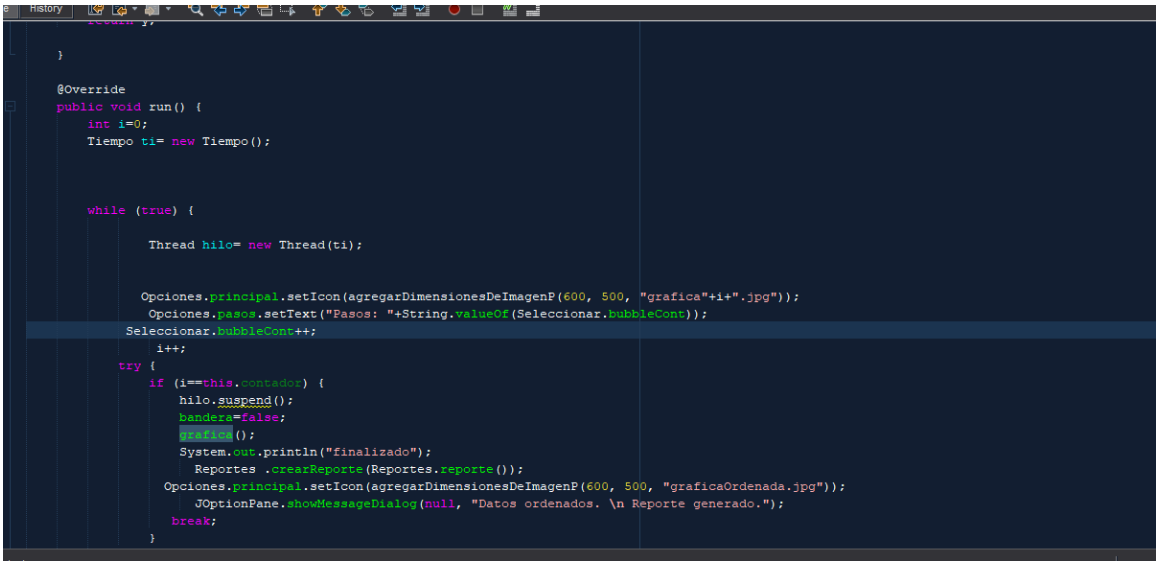
shellSort1Desc(): Tiene como proceso ordenar los datos de un arreglo de objetos por medio del método ShellSort de forma descendente.

```java
//Shell Sort descendiente
public  void shellSort1Desc(Datos [] arreglo,JLabel label,JLabel label2,JLabel label3) throws IOException{
    Seleccionar.cont=0;
      int salto,i,j,k;
      Datos auxiliar;
      salto=arreglo.length/2;
      while(salto>0){
          for ( i = salto; i < arreglo.length; i++) {
              j=i-salto;
              while (j>=0) {
                 k=j+salto;

                  if (arreglo[j].getCount()>=arreglo[k].getCount()) {
                      j=-1;
                  }else{
                      auxiliar=arreglo[j];
                      arreglo[j]=arreglo[k];
                      arreglo[k]=auxiliar;
                      j=j-salto;

                       grafica(Seleccionar.cont, arreglo);
                       Seleccionar.cont++;
                  }

              }
          }

           salto=salto/2;
      }
```

Hilo(int contador, JLabel label,JLabel mili,JLabel minutos): Constructor de la clase hilos recibe como parámetro 3 JLabels.

```java
 */
public class Hilo implements Runnable{
    int contador;
    int bubleCont;
    JLabel label;
    JLabel mili;
    JLabel minutos;
    Tiempo t;
    static boolean bandera;
   public Hilo(int contador, JLabel label,JLabel mili,JLabel minutos){
        this.contador=contador;
        this.label=label;
        this.mili=mili;
        this.minutos=minutos;
         t=new Tiempo(this.label,this.mili,minutos);
                  Thread hilo= new Thread(t);
                   bandera=true;
                  hilo.start();
   }

```

run(): El método de la clase Thread el cual tiene como proceso mostrar las imágenes de la gráfica según cada cambia y la velocidad correspondiente el hilo se detiene cuando se llega a la cantidad de pasos maxima.

