FACULTAD INGENIERIA SOFTWARE-SISTEMAS

TALLER DE PROGRAMACION:

PROYECTO FINAL

- Yucra Vilcachagua Sol Sheyla ¨ 3 ciclo ¨

Sede del Norte

2025

DESCRIPCIÓN:

Como entendido el presente informe se basa en dataset que contiene información detallada sobre el registro y la recolección de residuos sólidos realizada por la Sub-Gerencia de Residuos Sólidos de la Municipalidad Provincial de Chiclayo, correspondiente a los periodos 2023 hasta octubre de 2025.
A lo largo de estos años se ha venido reuniendo información sobre cómo se realizan ciertas labores de limpieza en la ciudad, lo que permite notar cambios en el ritmo de trabajo y en la atención de diferentes zonas.

este conjunto de datos permite analizar la eficiencia del servicio, identificar patrones de generación de residuos en distintos sectores de la ciudad y evaluar el comportamiento temporal de la recolección.

Objetivos son:

-Registrar de manera ordenada y continua la información sobre las actividades de recolección de residuos en la ciudad.

- Monitorear el desempeño operativo de las rutas, frecuencias y volúmenes recolectados.

- Detectar variaciones o problemas en la prestación del servicio para tomar decisiones oportunas.

- Optimizar la planificación de la recolección mediante datos actualizados y verificables.

- Generar reportes y estadísticas que apoyen la gestión municipal y la mejora del servicio.

- Contribuir a una gestión más eficiente y sostenible del manejo de residuos sólidos.

Con formato: Referencia intensa, Fuente: (Predeterminada) Times New Roman, 12 pto, Sin Negrita, Sin Cursiva

Nombre del modulo	*Funcionalidades del modulo*
validarCredenciales	Verifica usuario y contraseña
contarLineas	Cuenta registros
Cargar datos	Carga y valida registro por registro
generarReporteAnual	Totales anuales de compras
GenerarReportePorCategoria	Total de productos por categoría
GenerarReportPorMetodoPago	Total por método de pago
GenerarReportePorEstado	Conteo por estado logístico
GenerarReportePorCiudad	Monto por ciudad
GenerarReportePorProvedor	Conteo por proveedor
GenerarReportePorAñoFiscal	Detalle completo filtrado por año
DetalleFinancieros	Su función es **guardar y proporcionar la información financiera de una orden**,
EntidadesRelacionadas	Su función es **almacenar y proporcionar información sobre las entidades vinculadas a una orden**, como el proveedor, el repartidor y el año fiscal.
InformacionLogistica	La clase **InformacionLogistica** representa los datos relacionados con la logística de una orden: dónde se entrega, cuánto demora y en qué estado se encuentra.
OrdenBase	La clase **OrdenBase** representa la información principal o básica de una orden. Es una clase fundamental que almacena datos esenciales como el identificador, el número y la fecha de la orden.
ProductoVendido	representa **un registro individual de un producto que ha sido vendido**. Su objetivo es **almacenar y transportar información** sobre las ventas dentro del sistema.
RegistroMaestro	La clase **RegistroMaestro** actúa como un "contenedor principal" o **registro unificado** que agrupa toda la información relacionada con una orden.
UsuarioIngresado	La clase **Usuario** representa a un usuario del sistema. Su función principal es **almacenar la información necesaria para identificar y**

	autenticar a una persona que utiliza la aplicación.
ContraseñaIngresada	lector.nextLine();

Opciones del menú principal y lo que realizan cada uno:

Reporte anual (Monto total y productos)	Genera un reporte del monto total anual y una lista de productos vendidos durante todo el año.
Reporte productos comprados por categoría	Agrupa los productos por categoría y muestra cuántos se compraron en cada una.
Reporte monto total por método de pago	Suma el total vendido según el método de pago utilizado (efectivo, tarjeta, etc.).
Reporte conteo de órdenes por estado	Muestra cuántas órdenes hay en cada estado (Entregado, Enviado, Procesando, etc.).
Reporte monto total por ciudad de entrega	Suma el total vendido en función de la ciudad donde se entregaron las órdenes.
Reporte conteo de órdenes por proveedor	Muestra cuántas órdenes corresponden a cada proveedor.
Reporte anual por año fiscal	Genera un reporte específico filtrado por el año fiscal ingresado por el usuario.
S alir del sistema	Finaliza la ejecución del sistema y cierra sesión.

