

Universidad Rafael Landívar
Facultada de ingeniería
Área de informática y sistemas
Manejo e implementación de archivos
Catedrático: Ing. David Fernando Luna Hernández

Documentación “Segundo Proyecto”

Diego Jeancarlo Cosillo Ramos 1136222
José Daniel Alvarado Zapata 1047222

Guatemala, 07 de noviembre de 2023

MANUAL TECNICO DEL PROYECTO

REQUERIMIENTOS DE SOFTWARE Y HARDWARE.

- **Requerimientos de software.**

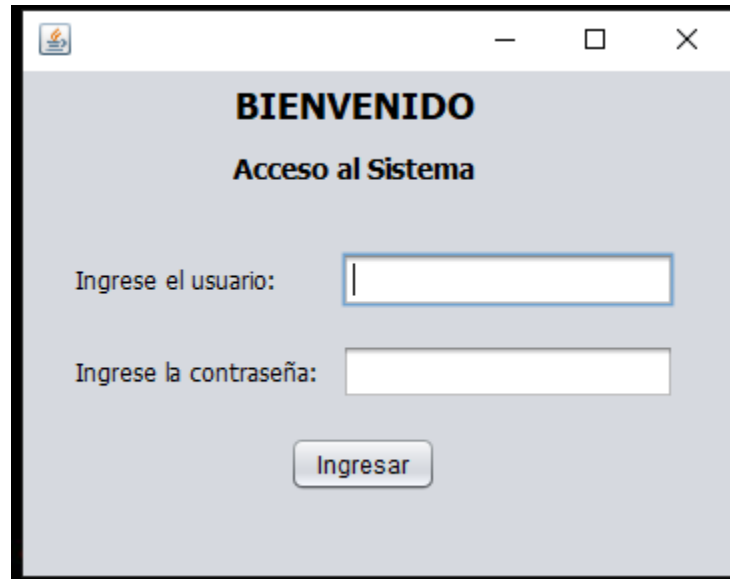
1. Tener en la computadora el software Apache NetBeans instalado
2. Mínimo de 781 MB de espacio libre en el almacenamiento de disco duro.
3. Mínimo de 512 MB disponibles en RAM.
4. Tener alguno de los siguientes sistemas operativos: Windows, Linux o macOS.
5. Tener instalado el Java Development kit (JDK).

- **Requerimientos de Hardware.**

1. Un procesador que tenga como mínimo 1GHz de velocidad, generalmente funciona con un procesador Intel Pentium III en adelante.
2. Tener Instalada una RAM mínima de 4 GB.
3. Cualquier Tarjeta gráfica, ya que NetBeans no requiere una grafica especial para su ejecución. Una tarjeta de gráficos integrados es suficiente.
4. Tener un Disco duro instalado en la computadora para la descarga de NetBeans.

INDICACIONES PARA EJECUTAR EL PROGRAMA

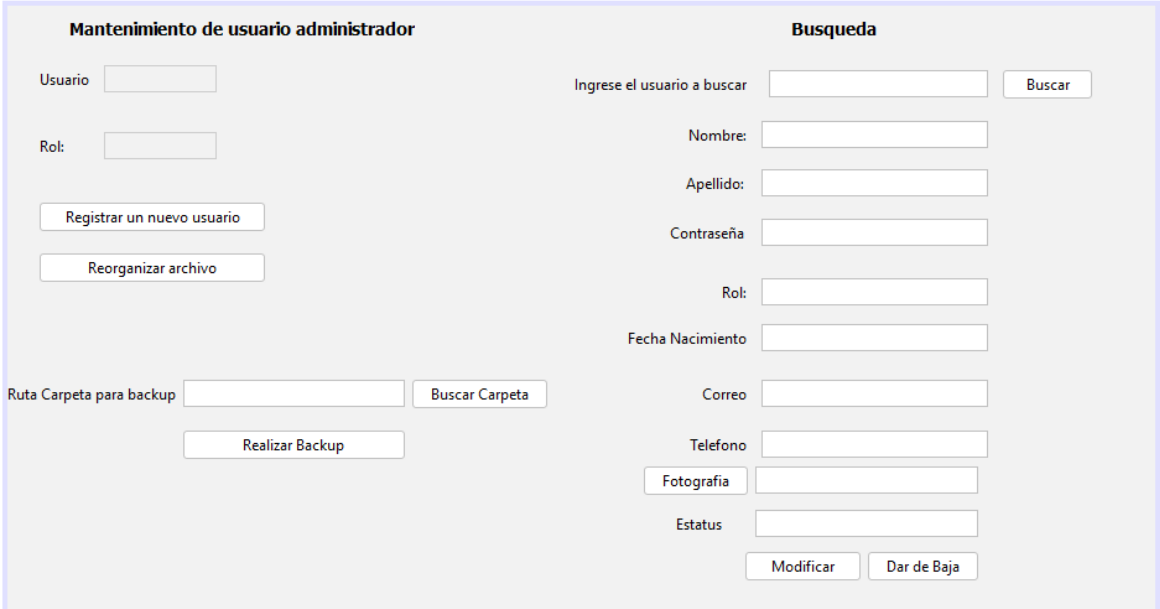
1. Form Acceso:



