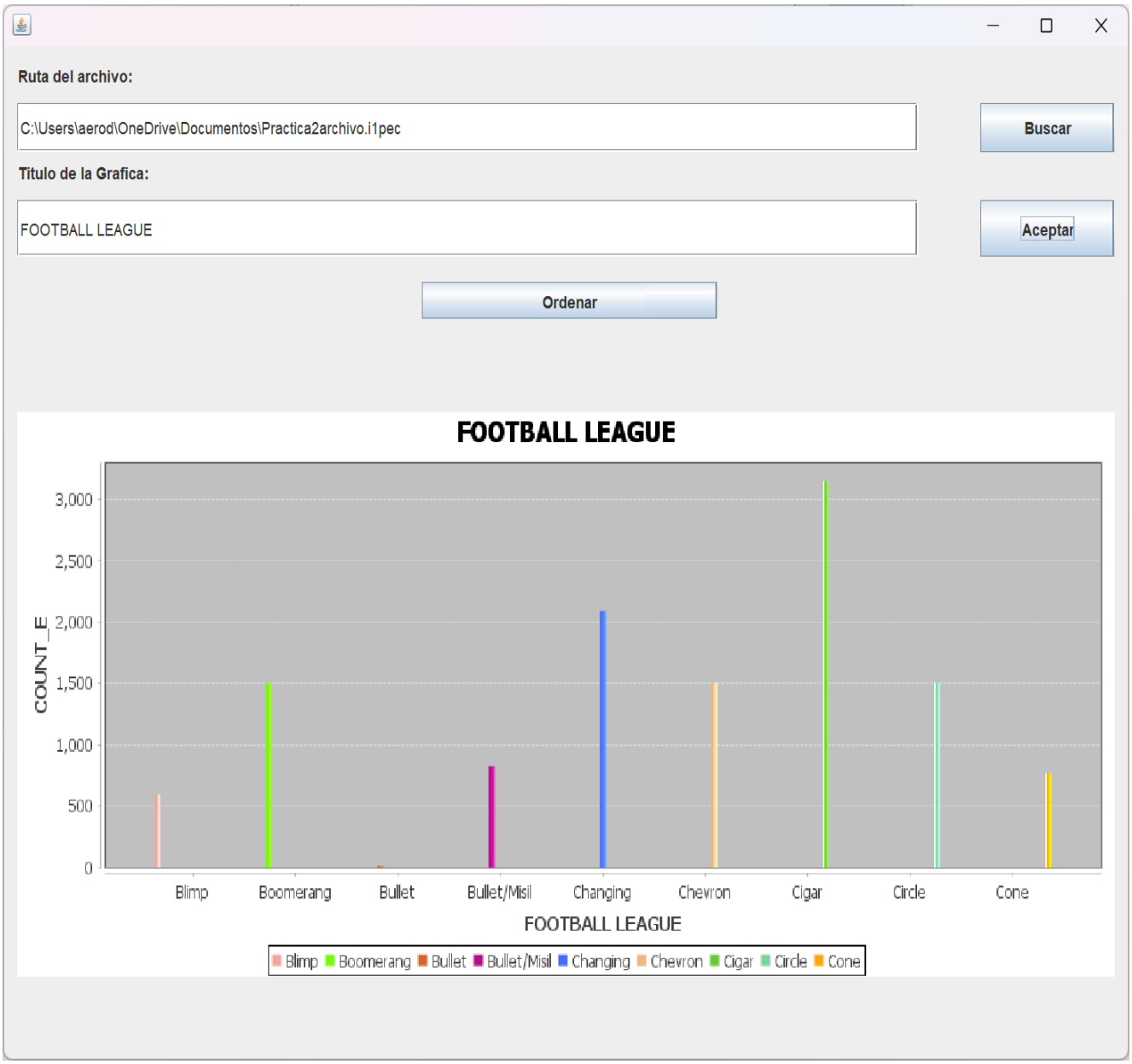
Manual Tecnico – USAC Processing Dat



Autor: Angel Emanuel Rodriguez Corado

Carne: 202404856

Fecha: 31/03/2025

Índice

Introducción 3
Objetivos 3
Dirigido 3
Especificación Técnica 3
Requisitos de Hardware 3
Requisitos de Software 4
Lógica del Programa 4
- Modelo 4
- Vista 4
- Controlador 4
- Librerías 4

Main 6
Modelo 6
Controlador 8
Atributos 14
Métodos 15
Métodos de Ordenamiento 16
Reporte 18
Vista 20
FrmPrincipal 20
FrmOpciones 21
FrmOrdenar 22
Diagrama de Flujo 23

Introducción

Este manual técnico proporciona una descripción detallada de la implementación de la aplicación USAC Processing Data. Se analizarán los métodos y estructuras utilizados en el código, explicando su propósito y funcionamiento completo.

Objetivos

- Explicar la funcionalidad del código completo con el que funciona la aplicación.
- Describir el flujo de datos en la aplicación.
- Facilitar la comprensión del código para futuros desarrolladores que quieran utilizar el código de esta aplicación.

Dirigido

Este manual está orientado a todas aquellas personas que tienen un interés por como es

el funcionamiento del desarrollo de esta aplicación para poder utilizar como idea esta práctica para futuros proyectos.

Especificación Técnica

Requisitos de Hardware

- Computadora de escritorio o portátil.
- Mínimo 2GB de Memoria RAM.
- Procesador Intel Core i3 en adelante.
- 100GB de Disco Duro en adelante.
- Resolución grafica mínima de 1280x720.

Requisitos de Software

- Sistema Operativo Windows 7 en adelante.
- Java Runtime Enviroment (JRE) versión 8.2 en adelante.
- Tener instalado Java Development Kit (JDK) versión 8.2 o superior.
- Lenguaje de Programación JAVA.
- NetBeans IDE 8.2 o superior.
- Librería interna java.util
- Librería interna jfreechart-1.0.19
- Librería interna iText-2.0.8

Lógica del Programa

La estructura del la aplicación sigue el Modelo-Vista-Controlador (MVC), separando claramente el manejo de datos, la interfaz de usuario y la gestión de eventos.

- ### Modelo (Manejo de Datos)

Ubicado en la carpeta Modelo, contiene clases que representan la lógica y los datos de la aplicación.

- ### Vista (Interfaz Gráfica)

Ubicada en la carpeta vista, contiene los formularios (Frm*.java), los cuales representan las pantallas de la aplicación.

- ### Controlador (Gestión de Eventos)

Ubicado en la carpeta Controlador, maneja la comunicación entre la vista y el modelo.

- ### Librerías

La aplicación utiliza las siguientes librerías externas, además del JDK 20 por defecto:

- iText 2.0.8

  Propósito: Permite la generación y manipulación de archivos PDF.