Paquete: Modelo - Esquema Estadístico Ordenado

Atributo / Método	Descripción
idOrden	Identificador único de la orden.
numeroOrden	Número o código interno de la orden.
getNumeroOrden()	Retorna el número de orden.
getIdOrden()	Retorna el ID de la orden.
fechaOrden	Fecha en la que fue generada la orden.
OrdenBase(int idOrden, String numeroOrden, String fechaOrden)	Constructor que inicializa los datos básicos de la orden.
getFechaOrden()	Retorna la fecha de creación de la orden.

Esquema Estadístico de Clases y Atributos

Clase: DetalleFinanciero

Atributo / Método	Descripción
monto	Monto total de la orden sin impuestos.
impuesto	Porcentaje o valor del impuesto aplicado.
metodoPago	Método utilizado para pagar la orden.
getMonto()	Devuelve el monto base.
getImpuesto()	Devuelve el impuesto aplicado.
getMetodoPago()	Devuelve el método de pago.

Clase: ProductoVendido

Atributo / Método	Descripción
nombreProducto	Nombre del producto vendido.
categoria	Categoría a la que pertenece el producto.
cantidad	Cantidad vendida del producto.
precioUnitario	Precio por unidad del producto.
getNombreProducto()	Retorna el nombre del producto.
getCategoria()	Retorna la categoría del producto.

getCantidad()	Retorna la cantidad vendida.
getPrecioUnitario()	Retorna el precio unitario.

Clase: InformacionLogistica

Atributo / Método	Descripción
ciudadEntrega	Ciudad donde se realizará la entrega.
tiempoEntrega	Tiempo estimado de entrega en días.
estadoOrden	Estado actual de la orden.
getCiudadEntrega()	Devuelve la ciudad de entrega.
getTiempoEntrega()	Devuelve el tiempo de entrega.
getEstadoOrden()	Devuelve el estado de la orden.

Clase: EntidadesRelacionadas

Atributo / Método	Descripción
nombreProveedor	Nombre del proveedor responsable.
nombreRepartidor	Nombre del repartidor asignado.
añoFiscal	Año fiscal asociado a la orden.
getNombreProveedor()	Devuelve el nombre del proveedor.
getNombreRepartidor()	Devuelve el nombre del repartidor.
getAñoFiscal()	Devuelve el año fiscal.

Clase: RegistroMaestro

Atributo / Método	Descripción
base	Datos principales de la orden.
financiero	Información financiera.
producto	Producto vendido asociado a la orden.
logistica	Detalles logísticos.
entidades	Información de proveedor y año fiscal.
RegistroMaestro(...)	Une todos los datos en un solo registro completo.
getBase()	Devuelve la información base.
getFinanciero()	Devuelve los detalles financieros.
getProducto()	Devuelve la información del producto.
getLogistica()	Devuelve la información logística.

getEntidades()	Devuelve las entidades relacionadas.

Clase: Usuario

Atributo / Método	Descripción
nombreUsuario	Nombre registrado del usuario.
contraseña	Contraseña para autenticación.
Usuario(String nombreUsuario, String contraseña, int nivelAcceso)	Inicializa un usuario del sistema.
getNombreUsuario()	Retorna el nombre de usuario.
getContraseña()	Retorna la contraseña.

Paquete: Vista

Atributo / Método	Descripción
NOMBRE_ARCHIVO_DATOS	Ruta del archivo de datos.
NOMBRE_ARCHIVO_USUARIO	Ruta del archivo de usuarios.

