Sistema de Análisis de Consumo Eléctrico

Integrantes:

-Serrano Arias Alexander

-Ramos Hilario Adriel

Problemática

- Muchas organizaciones almacenan datos de consumo eléctrico en archivos planos, pero no tienen una herramienta que los analice fácilmente. Por eso, desarrollamos un sistema que procesa esa información y genera reportes claros y exportables de forma rápida.

Objetivos del Proyecto

- **Objetivo general:**
  Diseñar un sistema que autentique usuarios, procese datos de consumo eléctrico y genere reportes estadísticos claros, permitiendo su visualización o exportación.
- **Objetivos específicos:**
- Implementar autenticación de usuarios desde un archivo de texto.
- Leer y procesar datos de consumo eléctrico desde un archivo CSV.
- Generar y exportar reportes con información estadística relevante.

ModeloConsumoKWh

- **Clase de modelo: Representa el promedio de consumo eléctrico (kWh) por cada distrito.**
- **Atributos principales:**
- **distrito: guarda el nombre del distrito.**
- **promedioConsumo: almacena el promedio de consumo en kWh.**
- **Constructor: Inicializa los valores de distrito y promedioConsumo cuando se crea un objeto.**
- **toString() con @Override:**
- **Sobrescribe el método toString() de la clase Object.**
- **Devuelve una cadena con formato, mostrando el distrito y el consumo con 2 decimales.**
- **Uso en el proyecto: Sirve para mostrar reportes claros de consumo por distrito en tablas o archivos.**

```
public class ModeloConsumoKWh {
      private String distrito;
    private double promedioConsumo;

    public ModeloConsumoKWh(String distrito, double promedioConsumo) {
        this.distrito = distrito;
        this.promedioConsumo = promedioConsumo;
    }

    @Override
    public String toString() {
        return String.format("%-30s | %10.2f", distrito, promedioConsumo);
    }
}
```

ControladorConsumoKWh

- Objetivos :

*Calcular el consumo promedio (kWh) por distrito a partir del archivo de datos CSV.

*Generar y mostrar un reporte en pantalla o exportarlo a un archivo de texto (consumoKWh.txt).

```
public class ControladorConsumoKWh {

      private final String archivoSalida = "consumoKWh.txt";
    private final String archivoLog = "src/auditoria.log";

    public void procesarArchivo(String rutaArchivo, boolean exportar) {
        String resultado = String.format("%-30s | %s\n", "Distrito", "Promedio (kWh)");
        resultado += "-----------------------------------------------\n";

        try (BufferedReader br1 = new BufferedReader(new FileReader(rutaArchivo))) {
            br1.readLine(); // saltar cabecera
            String linea;
            String distritosProcesados = "";

            while ((linea = br1.readLine()) != null) {
                String[] partes = linea.split(";");
                if (partes.length > 22) {
                    String distrito = partes[12].trim();

                    // Evita procesar el mismo distrito más de una vez
                    if (!distritosProcesados.contains("[" + distrito + "]")) {
                        distritosProcesados += "[" + distrito + "]";

                        double suma = 0;
                        int contador = 0;

                        try (BufferedReader br2 = new BufferedReader(new FileReader(rutaArchivo))) {
                            br2.readLine(); // saltar cabecera
                            String 12;
```

```
                        resultado += modelo.toString() + "\n";
                    }
                }
            }

            if (exportar) {
                try (FileWriter fw = new FileWriter(archivoSalida)) {
                    fw.write(resultado);
                }
                System.out.println("Reporte exportado a " + archivoSalida);
            } else {
                System.out.println(resultado);
            }

        } catch (Exception e) {
            registrarError("ControladorConsumoKWh", e.getMessage());
            System.out.println("Ocurrió un error al procesar el archivo.");
        }
    }

    private void registrarError(String clase, String mensaje) {
        try (FileWriter fw = new FileWriter(archivoLog, true)) {
            fw.write(new ModeloErrores(clase, mensaje).toString() + "\n");
        } catch (IOException ex) {
            System.err.println("Error al registrar auditoría: " + ex.getMessage());
        }
    }
```

MenuVista

Objetivos de MenuVista:

*Iniciar la ejecución del programa llamando al menú principal.

*Gestionar la interacción con el usuario a través de la clase VistaMenu.

Puntos Clave del Códigomain:

*(String[] args): Es el punto de entrada del programa en Java.

*VistaMenu menu = new VistaMenu(): Crea un objeto de la clase VistaMenu.