Formulario de acceso al sistema. El formulario tiene un fondo gris claro y un título "BIENVENIDO" en negrita. Debajo del título, el subtítulo "Acceso al Sistema" también está en negrita. Hay dos campos de entrada de texto: "Ingrese el usuario:" y "Ingrese la contraseña:". Debajo de los campos, hay un botón "Ingresar".

El usuario debe de ingresar un usuario existente, en dado caso no existe, da la opción a crear un nuevo usuario.

2. Form MenuAdmin:



Formulario de mantenimiento de usuario administrador. El formulario está dividido en dos secciones: "Mantenimiento de usuario administrador" y "Busqueda".

Mantenimiento de usuario administrador:

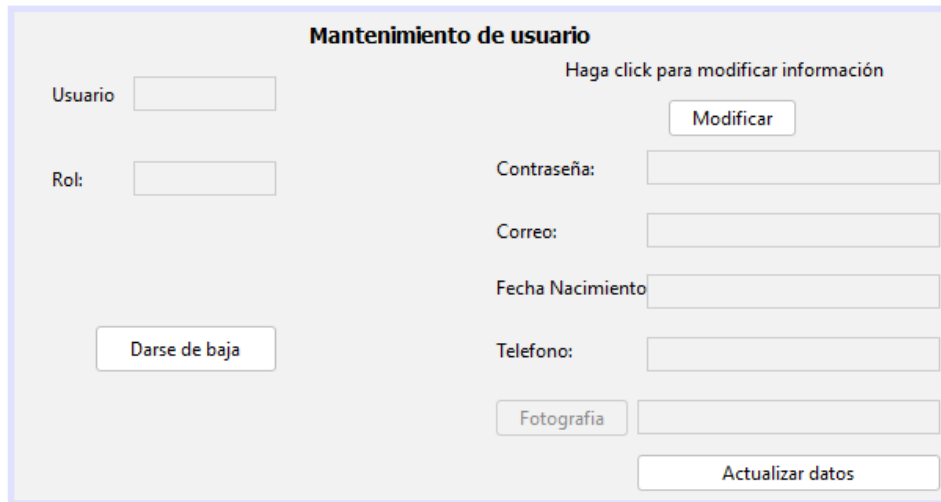
- Usuario:
- Rol:
- Registrar un nuevo usuario:
- Reorganizar archivo:
- Ruta Carpeta para backup: Buscar Carpeta:
- Realizar Backup:

Busqueda:

- Ingrese el usuario a buscar: Buscar:
- Nombre:
- Apellido:
- Contraseña:
- Rol:
- Fecha Nacimiento:
- Correo:
- Telefono:
- Fotografia:
- Estatus:
- Modificar: Dar de Baja:

Si el usuario entra con permisos de administrados, se le da la opción de poder crear, modificar, buscar y dar de baja usuario.