Metodos:

main()	***Inicia el sistema, pide credenciales y muestra el menú principal.***
mostrarMenuReportes()	***Muestra las opciones de reportes y ejecuta cada una.***

El Controlador:

Es el intermediario que conecta la Vista con el Modelo. Su trabajo es recibir las acciones que el usuario realiza en la Vista (como presionar un botón), tomar esos datos, enviarlos al Modelo para que sean procesados y finalmente actualizar la Vista con el resultado. El Controlador coordina todo el flujo entre lo que el usuario hace y lo que el programa necesita responder.

ControladorDatos.java × Ejecucion.java ×

Source History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Controlador;

import Modelo.ModeloDatos;
import Vista.Ejecucion;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 *
 * @author USER
 */
public class ControladorDatos {
    private ModeloDatos modelo;
    private Ejecucion vista;
    private final String NOMBRE_ARCHIVO = "Data/Datos.txt";

    public ControladorDatos(ModeloDatos modelo, Ejecucion vista) {
        this.modelo = modelo;
        this.vista = vista;

        System.out.println("Cargando datos iniciales de " + NOMBRE_ARCHIVO + "...");
        cargarDatos(NOMBRE_ARCHIVO);

        if (modelo.getNumColumnas() != 12 || modelo.getNumFilas() == 0) {
            System.out.println("\n[AVISO DE LECTURA] El archivo " + NOMBRE_ARCHIVO +
                               " tiene " + modelo.getNumColumnas() + "Debe tener 12 columnas");
            System.out.println("Asegurese de que el archivo existe en esa ruta y que el separador de columna es la coma (,)");
        }
    }


    public void cargarDatos(String nombreArchivo) {
        int numFilasPotenciales = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(nombreArchivo))) {
            String linea;
            br.readLine();
            while ((linea = br.readLine()) != null) {
                if (linea.trim().length() > 0) { numFilasPotenciales++; }
            }
        } catch (IOException e) {
            System.out.println("\n--- ERROR DE LECTURA ---");
            System.out.println("No se pudo leer el archivo: " + nombreArchivo);
            return;
        }
```

ControladorDatos.java × Ejecucion.java ×

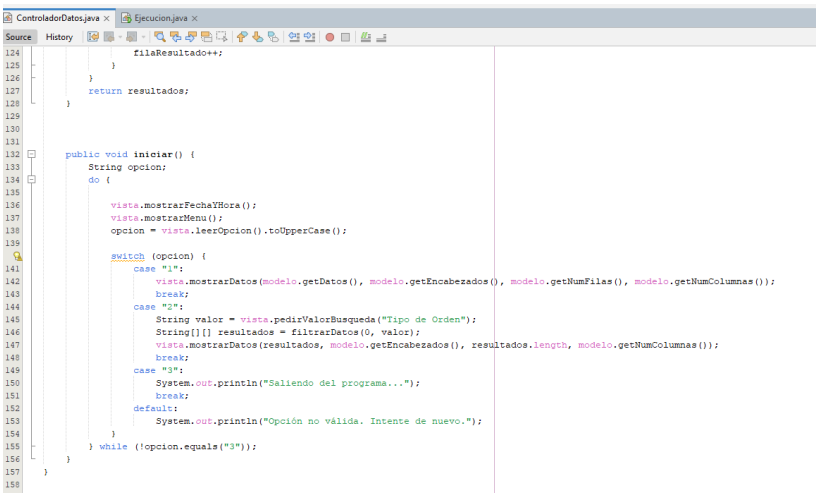
Source History

```
        }

        if (numFilasPotenciales == 0) return;

        String[][] datosCorregidos = new String[numFilasPotenciales][modelo.getNumColumnas()];
        int filaActual = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(nombreArchivo))) {
            br.readLine();
            String linea;
            while ((linea = br.readLine()) != null) {
                if (linea.trim().length() == 0) continue;