*menu.iniciar(): Llama al método que gestiona el inicio de sesión y muestra el menú principal.

```
 */
public class MenuVista {

      public static void main(String[] args) {

        VistaMenu menu = new VistaMenu();
        menu.iniciar();
    }
}
```

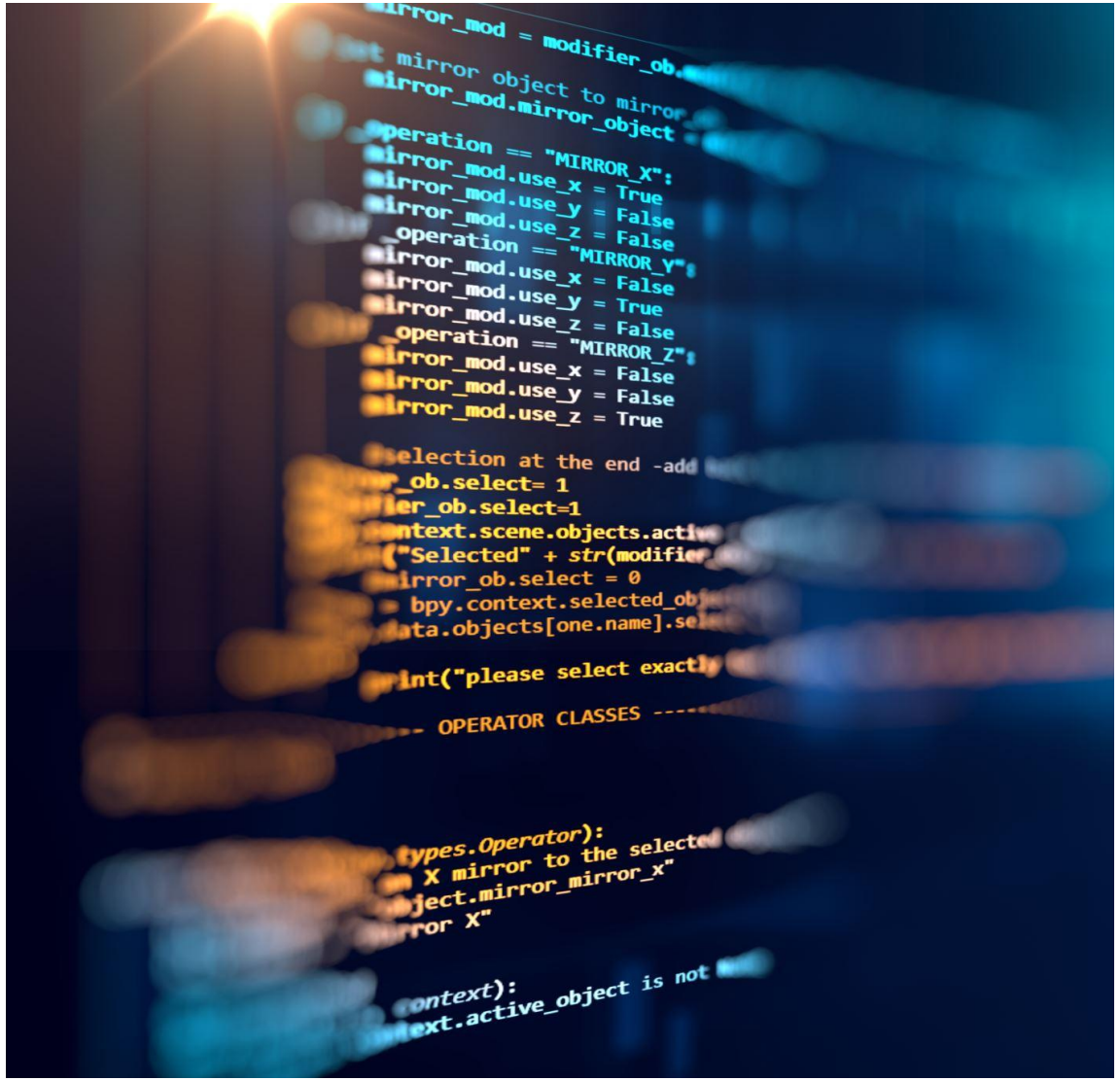
Uso de Estructuras en el Proyecto

```
do {
    System.out.println(menuPrincipal);
    System.out.print("Seleccione una opción: ");

    try {
        opcion = Integer.parseInt(lector.nextLine());
        switch (opcion) {
            case 1, 2, 3, 4 -> mostrarSubmenu(opcion);
            case 5 -> System.out.println("Saliendo del programa.");
            default -> System.out.println("Opción no válida.");
        }
    } catch (NumberFormatException e) {
        System.out.println("Por favor, ingrese un número válido.");
    }

} while (opcion != 5);
```

Estructura	¿Cuándo se usa en general?
if	Para tomar decisiones según una condición.
else	Para ejecutar algo cuando el if no se cumple.
while	Para repetir mientras una condición sea verdadera.
do-while	Para repetir al menos una vez y seguir si la condición es verdadera.
for	Para repetir un bloque un número conocido de veces (recorrer arreglos).
switch	Para elegir entre varias opciones según un valor.
arreglos	Para almacenar varios datos del mismo tipo.
try-catch	Para manejar errores sin detener el programa.
try-with-resources	Para manejar recursos (archivos) y cerrarlos automáticamente.

Conclusiones

- Se aplicó el modelo MVC, lo que permitió organizar el código y separar correctamente las funciones del sistema.
- Se gestionaron correctamente los archivos CSV y TXT, logrando trabajar con datos reales y registrar errores.
- El sistema ofrece una interfaz por consola sencilla, permite visualizar análisis y exportar resultados de forma práctica.



Bibliografia

- Video sobre lectura de archivos CSV en Java – https://www.youtube.com/watch?v=_GpN3GQFgP4
- Curso Java desde cero (Píldoras Informáticas) – https://www.youtube.com/playlist?list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B
- Scanner en Java – https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html