  Uso en la aplicación: Se utiliza en ReporteModelo.java para crear los reportes de transacciones, depósitos y retiros en formato PDF.

- Jfreechart 1.0.19

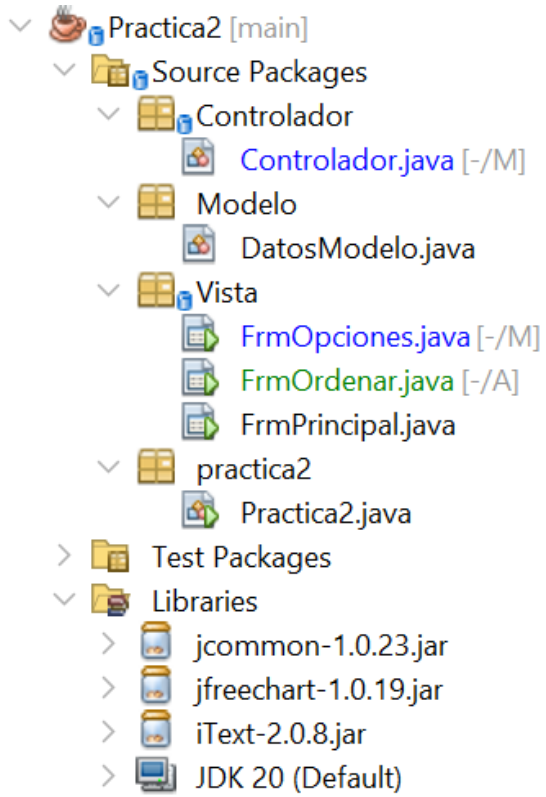
  Propósito: Genera graficas de diferente tipo y estilos con el manejo de datos

  Uso en la aplicación: Es utilizado para generar una grafica de barras en base a los datos de que se encuentran en el archivo leído.

- Absolute Layout (AbsoluteLayout.jar)

  Propósito: Proporciona un diseño de posicionamiento absoluto para los componentes en las interfaces gráficas.

  Uso en la aplicación: Es utilizado en los formularios (Frm*.java) dentro del paquete vista para organizar los elementos de la interfaz gráfica de manera precisa.



Main

```java
package practica2;

import Controlador.Controlador;
import Vista.FrmPrincipal;
import javax.swing.SwingUtilities;

public class Practica2 {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            FrmPrincipal vista = new FrmPrincipal();
            Controlador controlador = new Controlador(vista); // Guardamos la referencia del controlador
            vista.setVisible(b: true);
        });
    }
}
```

- Es el punto de entrada del programa.
- SwingUtilities.invokeLater() garantiza que la interfaz gráfica (FrmPrincipal) se ejecute en el hilo de eventos de Swing, evitando bloqueos en la ejecución.
- Se crea una instancia de FrmPrincipal (ventana principal de la aplicación).
- Se instancia el Controlador, pasándole la vista como argumento para manejar la interacción.
- Finalmente, se hace visible la ventana principal con vista.setVisible(true);.

Modelo

```java
package Modelo;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class DatosModelo {
    private List<String> categoriasX; // Nombres de la primera columna
    private List<Integer> valoresY;   // Valores numéricos de la segunda columna
    private String tituloX;
    private String tituloY;

    public DatosModelo() {
        categoriasX = new ArrayList<>();
        valoresY = new ArrayList<>();
    }

    public boolean cargarDesdeArchivo(File archivo) {
    try (BufferedReader br = new BufferedReader(new FileReader(file: archivo))) {
        String linea = br.readLine(); // Leer la primera línea (encabezados)
        if (linea == null) {
            System.out.println(x: "El archivo está vacío.");
            return false;
        }

        String[] encabezados = linea.split(regex: ",");
        if (encabezados.length < 2) {
            System.out.println(x: "El archivo no tiene el formato correcto.");
            return false;
        }

        tituloX = encabezados[0];
        tituloY = encabezados[1];

        categoriasX.clear();
        valoresY.clear();

        String lineaDatos;
        while ((lineaDatos = br.readLine()) != null) {
            String[] partes = lineaDatos.split(regex: ",");
            if (partes.length < 2) continue;

            try {
                categoriasX.add(partes[0]); // Nombre de la categoría
                valoresY.add(e: Integer.parseInt(partes[1])); // Valor numérico
            } catch (NumberFormatException e) {
                System.out.println("Error al convertir: " + partes[1]);
                return false;
```

```
            }
        }

        return true; // Indica que la carga fue exitosa
    } catch (Exception e) {
        System.out.println("Error al leer el archivo: " + e.getMessage());
        return false;
    }
}


public List<String> getCategoriasX() {
    return categoriasX;
}

public List<Integer> getValoresY() {
    return valoresY;
}

public String getTituloX() {
    return tituloX;
}

public String getTituloY() {
    return tituloY;
}
}

```

Clase DatosModelo

Esta clase maneja la carga y almacenamiento de datos desde un archivo.