                String[] campos = linea.split(modelo.getDELIMITADOR(), -1);
                String[] camposFinales = new String[modelo.getNumColumnas()];

                if (campos.length > 0) {

                    if (campos.length > modelo.getNumColumnas()) {

                        for (int i = 0; i < modelo.getINDICE_DESCRIPCION(); i++) {
                            camposFinales[i] = campos[i];
                        }
                        int inicioCamposFinales = campos.length - modelo.getCAMPOS_FINALES();
                        StringBuilder sb = new StringBuilder();
                        for (int i = modelo.getINDICE_DESCRIPCION(); i < inicioCamposFinales; i++) {
                            sb.append(campos[i]);
                            if (i < inicioCamposFinales - 1) { sb.append(","); }
                        }
                        camposFinales[modelo.getINDICE_DESCRIPCION()] = sb.toString();
                        camposFinales[modelo.getINDICE_DESCRIPCION() + 1] = campos[inicioCamposFinales];
                        camposFinales[modelo.getINDICE_DESCRIPCION() + 2] = campos[inicioCamposFinales + 1];

                    } else {

                        for(int i = 0; i < modelo.getNumColumnas(); i++) {
                            camposFinales[i] = (i < campos.length) ? campos[i] : "";
                        }
                    }
                    datosCorregidos[filaActual] = camposFinales;
                    filaActual++;
                }
            }
            modelo.setDatos(datosCorregidos);
            modelo.setNumFilas(filaActual);

        } catch (IOException e) {
            System.out.println("--- ERROR AL CARGAR DATOS EN EL ARREGLO ---");
        }
    }
```

ControladorDatos.java × Ejecucion.java ×

Source History

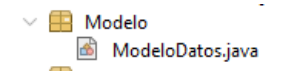
```
    public String[][] filtrarDatos(int indiceColumna, String valor) {
        String[][] datosActuales = modelo.getDatos();
        int numFilas = modelo.getNumFilas();
        int numColumnas = modelo.getNumColumnas();

        if (datosActuales == null || numFilas == 0) {
            return new String[0][numColumnas];
        }

        String valorNormalizado = valor.trim();
        int filasResultado = 0;
        for (int i = 0; i < numFilas; i++) {
            if (datosActuales[i][indiceColumna] != null && datosActuales[i][indiceColumna].trim().equalsIgnoreCase(valorNormalizado)) {
                filasResultado++;
            }
        }

        String[][] resultados = new String[filasResultado][numColumnas];
        int filaResultado = 0;

        for (int i = 0; i < numFilas; i++) {
            if (datosActuales[i][indiceColumna] != null && datosActuales[i][indiceColumna].trim().equalsIgnoreCase(valorNormalizado)) {
                System.arraycopy(datosActuales[i], 0, resultados[filaResultado], 0, numColumnas);
                filaResultado++;
            }
        }
        return resultados;
    }


    public void iniciar() {
        String opcion;
        do {

            vista.mostrarFechaYHora();
            vista.mostrarMenu();
            opcion = vista.leerOpcion().toUpperCase();

            switch (opcion) {
                case "1":
                    vista.mostrarDatos(modelo.getDatos(), modelo.getEncabezados(), modelo.getNumFilas(), modelo.getNumColumnas());
                    break;
                case "2":
                    String valor = vista.pedirValorBusqueda("Tipo de Orden");
                    String[][] resultados = filtrarDatos(0, valor);
                    vista.mostrarDatos(resultados, modelo.getEncabezados(), resultados.length, modelo.getNumColumnas());
                    break;
                case "3":
```




```
                filaResultado++;
            }
        }
        return resultados;
    }



    public void iniciar() {
        String opcion;
        do {

            vista.mostrarFechaYHora();
            vista.mostrarMenu();
            opcion = vista.leerOpcion().toUpperCase();