3. Form MenuUsuario:



Mantenimiento de usuario

Haga click para modificar información

Usuario

Rol:

Contraseña:

Correo:

Fecha Nacimiento:

Telefono:

Fotografia

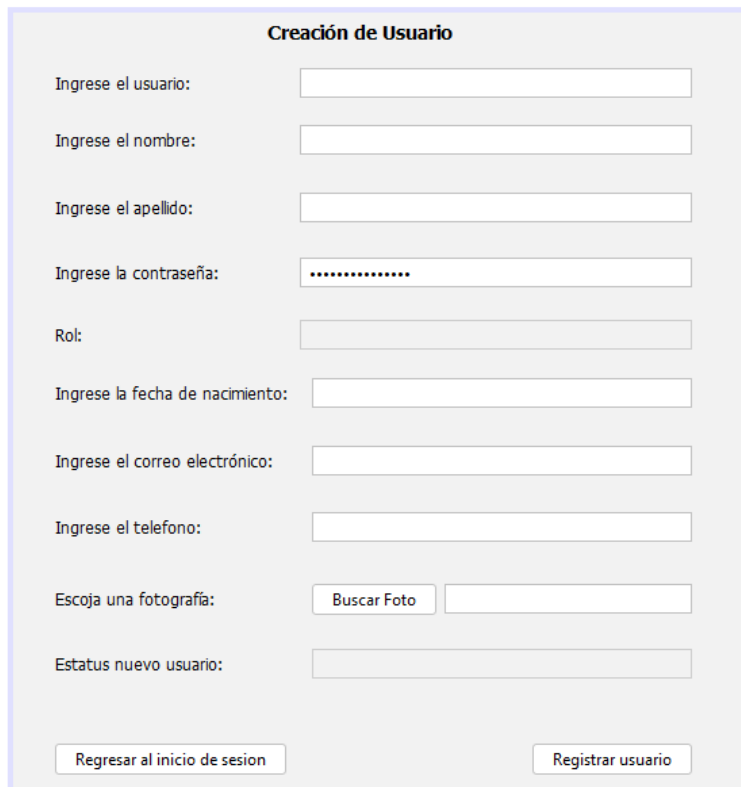
Darse de baja

Modificar

Actualizar datos

Este es un menú mas limitado en el cual el usuario no es administrados, el usuario podrá modificar únicamente su propia información y darse de baja del programa.

4. Form NuevoUsuario:



Creación de Usuario

Ingrese el usuario:

Ingrese el nombre:

Ingrese el apellido:

Ingrese la contraseña:

Rol:

Ingrese la fecha de nacimiento:

Ingrese el correo electrónico:

Ingrese el telefono:

Escoja una fotografía:

Estatus nuevo usuario:

Regresar al inicio de sesion

Registrar usuario

En este menú es donde se da la opción para crear un usuario, donde el rol se asigna automáticamente dependiendo si existen ya usuarios en el programa o no.

FUNCIONALIDADES DEL SISTEMA

- **Acceso al sistema**

- ✓ Permite que el primer usuario sea administrador.
- ✓ Si ya existe un usuario permite ingresar al usuario como rol únicamente de usuario.
- ✓ Muestra el nivel de seguridad de la contraseña.
- ✓ Genera el archivo donde se muestra la comparación de la contraseña.
- ✓ No permite ingresar contraseña con nivel bajo de seguridad.
- ✓ Permite buscar fotografía.
- ✓ Por defecto los usuarios ingresan como vigente.
- ✓ Si no se ingresan todos los datos en el archivo no deja ingresar.
- ✓ Al ingresar se muestra el usuario y el rol, más no la fotografía.

- **Menú de administración**

- ✓ Usuario que no sea administrador solo se puede modificar el mismo y darse de baja el mismo.
- ✓ Si el usuario es administrador tiene el poder de hacer lo mismo que un usuario no administrador, más también puede ingresar nuevos usuarios, buscar usuarios existentes y modificar estos mismos, y también puede dar de baja a otros usuarios, más no a el mismo.

- **Reorganización del archivo usuario**

- ✓ El administrador puede reorganizar un archivo.
- ✓ Se condensa cada bloque cuando hay algún usuario que se da de baja.