Atributos

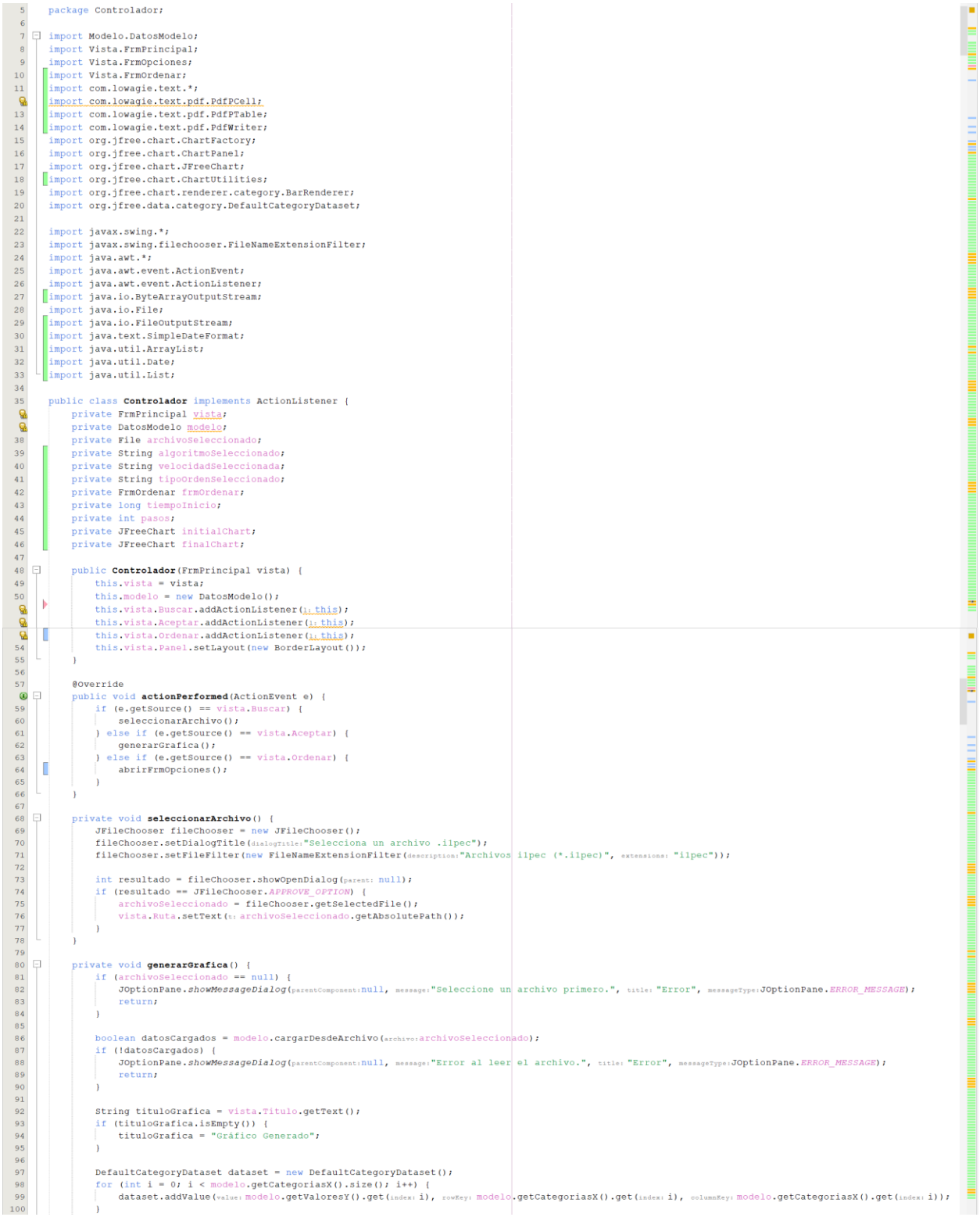
- List<String> categoriasX → Guarda los nombres de las categorías.
- List<Integer> valoresY → Guarda los valores numéricos asociados a las categorías.
- String tituloX → Almacena el título de la primera columna.
- String tituloY → Almacena el título de la segunda columna.

Métodos

- DatosModelo() → Constructor que inicializa las listas categoriasX y valoresY.
- boolean cargarDesdeArchivo(File archivo) → Carga datos desde un archivo CSV, extrayendo títulos y valores numéricos. Retorna true si la carga es exitosa, false si hay errores.
- List<String> getCategoriasX() → Devuelve la lista de categorías (categoriasX).
- List<Integer> getValoresY() → Devuelve la lista de valores numéricos (valoresY).
- String getTituloX() → Retorna el título de la columna X.

- String getTituloY() → Retorna el título de la columna Y.