            switch (opcion) {
                case "1":
                    vista.mostrarDatos(modelo.getDatos(), modelo.getEncabezados(), modelo.getNumFilas(), modelo.getNumColumnas());
                    break;
                case "2":
                    String valor = vista.pedirValorBusqueda("Tipo de Orden");
                    String[][] resultados = filtrarDatos(0, valor);
                    vista.mostrarDatos(resultados, modelo.getEncabezados(), resultados.length, modelo.getNumColumnas());
                    break;
                case "3":
                    System.out.println("Saliendo del programa...");
                    break;
                default:
                    System.out.println("Opción no válida. Intente de nuevo.");
            }
        } while (!opcion.equals("3"));
    }
}

```

El Modelo:

es la parte del programa encargada de gestionar los datos y la lógica interna. Aquí se crean las clases que representan objetos del mundo real y sus comportamientos, como clientes, productos, registros o cualquier entidad que maneje información. El Modelo no sabe nada de ventanas ni botones; solo se encarga de almacenar datos y ofrecer métodos para manipularlos.



ControladorDatos.java | Ejecucion.java | ModeloDatos.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Modelo;

/**
 *
 * @author USER
 */
public class ModeloDatos {



    private final String DELIMITADOR = ",";
    private final String[] ENCABEZADOS = {
        "Tipo de orden", "Año de la orden", "RUC de la entidad",
        "Numero de orden de pedido", "Numero de la orden", "Numero de SIAF",
        "Fecha de la orden", "Monto de la orden", "Nombre del proveedor",
        "Descripcion de la orden", "Tipo de periodo", "Tipo de Periodista"
    };
    private final int INDICE_DESCRIPCION = 9;
    private final int CAMPOS_FINALES = 2;

    private String[][] datos;
    private int numFilas = 0;
    private final int numColumnas = ENCABEZADOS.length;


    public ModeloDatos() {
        this.datos = null;
        this.numFilas = 0;
    }


    public String[][] getDatos() { return datos; }

    public void setDatos(String[][] datos) {
        this.datos = datos;
        this.numFilas = (datos != null) ? datos.length : 0;
    }

    public String[] getEncabezados() { return ENCABEZADOS; }
    public int getNumFilas() { return numFilas; }
    public void setNumFilas(int numFilas) { this.numFilas = numFilas; }
    public int getNumColumnas() { return numColumnas; }
    public String getDELIMITADOR() { return DELIMITADOR; }
    public int getINDICE_DESCRIPCION() { return INDICE_DESCRIPCION; }
    public int getCAMPOS_FINALES() { return CAMPOS_FINALES; }
```


ControladorDatos.java
Ejecucion.java
ModeloDatos.java
Source
History
"Descripcion de la orden", "Tipo de periodo", "Tipo de Periodista"
};
private final int INDICE_DESCRIPCION = 9;
private final int CAMPOS_FINALES = 2;
private String[][] datos;
private int numFilas = 0;
private final int numColumnas = ENCABEZADOS.length;
public ModeloDatos() {
this.datos = null;
this.numFilas = 0;
}
public String[][] getDatos() { return datos; }
public void setDatos(String[][] datos) {
this.datos = datos;
this.numFilas = (datos != null) ? datos.length : 0;
}
public String[] getEncabezados() { return ENCABEZADOS; }
public int getNumFilas() { return numFilas; }
public void setNumFilas(int numFilas) { this.numFilas = numFilas; }
public int getNumColumnas() { return numColumnas; }
public String getDELIMITADOR() { return DELIMITADOR; }
public int getINDICE_DESCRIPCION() { return INDICE_DESCRIPCION; }
public int getCAMPOS_FINALES() { return CAMPOS_FINALES; }
@Override
public String toString() {
if (datos == null || numFilas == 0) {
return "ModeloDatos [Estado: VACIO] - No hay datos cargados.";
}
return "ModeloDatos [Filas: " + numFilas + ", Columnas: " + numColumnas + "]";
}
}