- **Creación de Archivos**

- ✓ Archivo "Usuario".
- ✓ Archivo "desc_usuario".
- ✓ Bloques para cada determinado numero de usuario
- ✓ Descriptor de cada bloque

- **Aspectos generales del programa**

- ✓ Todos los archivos se guardan en el directorio C:/MEIA.
- ✓ Todos los archivos cuentan con descriptor.
- ✓ Todos los archivos cuentan con el nombre y estructura indicada.
- ✓ Los registros se encuentran delimitados por salto de línea.
- ✓ Se cuenta con las validaciones indicadas.
- ✓ El separador en los archivos es |
- ✓ Cada que se llega al numero indicado de usuarios se crea un nuevo bloque

- ✓ En el archivo usuario se va ordenando y muestra cual es el siguiente al que apunta.
- ✓ Se Ordena por usuario.

ESTRUCTURA DEL PROGRAMA

Métodos

- Método validar:

```
private boolean validar(String usuario, String contra){
    NuevoUsuario nuevoUsuario = new NuevoUsuario();
    MenuUsuario menuUsuario = new MenuUsuario();
    MenuAdmin menuAdmin = new MenuAdmin();
    try{
        String filePath = "C:/MEIA/Usuario.txt";
        File file = new File(pathname: filePath);
        Scanner scanner = new Scanner(source: file);
        if(file.exists()){
            if(file.length() == 0){
                JOptionPane.showMessageDialog(parentComponent:null, message:"No existen usuarios, se creará el primer usuario");
                //crearUsuario(file);
                nuevoUsuario.setVisible(b: true);
            } else{
                String userGuardado = "";
                while(scanner.hasNextLine()){
                    String line = scanner.nextLine();
                    String[] parts = line.split(regex: "\\|");
                    userGuardado = parts[0].trim();
                    String contraGuardada = parts[3].trim();
                    int rolGuardado = Integer.parseInt(parts[4]);
                    if(usuario.equals(anObject: userGuardado) && contra.equals(anObject: contraGuardada)){
                        String rolU = "";
                        if(rolGuardado==0){
                            rolU = "Usuario";
                            menuUsuario.mostrarDatos(usuario);
                            menuUsuario.setVisible(b: true);
                        } else{
                            rolU = "Administrador";
                            menuAdmin.setVisible(b: true);
                        }
                    }
                    Icon icono = new ImageIcon(location: getClass().getResource(name: "6590944.png"));
                    //JOptionPane.showMessageDialog(rootPane, "Usuario: "+usuario+" Rol: "+rolU, "Datos de Ingreso", JOptionPane.PLAIN_MESSAGE, icono);
                    return true;
                }
            }
            if(!usuario.equals(anObject: userGuardado)){
                JOptionPane.showMessageDialog(parentComponent:null, message:"El usuario no existe, se creará un nuevo usuario");
                nuevoUsuario.setVisible(b: true);
            }
            scanner.close();
        }
    } catch(IOException e){
        e.printStackTrace();
    }
}
```

Entradas:

String usuario: Es una cadena que representa el nombre de usuario que se desea validar.

String contra: Es una cadena que representa la contraseña que se desea validar.

Procesos:

El método crea instancias de las siguientes clases: NuevoUsuario, MenuUsuario, y MenuAdmin.

Luego, intenta abrir un archivo ubicado en "C:/MEIA/Usuario.txt" y crea un objeto File y un objeto Scanner para leer el contenido del archivo.

Verifica si el archivo existe y si su longitud es igual a cero (lo que significa que no hay usuarios en el archivo).

Si el archivo no existe o está vacío, muestra un mensaje de advertencia ("No existen usuarios, se creará el primer usuario") y llama a la función `nuevousuario.setVisible(true)`; para mostrar la ventana de creación de un nuevo usuario.

Si el archivo contiene usuarios, el método recorre cada línea del archivo, divide la línea en partes utilizando el carácter '|' como separador y guarda diferentes valores en variables.

Compara el nombre de usuario y la contraseña proporcionados como entrada con los valores leídos del archivo.