Controlador



```java
package Controlador;

import Modelo.DatosModelo;
import Vista.FrmPrincipal;
import Vista.FrmOpciones;
import Vista.FrmOrdenar;
import com.lowagie.text.*;
import com.lowagie.text.pdf.PdfPCell;
import com.lowagie.text.pdf.PdfPTable;
import com.lowagie.text.pdf.PdfWriter;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.renderer.category.BarRenderer;
import org.jfree.data.category.DefaultCategoryDataset;

import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Controlador implements ActionListener {
    private FrmPrincipal vista;
    private DatosModelo modelo;
    private File archivoSeleccionado;
    private String algoritmoSeleccionado;
    private String velocidadSeleccionada;
    private String tipoOrdenSeleccionado;
    private FrmOrdenar frmOrdenar;
    private long tiempoInicio;
    private int pasos;
    private JFreeChart initialChart;
    private JFreeChart finalChart;

    public Controlador(FrmPrincipal vista) {
        this.vista = vista;
        this.modelo = new DatosModelo();
        this.vista.Buscar.addActionListener(l: this);
        this.vista.Aceptar.addActionListener(l: this);
        this.vista.Ordenar.addActionListener(l: this);
        this.vista.Panel.setLayout(new BorderLayout());
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == vista.Buscar) {
            seleccionarArchivo();
        } else if (e.getSource() == vista.Aceptar) {
            generarGrafica();
        } else if (e.getSource() == vista.Ordenar) {
            abrirFrmOpciones();
        }
    }

    private void seleccionarArchivo() {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setDialogTitle(dialogTitle:"Selecciona un archivo .ilpec");
        fileChooser.setFileFilter(new FileNameExtensionFilter(description:"Archivos ilpec (*.ilpec)", extensions: "ilpec"));

        int resultado = fileChooser.showOpenDialog(parent: null);
        if (resultado == JFileChooser.APPROVE_OPTION) {
            archivoSeleccionado = fileChooser.getSelectedFile();
            vista.Ruta.setText(t: archivoSeleccionado.getAbsolutePath());
        }
    }

    private void generarGrafica() {
        if (archivoSeleccionado == null) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Seleccione un archivo primero.", title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
            return;
        }

        boolean datosCargados = modelo.cargarDesdeArchivo(archivo:archivoSeleccionado);
        if (!datosCargados) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Error al leer el archivo.", title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
            return;
        }

        String tituloGrafica = vista.Titulo.getText();
        if (tituloGrafica.isEmpty()) {
            tituloGrafica = "Gráfico Generado";
        }

        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        for (int i = 0; i < modelo.getCategoriasX().size(); i++) {
            dataset.addValue(value: modelo.getValoresY().get(index: i), rowKey: modelo.getCategoriasX().get(index: i), columnKey: modelo.getCategoriasX().get(index: i));
        }
```

```
        initialChart = ChartFactory.createBarChart(
                title: tituloGrafica,
                categoryAxisLabel: modelo.getTituloX(),
                valueAxisLabel: modelo.getTituloY(),
                dataset
        );

        BarRenderer renderer = (BarRenderer) initialChart.getCategoryPlot().getRenderer();
        for (int i = 0; i < modelo.getCategoriasX().size(); i++) {
            renderer.setSeriesPaint(series: i, new Color((int) (Math.random() * 255), (int) (Math.random() * 255), (int) (Math.random() * 255)));
        }

        ChartPanel chartPanel = new ChartPanel(chart: initialChart);
        chartPanel.setPreferredSize(new Dimension(width: 500, height: 400));

        vista.Panel.removeAll();
        vista.Panel.add(comp: chartPanel, constraints:BorderLayout.CENTER);
        vista.Panel.revalidate();
        vista.Panel.repaint();
    }

    private void abrirFrmOpciones() {
        FrmOpciones opciones = new FrmOpciones();
        opciones.setVisible(b: true);

        opciones.Ordenar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                obtenerOpcionesOrdenamiento(opciones);
                opciones.setVisible(b: false);
                abrirFrmOrdenar();
            }
        });

        opciones.Cancelar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                opciones.setVisible(b: false);
            }
        });
    }

    private void obtenerOpcionesOrdenamiento(FrmOpciones opciones) {
        if (opciones.Bubble.isSelected()) {
            algoritmoSeleccionado = "Bubble";
        } else if (opciones.Insert.isSelected()) {
            algoritmoSeleccionado = "Insert";
```

```
        } else if (opciones.Merge.isSelected()) {
            algoritmoSeleccionado = "Merge";
        } else if (opciones.Quick.isSelected()) {
            algoritmoSeleccionado = "Quick";
        } else if (opciones.Select.isSelected()) {
            algoritmoSeleccionado = "Select";
        } else if (opciones.Shell.isSelected()) {
            algoritmoSeleccionado = "Shell";
        }

        tipoOrdenSeleccionado = opciones.Ascendente.isSelected() ? "Ascendente" : "Descendente";

        if (opciones.Alta.isSelected()) {
            velocidadSeleccionada = "Alta";
        } else if (opciones.Media.isSelected()) {
            velocidadSeleccionada = "Media";
        } else {
            velocidadSeleccionada = "Baja";
        }
    }

    private void abrirFrmOrdenar() {
        frmOrdenar = new FrmOrdenar();
        frmOrdenar.setVisible(b: true);
        generarGrafica();

        frmOrdenar.Algoritmo.setText("Algoritmo: " + algoritmoSeleccionado);
        frmOrdenar.Velocidad.setText("Velocidad: " + velocidadSeleccionada);
        frmOrdenar.Orden.setText("Orden: " + tipoOrdenSeleccionado);

        new Thread(() -> {
            tiempoInicio = System.currentTimeMillis();
            pasos = 0;

            switch (algoritmoSeleccionado) {
                case "Bubble":
                    bubbleSort(frmOrdenar);
                    break;
                case "Insert":
                    insertSort(frmOrdenar);
                    break;
                case "Merge":
                    mergeSort(frmOrdenar);
                    break;
                case "Quick":
                    quickSort(frmOrdenar);
                    break;
                case "Select":
```

```
                selectSort(frmOrdenar);
                break;
            case "Shell":
                shellSort(frmOrdenar);
                break;
        }

        generarPDF(datosOrdenados: modelo.getValoresY()); // Generar el PDF después de ordenar
    }).start();
}

private void actualizarLabels(FrmOrdenar frmOrdenar, long tiempoTranscurrido, int pasos) {
    if (frmOrdenar == null) return;

    SimpleDateFormat sdf = new SimpleDateFormat(pattern:"mm:ss:SSS");
    String tiempoFormateado = sdf.format(new Date(date:tiempoTranscurrido));

    frmOrdenar.Tiempo.setText("Tiempo: " + tiempoFormateado);
    frmOrdenar.Pasos.setText("Pasos: " + pasos);
}

private void bubbleSort(FrmOrdenar frmOrdenar) {
    List<Integer> datos = modelo.getValoresY();
    int n = datos.size();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if ((tipoOrdenSeleccionado.equals(anObject:"Ascendente") && datos.get(index:j) > datos.get(j + 1)) ||
                (tipoOrdenSeleccionado.equals(anObject:"Descendente") && datos.get(index:j) < datos.get(j + 1))) {
                int temp = datos.get(index:j);
                datos.set(index:j, element:datos.get(j + 1));
                datos.set(j + 1, element:temp);

                pasos++;
                long tiempoTranscurrido = System.currentTimeMillis() - tiempoInicio;
                SwingUtilities.invokeLater(() -> {
                    actualizarGraficaPasoAPaso(datos);
                    actualizarLabels(frmOrdenar, tiempoTranscurrido, pasos);
                });
            }

            try {
                if (velocidadSeleccionada.equals(anObject:"Alta")) {
                    Thread.sleep(millis:100);
                } else if (velocidadSeleccionada.equals(anObject:"Media")) {
                    Thread.sleep(millis:200);
                } else {
                    Thread.sleep(millis:500);
                }
```

```
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            }
        }
    }

    private void insertSort(FrmOrdenar frmOrdenar) {
        List<Integer> datos = modelo.getValoresY();
        for (int i = 1; i < datos.size(); i++) {
            int key = datos.get(index:i);
            int j = i - 1;
            while (j >= 0 && ((tipoOrdenSeleccionado.equals(anObject:"Ascendente") && datos.get(index:j) > key) ||
                   (tipoOrdenSeleccionado.equals(anObject:"Descendente") && datos.get(index:j) < key))) {
                datos.set(j + 1, element:datos.get(index:j));
                j--;
            }
            datos.set(j + 1, element:key);

            pasos++;
            long tiempoTranscurrido = System.currentTimeMillis() - tiempoInicio;
            SwingUtilities.invokeLater(() -> {
                actualizarGraficaPasoAPaso(datos);
                actualizarLabels(frmOrdenar, tiempoTranscurrido, pasos);
            });

            try {
                if (velocidadSeleccionada.equals(anObject:"Alta")) {
                    Thread.sleep(millis:100);
                } else if (velocidadSeleccionada.equals(anObject:"Media")) {
                    Thread.sleep(millis:200);
                } else {
                    Thread.sleep(millis:500);
                }
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }

    private void quickSort(FrmOrdenar frmOrdenar) {
        List<Integer> datos = modelo.getValoresY();
        quickSortHelper(datos, low:0, datos.size() - 1, frmOrdenar);
    }

    private void quickSortHelper(List<Integer> datos, int low, int high, FrmOrdenar frmOrdenar) {
        if (low < high) {
            int pivotIndex = partition(datos, low, high, frmOrdenar);
```