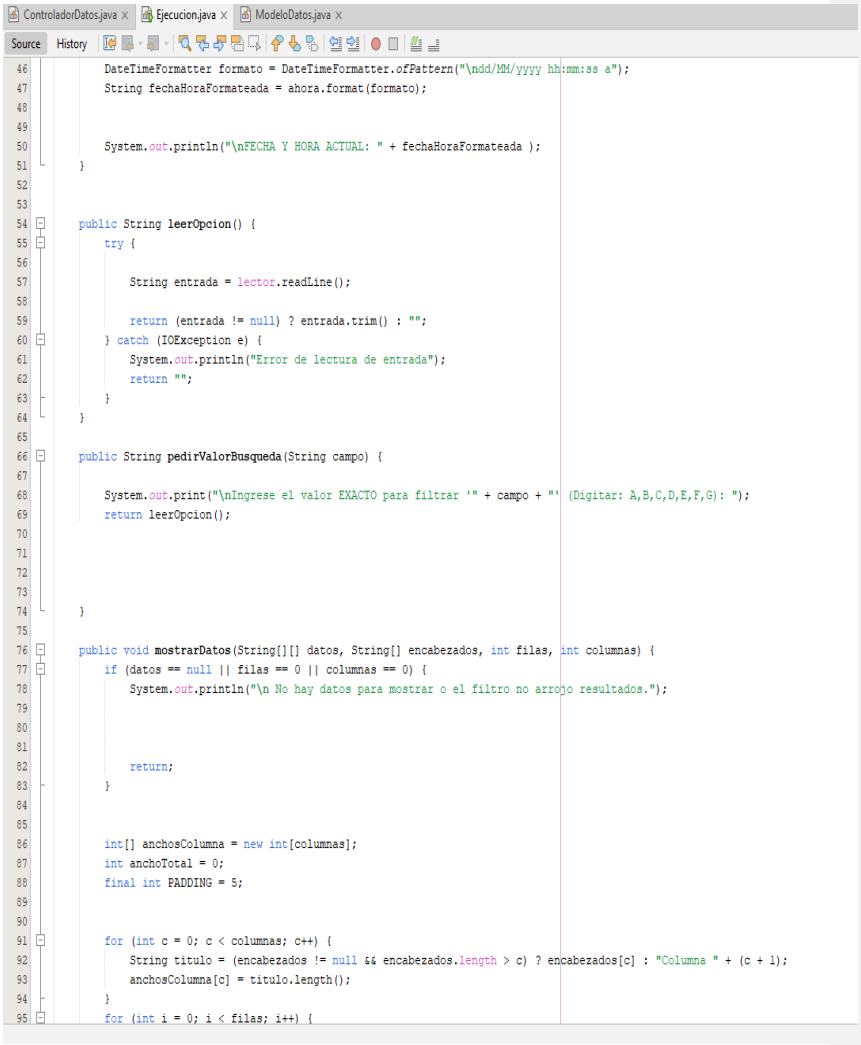
La Vista:

Es la interfaz gráfica con la que interactúa el usuario. Incluye las ventanas, botones, campos de texto, tablas y cualquier elemento visual. Su función principal es mostrar información y permitir que el usuario realice acciones. La Vista no procesa la lógica compleja, solo recibe o muestra datos.

ControladorDatos.java | Ejecucion.java | ModeloDatos.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Vista;

import Controlador.ControladorDatos;
import Modelo.ModeloDatos;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 *
 * @author USER
 */
public class Ejecucion {




    private final BufferedReader lector;

    public Ejecucion() {
        this.lector = new BufferedReader(new InputStreamReader(System.in));  }


    public void mostrarMenu() {
        System.out.println("\n--- MENÚ DE OPCIONES ---");

        System.out.println(" 1. Mostrar todos los Datos (Tabla completa)");
        System.out.println(" 2. Filtrar por tipo de orden");
        System.out.println(" 3. Salir");
        System.out.print("------------------------\nSeleccione una opcion: ");
    }


    public void mostrarFechaYHora() {


        LocalDateTime ahora = LocalDateTime.now();

        DateTimeFormatter formato = DateTimeFormatter.ofPattern("\ndd/MM/yyyy hh:mm:ss a");
        String fechaHoraFormateada = ahora.format(formato);


        System.out.println("\nFECHA Y HORA ACTUAL: " + fechaHoraFormateada );
```




```
        DateTimeFormatter formato = DateTimeFormatter.ofPattern("\ndd/MM/yyyy hh:mm:ss a");
        String fechaHoraFormateada = ahora.format(formato);


        System.out.println("\nFECHA Y HORA ACTUAL: " + fechaHoraFormateada );
    }


    public String leerOpcion() {
        try {

            String entrada = lector.readLine();

            return (entrada != null) ? entrada.trim() : "";
        } catch (IOException e) {
            System.out.println("Error de lectura de entrada");
            return "";
        }
    }

    public String pedirValorBusqueda(String campo) {

        System.out.print("\nIngrese el valor EXACTO para filtrar '" + campo + "' (Digitar: A,B,C,D,E,F,G): ");
        return leerOpcion();