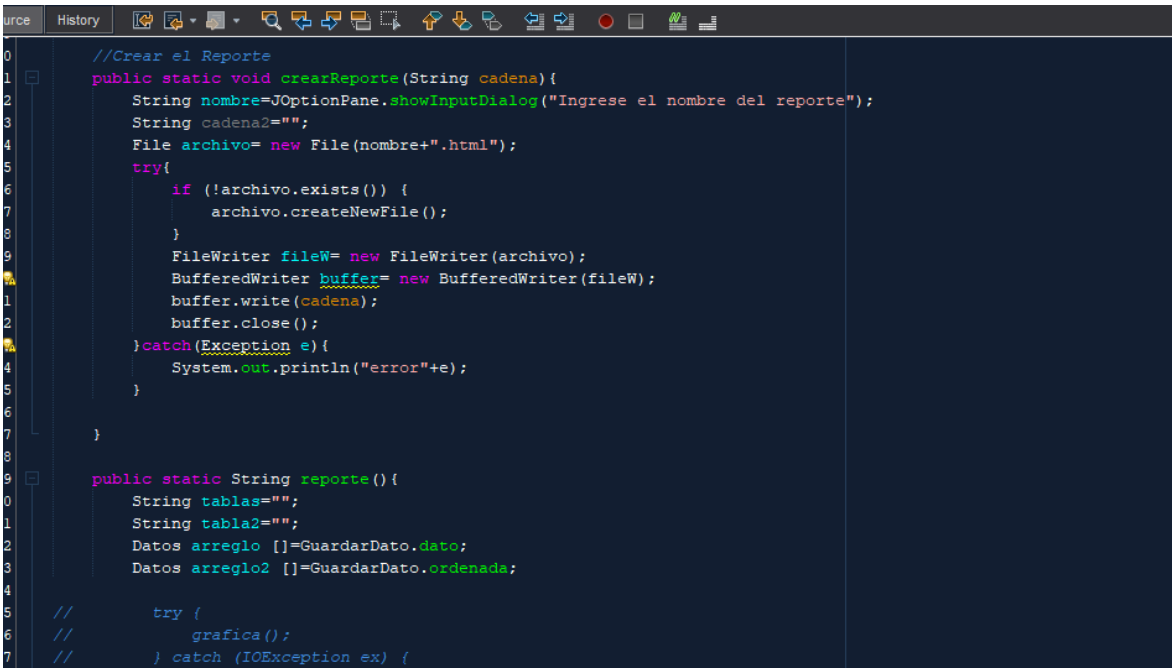


```
    }

    @Override
    public void run() {
        int i=0;
        Tiempo ti= new Tiempo();



        while (true) {

                Thread hilo= new Thread(ti);


               Opciones.principal.setIcon(agregarDimensionesDeImagenP(600, 500, "grafica"+i+".jpg"));
                Opciones.pasos.setText("Pasos: "+String.valueOf(Seleccionar.bubbleCont));
             Seleccionar.bubbleCont++;
                i++;
            try {
                if (i==this.contador) {
                    hilo.suspend();
                    bandera=false;
                    grafica();
                    System.out.println("finalizado");
                      Reportes .crearReporte(Reportes.reporte());
                  Opciones.principal.setIcon(agregarDimensionesDeImagenP(600, 500, "graficaOrdenada.jpg"));
                      JOptionPane.showMessageDialog(null, "Datos ordenados. \n Reporte generado.");
                   break;
                }
```

Tiempo(JLabel label, JLabel mili,JLabel minutos): Constructor de la clase Tiempo recibe como parámetro 3 JLabels.

```
public class Tiempo implements Runnable{
    int contador;
    JLabel label;
    JLabel mili;
    JLabel minutos;
   int segundos;
    public static String tiempo;
    public static String tiempo2;
    public static String tiempo3="00";
    public Tiempo(){

    }
     public Tiempo(JLabel label, JLabel mili,JLabel minutos) {
        this.label = label;
        this.mili = mili;
        this.minutos=minutos;

     }

     @Override
     public void run() {
        int i=0;
        segundos=0;
        int minutos1=0;
        int milisegundos=0;
       while(Hilo.bandera){

            try {
                Thread.sleep(10);
                 milisegundos+=10;
```

crearReporte(String cadena): Método de tipo void recibe como parámetro un String , tiene como proceso el generar un archivo html.



```
urce  History
    //Crear el Reporte
    public static void crearReporte(String cadena){
        String nombre=JOptionPane.showInputDialog("Ingrese el nombre del reporte");
        String cadena2="";
        File archivo= new File(nombre+".html");
        try{
            if (!archivo.exists()) {
                archivo.createNewFile();
            }
            FileWriter fileW= new FileWriter(archivo);
            BufferedWriter buffer= new BufferedWriter(fileW);
            buffer.write(cadena);
            buffer.close();
        }catch(Exception e){
            System.out.println("error"+e);
        }

    }

    public static String reporte(){
        String tablas="";
        String tabla2="";
        Datos arreglo []=GuardarDato.dato;
        Datos arreglo2 []=GuardarDato.ordenada;

//        try {
//            grafica();
//        } catch (IOException ex) {
```