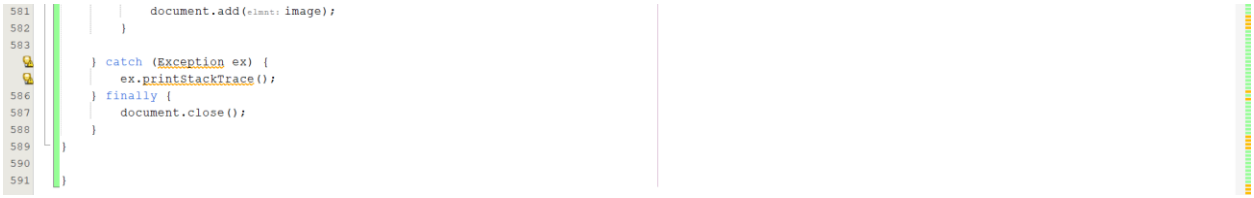
```
        quickSortHelper(datos, low, pivotIndex - 1, frmOrdenar);
        quickSortHelper(datos, pivotIndex + 1, high, frmOrdenar);
    }
}

private int partition(List<Integer> datos, int low, int high, FrmOrdenar frmOrdenar) {
    int pivot = datos.get(index: high);
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if ((tipoOrdenSeleccionado.equals(anObject: "Ascendente") && datos.get(index: j) <= pivot) ||
                (tipoOrdenSeleccionado.equals(anObject: "Descendente") && datos.get(index: j) >= pivot)) {
            i++;
            int temp = datos.get(index: i);
            datos.set(index: i, element:datos.get(index: j));
            datos.set(index: j, element:temp);

            pasos++;
            long tiempoTranscurrido = System.currentTimeMillis() - tiempoInicio;
            SwingUtilities.invokeLater(() -> actualizarLabels(frmOrdenar, tiempoTranscurrido, pasos));
        }
    }

    int temp = datos.get(i + 1);
    datos.set(i + 1, element:datos.get(index: high));
    datos.set(index: high, element:temp);

    pasos++;
    long tiempoTranscurrido = System.currentTimeMillis() - tiempoInicio;
    SwingUtilities.invokeLater(() -> {
        actualizarGraficaPasoAPaso(datos);
        actualizarLabels(frmOrdenar, tiempoTranscurrido, pasos);
    });

    try {
        if (velocidadSeleccionada.equals(anObject: "Alta")) {
            Thread.sleep(millis: 100);
        } else if (velocidadSeleccionada.equals(anObject: "Media")) {
            Thread.sleep(millis: 200);
        } else {
            Thread.sleep(millis: 500);
        }
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }

    return i + 1;
}
```

```
private void selectSort(FrmOrdenar frmOrdenar) {
    List<Integer> datos = modelo.getValoresY();
    for (int i = 0; i < datos.size() - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < datos.size(); j++) {
            if ((tipoOrdenSeleccionado.equals(anObject: "Ascendente") && datos.get(index: j) < datos.get(index: minIndex)) ||
                    (tipoOrdenSeleccionado.equals(anObject: "Descendente") && datos.get(index: j) > datos.get(index: minIndex))) {
                minIndex = j;
            }
        }

        int temp = datos.get(index: i);
        datos.set(index: i, element:datos.get(index: minIndex));
        datos.set(index: minIndex, element:temp);

        pasos++;
        long tiempoTranscurrido = System.currentTimeMillis() - tiempoInicio;
        SwingUtilities.invokeLater(() -> {
            actualizarGraficaPasoAPaso(datos);
            actualizarLabels(frmOrdenar, tiempoTranscurrido, pasos);
        });

        try {
            if (velocidadSeleccionada.equals(anObject: "Alta")) {
                Thread.sleep(millis: 100);
            } else if (velocidadSeleccionada.equals(anObject: "Media")) {
                Thread.sleep(millis: 200);
            } else {
                Thread.sleep(millis: 500);
            }
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
}

private void shellSort(FrmOrdenar frmOrdenar) {
    List<Integer> datos = modelo.getValoresY();
    int n = datos.size();
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = datos.get(index: i);
            int j = i;
            while (j >= gap && ((tipoOrdenSeleccionado.equals(anObject: "Ascendente") && datos.get(j - gap) > temp) ||
                    (tipoOrdenSeleccionado.equals(anObject: "Descendente") && datos.get(j - gap) < temp))) {
                datos.set(index: j, element:datos.get(j - gap));
                j -= gap;
```

```
                }
                datos.set(index: j, element:temp);

                pasos++;
                long tiempoTranscurrido = System.currentTimeMillis() - tiempoInicio;
                SwingUtilities.invokeLater(() -> {
                    actualizarGraficaPasoAPaso(datos);
                    actualizarLabels(frmOrdenar, tiempoTranscurrido, pasos);
                });

                try {
                    if (velocidadSeleccionada.equals(anObject: "Alta")) {
                        Thread.sleep(millis: 100);
                    } else if (velocidadSeleccionada.equals(anObject: "Media")) {
                        Thread.sleep(millis: 200);
                    } else {
                        Thread.sleep(millis: 500);
                    }
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            }
        }
    }

    private void mergeSort(FrmOrdenar frmOrdenar) {
        List<Integer> datos = modelo.getValoresY();
        mergeSortHelper(datos, left: 0, datos.size() - 1, frmOrdenar);
    }

    private void mergeSortHelper(List<Integer> datos, int left, int right, FrmOrdenar frmOrdenar) {
        if (left < right) {
            int mid = (left + right) / 2;

            mergeSortHelper(datos, left, right: mid, frmOrdenar);
            mergeSortHelper(datos, mid + 1, right, frmOrdenar);

            merge(datos, left, mid, right, originalDatos: datos, frmOrdenar);
        }
    }

    private void merge(List<Integer> datos, int left, int mid, int right, List<Integer> originalDatos, FrmOrdenar frmOrdenar) {
        List<Integer> temp = new ArrayList<>();
        int i = left, j = mid + 1;

        while (i <= mid && j <= right) {
            if ((tipoOrdenSeleccionado.equals(anObject: "Ascendente") && datos.get(index: i) <= datos.get(index: j)) ||
                (tipoOrdenSeleccionado.equals(anObject: "Descendente") && datos.get(index: i) >= datos.get(index: j))) {
```

```
                temp.add(e: datos.get(index: i));
                i++;
            } else {
                temp.add(e: datos.get(index: j));
                j++;
            }
        }

        while (i <= mid) {
            temp.add(e: datos.get(index: i));
            i++;
        }

        while (j <= right) {
            temp.add(e: datos.get(index: j));
            j++;
        }

        for (int k = left; k <= right; k++) {
            datos.set(index: k, element:temp.get(k - left));

            pasos++;
            long tiempoTranscurrido = System.currentTimeMillis() - tiempoInicio;
            SwingUtilities.invokeLater(() -> {
                actualizarGraficaPasoAPaso(datos: originalDatos);
                actualizarLabels(frmOrdenar, tiempoTranscurrido, pasos);
            });

            try {
                if (velocidadSeleccionada.equals(anObject: "Alta")) {
                    Thread.sleep(millis: 100);
                } else if (velocidadSeleccionada.equals(anObject: "Media")) {
                    Thread.sleep(millis: 200);
                } else {
                    Thread.sleep(millis: 500);
                }
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }

    private void actualizarGraficaPasoAPaso(List<Integer> datos) {
        if (frmOrdenar == null) return;

        String tituloGrafica = vista.Titulo.getText().trim();
        if (tituloGrafica.isEmpty()) {
            tituloGrafica = "Gráfico Ordenado";
```

```
        }

        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        for (int i = 0; i < modelo.getCategoriasX().size(); i++) {
            dataset.addValue(value: datos.get(index: i), rowKey: modelo.getCategoriasX().get(index: i), columnKey: modelo.getCategoriasX().get(index: i));
        }

        finalChart = ChartFactory.createBarChart(
                title: tituloGrafica,
                categoryAxisLabel: modelo.getTituloX(),
                valueAxisLabel: modelo.getTituloY(),
                dataset
        );

        BarRenderer renderer = (BarRenderer) finalChart.getCategoryPlot().getRenderer();
        for (int i = 0; i < modelo.getCategoriasX().size(); i++) {
            renderer.setSeriesPaint(series: i, new Color((int) (Math.random() * 255), (int) (Math.random() * 255), (int) (Math.random() * 255)));
        }

        ChartPanel chartPanel = new ChartPanel(chart: finalChart);
        chartPanel.setPreferredSize(new Dimension(width: 500, height: 400));

        frmOrdenar.Mostrar.removeAll();
        frmOrdenar.Mostrar.add(comp: chartPanel, constraints:BorderLayout.CENTER);
        frmOrdenar.Mostrar.revalidate();
        frmOrdenar.Mostrar.repaint();
    }

    private void generarPDF(List<Integer> datosOrdenados) {
    Document document = new Document();
    try {
        PdfWriter.getInstance(dcmnt: document, new FileOutputStream(name: "Reporte_Ordenamiento.pdf"));
        document.open();

        // Información del estudiante
        document.add(new Paragraph(string: "Angel Emanuel Rodriguez Corado"));
        document.add(new Paragraph(string: "202404856"));
        document.add(elmnt: Chunk.NEWLINE);

        // Información del ordenamiento
        document.add(new Paragraph("Algoritmo: " + algoritmoSeleccionado));
        document.add(new Paragraph("Velocidad: " + velocidadSeleccionada));
        document.add(new Paragraph("Orden: " + tipoOrdenSeleccionado));
        document.add(new Paragraph("" + frmOrdenar.Tiempo.getText()));
        document.add(new Paragraph("" + frmOrdenar.Pasos.getText()));
        document.add(elmnt: Chunk.NEWLINE);

        // Dato mínimo y máximo
        int min = datosOrdenados.stream().min(Integer::compare).orElse(other: 0);
        int max = datosOrdenados.stream().max(Integer::compare).orElse(other: 0);
        document.add(new Paragraph("Dato mínimo: " + min));
        document.add(new Paragraph("Dato máximo: " + max));
        document.add(elmnt: Chunk.NEWLINE);

        // Página 1: Datos y gráfica inicial (desordenados)
        document.add(new Paragraph(string: "Datos iniciales (Desordenados):"));
        PdfPTable tablaInicial = new PdfPTable(i: 2); // Dos columnas: X y Y
        tablaInicial.addCell(string: modelo.getTituloX()); // Título X
        tablaInicial.addCell(string: modelo.getTituloY()); // Título Y
        for (int i = 0; i < modelo.getCategoriasX().size(); i++) {
            tablaInicial.addCell(string: modelo.getCategoriasX().get(index: i)); // Valor de X
            tablaInicial.addCell(string: String.valueOf(obj:modelo.getValoresY().get(index: i))); // Valor de Y
        }
        document.add(elmnt: tablaInicial);
        document.add(elmnt: Chunk.NEWLINE);

        // Gráfica inicial
        document.add(new Paragraph(string: "Gráfica inicial:"));
        if (initialChart != null) {
            ByteArrayOutputStream chartImage = new ByteArrayOutputStream();
            ChartUtilities.writeChartAsPNG(out:chartImage, chart: initialChart, width: 300, height: 200);
            com.lowagie.text.Image image = com.lowagie.text.Image.getInstance(bytes: chartImage.toByteArray());
            document.add(elmnt: image);
        }

        // Salto de página
        document.newPage();

        // Página 2: Datos y gráfica ordenados
        document.add(new Paragraph(string: "Datos ordenados:"));
        PdfPTable tablaOrdenada = new PdfPTable(i: 2); // Dos columnas: X y Y
        tablaOrdenada.addCell(string: modelo.getTituloX()); // Título X
        tablaOrdenada.addCell(string: modelo.getTituloY()); // Título Y
        for (int i = 0; i < datosOrdenados.size(); i++) {
            tablaOrdenada.addCell(string: String.valueOf(i)); // Índice como X
            tablaOrdenada.addCell(string: String.valueOf(obj:datosOrdenados.get(index: i))); // Dato ordenado como Y
        }
        document.add(elmnt: tablaOrdenada);
        document.add(elmnt: Chunk.NEWLINE);

        // Gráfica final
        document.add(new Paragraph(string: "Gráfica final:"));
        if (finalChart != null) {
            ByteArrayOutputStream chartImage = new ByteArrayOutputStream();
            ChartUtilities.writeChartAsPNG(out:chartImage, chart: finalChart, width: 300, height: 200);
            com.lowagie.text.Image image = com.lowagie.text.Image.getInstance(bytes: chartImage.toByteArray());
```