    }

    public void mostrarDatos(String[][] datos, String[] encabezados, int filas, int columnas) {
        if (datos == null || filas == 0 || columnas == 0) {
            System.out.println("\n No hay datos para mostrar o el filtro no arrojo resultados.");



            return;
        }


        int[] anchosColumna = new int[columnas];
        int anchoTotal = 0;
        final int PADDING = 5;


        for (int c = 0; c < columnas; c++) {
            String titulo = (encabezados != null && encabezados.length > c) ? encabezados[c] : "Columna " + (c + 1);
            anchosColumna[c] = titulo.length();
        }
        for (int i = 0; i < filas; i++) {
```

ControladorDatos.java × Ejecucion.java × ModeloDatos.java ×

Source History

```
        for (int c = 0; c < columnas; c++) {
            String titulo = (encabezados != null && encabezados.length > c) ? encabezados[c] : "Columna " + (c + 1);
            anchosColumna[c] = titulo.length();
        }
        for (int i = 0; i < filas; i++) {
            for (int j = 0; j < columnas; j++) {
                if (datos[i][j] != null) {

                }


            }


        }
        for(int ancho : anchosColumna) { anchoTotal += ancho + PADDING + 1; }
        anchoTotal += 1;


        System.out.println("\n" + "=".repeat(anchoTotal));
        System.out.println("Filas mostradas: " + filas + " | Columnas: " + columnas);


        System.out.print("|");
        for (int c = 0; c < columnas; c++) {
            String titulo = (encabezados != null && encabezados.length > c) ? encabezados[c] : "Columna " + (c + 1);
            int anchoCelda = anchosColumna[c] + PADDING;
            System.out.print(String.format(" %-" + anchoCelda + "s", titulo) + "|");
        }
        System.out.println("\n" + "=".repeat(anchoTotal));


        for (int i = 0; i < filas; i++) {
            System.out.print("|");
            for (int j = 0; j < columnas; j++) {
                String celda = (datos[i][j] != null) ? datos[i][j] : "";
                int anchoCelda = anchosColumna[j] + PADDING;
                System.out.print(String.format(" %-" + anchoCelda + "s", celda) + "|");
            }
            System.out.println();
            if ((i + 1) % 10 == 0 && i < filas - 1) {
                System.out.println("-" + "-".repeat(anchoTotal - 1));
            }
        }
        System.out.println("=" + "=".repeat(anchoTotal - 1));
    }


    public static void main(String[] args) {
```

ControladorDatos.java × Ejecucion.java × ModeloDatos.java ×

Source History

```
        for (int c = 0; c < columnas; c++) {
            String titulo = (encabezados != null && encabezados.length > c) ? encabezados[c] : "Columna " + (c + 1);
            int anchoCelda = anchosColumna[c] + PADDING;
            System.out.print(String.format(" %-" + anchoCelda + "s", titulo) + "|");
        }
        System.out.println("\n" + "=".repeat(anchoTotal));


        for (int i = 0; i < filas; i++) {
            System.out.print("|");
            for (int j = 0; j < columnas; j++) {
                String celda = (datos[i][j] != null) ? datos[i][j] : "";
                int anchoCelda = anchosColumna[j] + PADDING;
                System.out.print(String.format(" %-" + anchoCelda + "s", celda) + "|");
            }
            System.out.println();
            if ((i + 1) % 10 == 0 && i < filas - 1) {
                System.out.println("-" + "-".repeat(anchoTotal - 1));
            }
        }
        System.out.println("=" + "=".repeat(anchoTotal - 1));
    }


    public static void main(String[] args) {

        ModeloDatos modelo = new ModeloDatos();


        Ejecucion vista = new Ejecucion();


        ControladorDatos controlador = new ControladorDatos(modelo, vista);


        controlador.iniciar();
    }
}
```