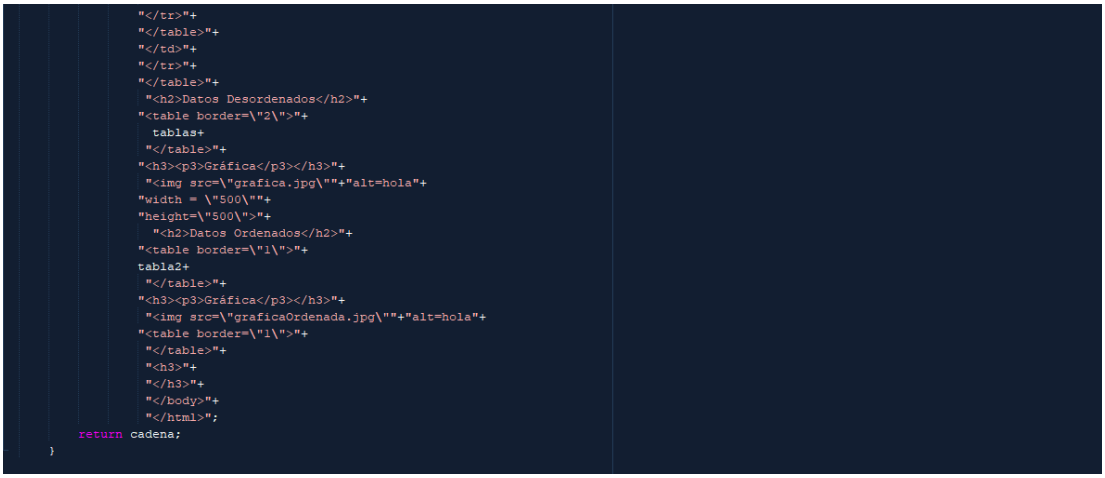
reporte(): Es una función que devuelve una String la cual almacena la estructura de un archivo html.

```
    public static String reporte(){
        String tablas="";
        String tabla2="";
        Datos arreglo []=GuardarDato.dato;
        Datos arreglo2 []=GuardarDato.ordenada;

//        try {
//            grafica();
//        } catch (IOException ex) {
//
//        }

        tablas+="<tr>";
         tablas += "<th>"+VentanaPrincipal.tituloX+"</th>";
        for (int i = 0; i < arreglo.length; i++) {

          if (arreglo[i]!=null) {

              tablas += "<th>"+arreglo[i].getUfoShape()+"</th>";

          }

        }
         tablas += "</tr>";
         tablas+="<tr>";
          tablas += "<th>"+VentanaPrincipal.tituloY+"</th>";
        for (int i = 0; i < arreglo.length; i++) {
```

```
"<head>"+
"<meta charset=\"UTF-8\">"+
"<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">"+
"<style>"+
"body{"+
"font-size: 20px;"+
"}"+
"h1{"+
"size: 108px;"+
"text-align: center;"+
"}"+
"</style>"+
"<title>Reporte</title>"+
"</head>"+
"<body>"+
"<style>"+
"body{"+
"background-image: url(https://static.vix.com/es/sites/default/files/styles/1x1/public/btg/tech.batanga.com/files/Fondos-para-pag
"background-size: 100%;"+
"}"+
"</style>"+
"<header><h2>Kelly Mischel Herrera Espino </h2></header>"+
"<header><h2>201900716 </h2></header>"+
"<style>"+
"h2{"+
"color: blue;"+
"text-align: center;"+
"}"+
```