Atributos

- **private FrmPrincipal vista**: Referencia a la vista del formulario principal (FrmPrincipal).
- **private DatosModelo modelo**: Referencia al modelo que maneja los datos cargados desde el archivo (DatosModelo).
- **private BitacoraModelo bitacora**: Referencia al modelo de bitácora que registra eventos importantes (BitacoraModelo).
- **private File archivoSeleccionado**: Archivo seleccionado por el usuario con los datos a ordenar.
- **private String algoritmoSeleccionado**: Nombre del algoritmo de ordenamiento elegido (Bubble, Insert, Merge, Quick, Select, Shell).
- **private String velocidadSeleccionada**: Velocidad de ejecución del ordenamiento (Alta, Media, Baja).
- **private String tipoOrdenSeleccionado**: Tipo de ordenamiento (Ascendente o Descendente).
- **private FrmOrdenar frmOrdenar**: Referencia a la vista donde se visualiza el proceso de ordenamiento (FrmOrdenar).
- **private long tiempoInicio**: Marca de tiempo cuando inicia el ordenamiento.
- **private int pasos**: Contador de pasos realizados durante el proceso de ordenamiento.
- **private JFreeChart initialChart**: Gráfica inicial generada antes del ordenamiento.
- **private JFreeChart finalChart**: Gráfica final después de completar el ordenamiento.

Métodos

- Contructor: Inicializa la vista (FrmPrincipal), el modelo de datos (DatosModelo) y la bitácora (BitacoraModelo).Agrega ActionListener a los botones Buscar, Aceptar y Ordenar en la vista principal.
  Configura el panel para mostrar la gráfica.

private void seleccionarArchivo()

- **Descripción**: Permite al usuario seleccionar un archivo con extensión .i1pec y muestra su ruta en la interfaz.
- **Acciones**:
  1. Muestra un JFileChooser para seleccionar un archivo.
  2. Si el usuario elige un archivo, se guarda en archivoSeleccionado y se muestra la ruta en la vista.

private void generarGrafica()

- **Descripción**: Carga los datos desde el archivo y genera una gráfica inicial antes del ordenamiento.
- **Acciones**:
  1. Verifica si el usuario ha seleccionado un archivo.
  2. Carga los datos en el modelo (modelo.cargarDesdeArchivo(archivoSeleccionado)).
  3. Crea una gráfica de barras con JFreeChart usando los datos cargados.
  4. Muestra la gráfica en vista.Panel.

private void abrirFrmOpciones()

- **Descripción**: Abre la ventana FrmOpciones, donde el usuario selecciona el algoritmo, la velocidad y el tipo de ordenamiento.
- **Acciones**:
  1. Muestra FrmOpciones.
  2. Espera la selección del usuario y almacena las opciones elegidas.
  3. Cierra FrmOpciones y abre FrmOrdenar para ejecutar el ordenamiento.

private void obtenerOpcionesOrdenamiento(FrmOpciones opciones)

- **Descripción**: Obtiene las opciones seleccionadas en FrmOpciones.
- **Acciones**:
    1. Determina el algoritmo de ordenamiento seleccionado.
    2. Obtiene el tipo de orden (Ascendente/Descendente).
    3. Guarda la velocidad seleccionada.

private void abrirFrmOrdenar()

- **Descripción**: Inicia el proceso de ordenamiento y muestra los detalles en FrmOrdenar.
- **Acciones**:
    1. Muestra FrmOrdenar con la información del algoritmo, velocidad y orden seleccionado.
    2. Inicia un Thread para ejecutar el algoritmo seleccionado.
    3. Registra el inicio del proceso en BitacoraModelo.

Métodos de Ordenamiento

- **Bubble Sort (Ordenamiento de Burbuja)**

Concepto: Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Se repite hasta que la lista esté ordenada.

Funcionamiento Paso a Paso:

1. Se recorre el arreglo de izquierda a derecha.
2. Se comparan los elementos adyacentes y se intercambian si es necesario.
3. Al finalizar un recorrido, el elemento más grande (o más pequeño) queda en su posición correcta.
4. Se repite el proceso hasta que no haya más intercambios.

- **Insertion Sort (Ordenamiento por Inserción)**

Concepto: Toma un elemento y lo inserta en su posición correcta en la parte ordenada de la lista.

Funcionamiento Paso a Paso:

1. Se inicia con el segundo elemento, tratándolo como la primera "carta" a insertar.
2. Se compara con los elementos anteriores y se mueve hasta encontrar su posición correcta.
3. Se repite con cada nuevo elemento hasta que la lista está ordenada.

- **Merge Sort (Ordenamiento por Mezcla)**

Concepto: Divide el arreglo en mitades hasta que cada sublista tenga un solo elemento y luego las mezcla en orden.

Funcionamiento Paso a Paso:

1. Divide el arreglo en dos mitades recursivamente hasta llegar a elementos individuales.
2. Fusiona las mitades en orden ascendente o descendente.
3. Sigue fusionando hasta reconstruir el arreglo completo.

- **Quick Sort (Ordenamiento Rápido)**

Concepto: Usa un **pivote** para dividir el arreglo en dos partes: una con elementos menores y otra con elementos mayores. Luego, ordena cada parte recursivamente.

Funcionamiento Paso a Paso:

1. Se elige un pivote (generalmente el último elemento).
2. Se reorganizan los elementos, colocando los menores a la izquierda y los mayores a la derecha.
3. Se aplica el mismo proceso recursivamente a cada sublista.

- **Selection Sort (Ordenamiento por Selección)**

Concepto:

Selecciona el elemento más pequeño (o más grande) y lo coloca en su posición correcta, repitiendo el proceso hasta que el arreglo esté ordenado.

Funcionamiento Paso a Paso:

1. Se busca el **mínimo/máximo** en la parte no ordenada del arreglo.
2. Se intercambia con el primer elemento de la parte no ordenada.
3. Se repite con el siguiente elemento hasta que todo el arreglo esté ordenado.

- **Shell Sort (Ordenamiento de Shell)**

Concepto:

Es una mejora del **Insertion Sort** que ordena primero elementos distantes y luego reduce la distancia entre ellos hasta que el arreglo está ordenado.

Funcionamiento Paso a Paso:

1. Se define un **gap** (salto), generalmente **n/2**.
2. Se aplica **Insertion Sort** a los elementos separados por el gap.
3. Se reduce el gap y se repite el proceso hasta que **gap = 1**.

Reporte

private void generarPDF

Este código está diseñado para generar un archivo PDF que documenta un proceso de ordenamiento de datos. Aquí está el flujo de trabajo detallado:

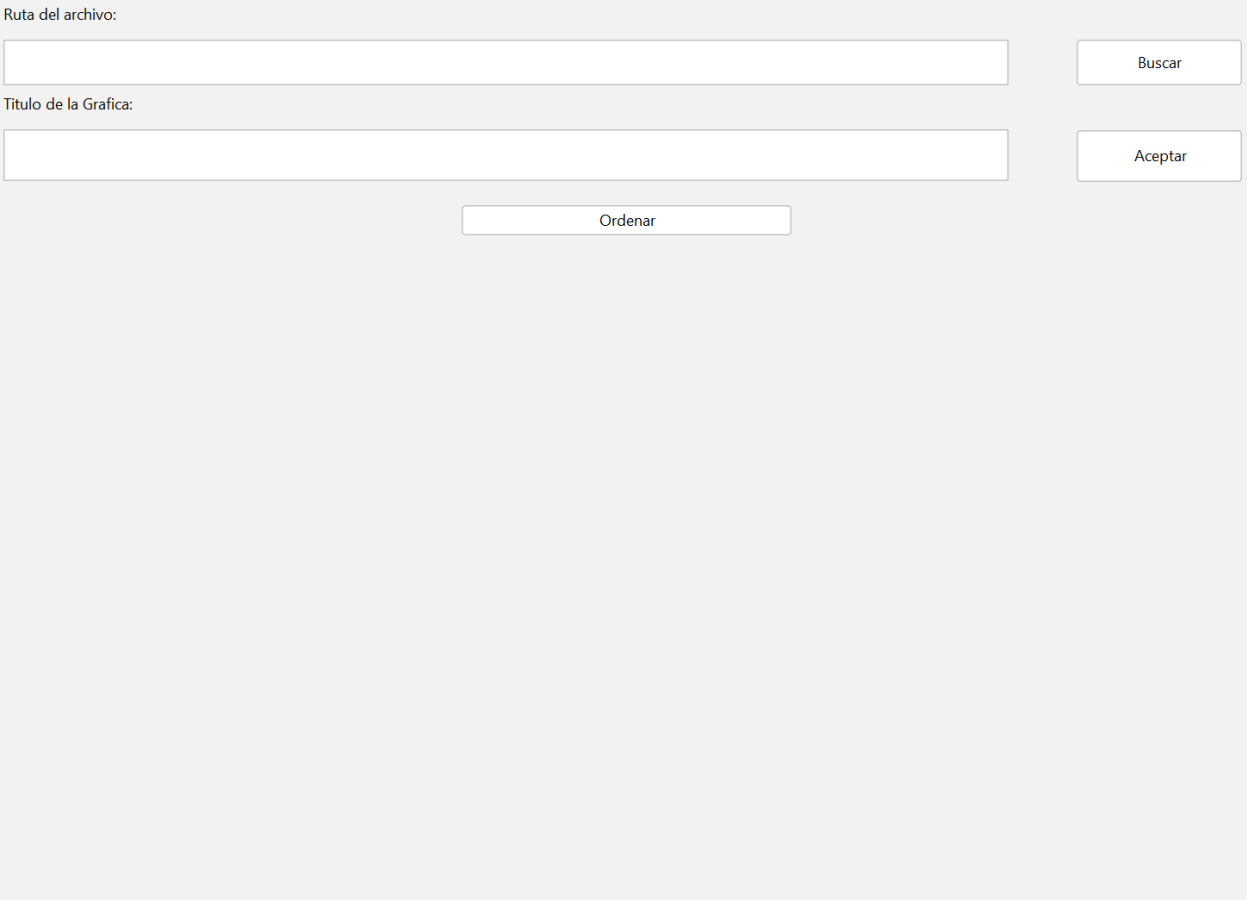
1. **Creación del documento**: Primero, se crea un objeto Document, que representa el archivo PDF a generar.
2. **Configuración del escritor de PDF**: Luego, se configura un escritor de PDF (PdfWriter) que asocia el documento con un archivo de salida llamado "Reporte_Ordenamiento.pdf". Se abre el documento para comenzar a agregar contenido.
3. **Información del estudiante y del ordenamiento**:

- Se agregan varias líneas de texto al documento con la información del estudiante (nombre y número de identificación).
- Después, se agrega información sobre el algoritmo de ordenamiento utilizado, la velocidad de ejecución, el tipo de orden (ascendente o descendente), y algunos valores de la interfaz de usuario (como el tiempo de ejecución y el número de pasos del proceso de ordenamiento).

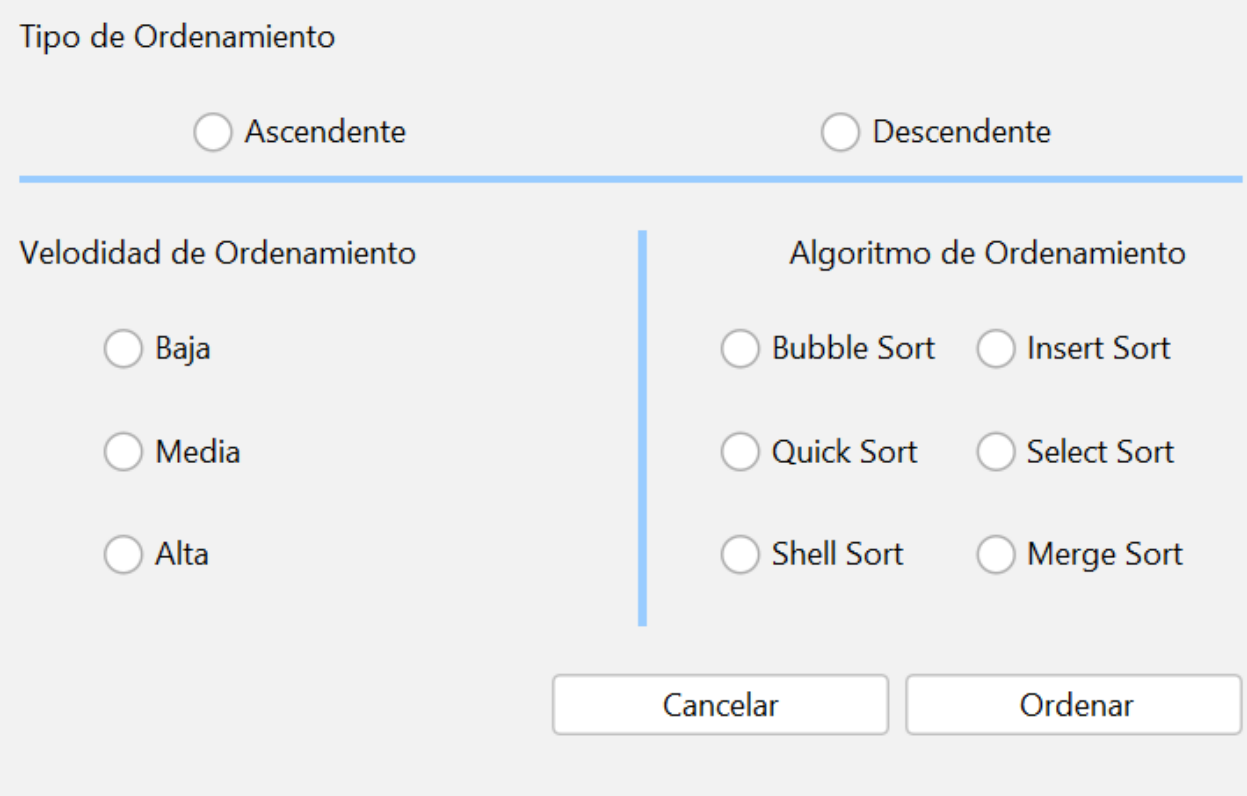
4. **Cálculo de los valores mínimo y máximo**: El código obtiene el valor mínimo y máximo de la lista de datos ordenados usando stream de Java. Estos valores se agregan al documento como parte del resumen.
5. **Página 1: Datos iniciales y gráfica**:
   - Se agregan los datos iniciales (desordenados) en una tabla, donde las columnas representan categorías X e Y.
   - Se genera una gráfica de los datos desordenados (si existe), y se agrega al PDF como una imagen en formato PNG.
6. **Salto de página**: Luego, se hace un salto de página para empezar la segunda sección del informe.
7. **Página 2: Datos ordenados y gráfica**:
   - Se agrega una tabla con los datos ordenados. La tabla muestra el índice de los datos como las coordenadas X y el valor de los datos como las coordenadas Y.
   - Se genera una gráfica de los datos ordenados (si existe), y también se agrega al PDF como una imagen.
8. **Manejo de excepciones**: Si ocurre algún error durante la ejecución, se captura y se imprime la traza del error.
9. **Cierre del documento**: Finalmente, se cierra el documento, asegurando que todo el contenido se guarde correctamente.

Vista

FrmPrincipal



FrmOpciones



FrmOrdenar

Diagrama de Flujo