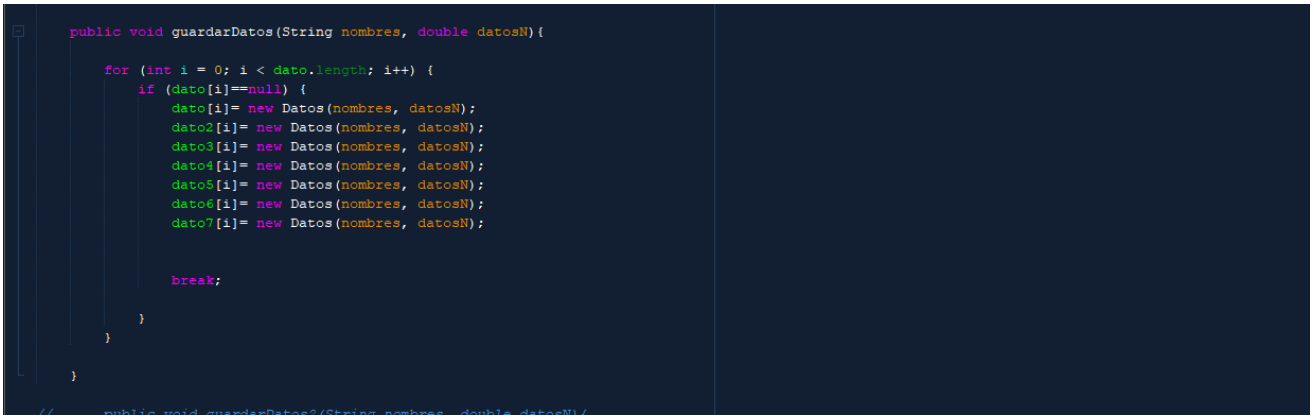
```
"text-align: center;"+
"}"+
"</style>"+
"<style>"+
"h3{"+
"color:darkblue;"+
"text-align: center;"+
"}"+
"</style>"+
"<style>"+
"table{"+
"text-align: center;"+
"margin: auto;"+
"}"+
"</style>"+
"<style>"+
"table{"+
"text-align: center;"+
"margin: auto;"+
"width: 25%;"+
"height: 25%;"+
"color: black:"+
"}"+
"</style>"+
"<style>"+
"#columnas{"+
"text-align: center;"+
"column-count:2;"+
"column-gap:20px;"+
"column-rule:4px dotted gray;"+
```

```
"margin-left: auto;"+
"margin-right:auto;"+
"}"+
"</style>"+
 "<div id=\"columnas\">"+
"<b><ol>Algoritmo: "+Seleccionar.nombreAlgo+"</ol></b>"+
"<b><ol>Velocidad: "+Seleccionar.velocidadA+"</ol></b>"+
"<b><ol>Orden: "+Seleccionar.orden1+"</ol></b>"+
"<b><ol>Tiempo :"+Tiempo.tiempo3+":"+Tiempo.tiempo2+":"+Tiempo.tiempo+"</ol></b>"+
"<b><ol>Pasos:"+(Seleccionar.bubbleCont-1)+"</ol></b>"+
"</div>"+
 "<table id=\"separar\">"+
 "<tr>"+
 "<td>"+
"<table id=\"mayor\" style=\"width:20%\" border=\"4\">"+
"<tr>"+
 "<th colspan = \"3\">Valor Minimo</th>"+
"</tr>"+
"<tr>"+
 "<td>"+Seleccionar.menor+"</td>"+
 "<td>"+Seleccionar.menorN+"</td>"+
"</tr>"+
"</table>"+
"</td>"+
"<td id=\"s\">"+
"<table id=\"mayor\" style=\"width:20%\" border=\"4\">"+
"<tr>"+
```



```
            "</tr>"+
            "</table>"+
            "</td>"+
            "</tr>"+
            "</table>"+
             "<h2>Datos Desordenados</h2>"+
            "<table border=\"2\">"+
              tablas+
             "</table>"+
            "<h3><p3>Gráfica</p3></h3>"+
             "<img src=\"grafica.jpg\""+"alt=hola"+
            "width = \"500\""+
            "height=\"500\">"+
              "<h2>Datos Ordenados</h2>"+
            "<table border=\"1\">"+
            tabla2+
             "</table>"+
            "<h3><p3>Gráfica</p3></h3>"+
             "<img src=\"graficaOrdenada.jpg\""+"alt=hola"+
            "<table border=\"1\">"+
             "</table>"+
             "<h3>"+
             "</h3>"+
             "</body>"+
             "</html>";
     return cadena;
 }
```

guardarDatos(): Tiene como proceso generar objetos de tipo Datos.



```
public void guardarDatos(String nombres, double datosN){

    for (int i = 0; i < dato.length; i++) {
        if (dato[i]==null) {
            dato[i]= new Datos(nombres, datosN);
            dato2[i]= new Datos(nombres, datosN);
            dato3[i]= new Datos(nombres, datosN);
            dato4[i]= new Datos(nombres, datosN);
            dato5[i]= new Datos(nombres, datosN);
            dato6[i]= new Datos(nombres, datosN);
            dato7[i]= new Datos(nombres, datosN);


            break;

        }
    }

}
```

Datos(): Constructor de la clase Datos que recibe como parámetro un entero y un String.

```
     * @return the count
     */
    public double getCount() {
        return count;
    }

    /**
     * @param count the count to set
     */
    public void setCount(int count) {
        this.count = count;
    }

    public void imprimir(){
        System.out.println("n: "+this.getUfoShape());
        System.out.println("numero "+this.getCount());
    }

}
```

Método Main:

Este es el metodo principal el cual se encuentra en la clase Practiaca2. En este metodo se llama a la clase VentanaPrincipal.

```
 */
package practica2;

import Ventanas.VentanaPrincipal;

/**
 *
 * @author Kelly
 */
public class Practica2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        VentanaPrincipal ven = new VentanaPrincipal();
        ven.show();
    }

}
```

Clases

- **BubbleSort**
- **Datos**
- **GuardarDatos**
- **Hilo**
- **Opciones**
- **OrdenamientoShellSort**
- **QuickSort**
- **Reportes**
- **Seleccionar**
- **Tiempo**
- **VentanaPrincipal**
- **Practiaca2**

Packages

- **Ventanas**
- **Practica2**