Si encuentra una coincidencia, determina el rol del usuario (0 para "Usuario" o cualquier otro valor para "Administrador"), muestra un mensaje y devuelve `true`.

Si no encuentra un usuario con el nombre proporcionado, muestra un mensaje ("El usuario no existe, se creará un nuevo usuario") y llama a `nuevousuario.setVisible(true)`; para mostrar la ventana de creación de un nuevo usuario.

Finalmente, se cierra el Scanner y el método devuelve `false`.

Salidas:

`true`: Si el nombre de usuario y la contraseña coinciden con los valores almacenados en el archivo y se encuentra un usuario correspondiente, el método devuelve `true`.

`false`: Si no se encuentra un usuario con el nombre proporcionado o si el archivo está vacío, el método devuelve `false`.

- Método Mostrar Datos:

```
public void mostrarDatos(String usuario){
    try{
        String filePath = "C:/MEIA/Usuario.txt";
        File file = new File(pathname: filePath);
        Scanner scanner = new Scanner(source: file);
        if(file.exists()){
            if(file.length() == 0){
                JOptionPane.showMessageDialog(parentComponent:null, message:"No existen usuarios");
            }
            else{
                String userGuardado = "";
                while(scanner.hasNextLine()){
                    String line = scanner.nextLine();
                    String[] parts = line.split(regex: "\\|");
                    userGuardado = parts[0].trim();
                    String contraGuardada = parts[3].trim();
                    int rolGuardado = Integer.parseInt(parts[4]);
                    if(usuario.equals(userGuardado)){
                        String rolU = "";
                        if(rolGuardado==0){
                            rolU = "Usuario";
                        }
                        else{
                            rolU = "Administrador";
                        }
                        txtUsuario.setText(s: usuario);
                        txtRol.setText(s: rolU);
                        Icon icono = new ImageIcon(location: getClass().getResource(name: "6590944.png"));
                        //JOptionPane.showMessageDialog(rootPane, "Usuario: "+usuario+" Rol: "+rolU, "Datos de Ingreso", JOptionPane.PLAIN_MESSAGE, icono);
                    }
                }
            }
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

Entradas:

String usuario: Es una cadena que representa el nombre de usuario para el cual se desean mostrar los datos.

Procesos:

El método intenta abrir un archivo ubicado en "C:/MEIA/Usuario.txt" y crea un objeto File y un objeto Scanner para leer el contenido del archivo.

Verifica si el archivo existe y si su longitud es igual a cero (lo que significa que no hay usuarios en el archivo).

Si el archivo no existe o está vacío, muestra un mensaje de advertencia ("No existen usuarios") mediante JOptionPane.showMessageDialog.

Si el archivo contiene usuarios, el método recorre cada línea del archivo, divide la línea en partes utilizando el carácter '|' como separador y guarda diferentes valores en variables.

Compara el nombre de usuario proporcionado con los nombres de usuario leídos del archivo.

Si encuentra una coincidencia, determina el rol del usuario (0 para "Usuario" o cualquier otro valor para "Administrador").

Actualiza dos campos de texto en la interfaz de usuario (txtUsuario y txtRol) con el nombre de usuario y el rol del usuario correspondiente.

No devuelve ningún valor (el método es de tipo void).

Salidas:

El método no devuelve ningún valor. En su lugar, actualiza los campos de texto en la interfaz de usuario con el nombre de usuario y el rol del usuario correspondiente.

- **Método Leer puntuación:**

```
public void LeerPuntuacion(int largo, int mayusculas, int letrasIngresadas, int numerosIngresados, int simbolos, File scann) {

    String rutaArchivo = "C:/MEIA/puntuacion.txt";
    int puntuacion = 0;

    try{
        FileReader fileReader = new FileReader(fileName: rutaArchivo);
        BufferedReader bufferedReader = new BufferedReader(in: fileReader);

        String linea;
        int numeroDeLinea = 1;

        while ((linea = bufferedReader.readLine()) != null) {
            int numLinea = Integer.parseInt(·: linea);
            if(numeroDeLinea == 1){
                if(largo < numLinea){
                    JOptionPane.showMessageDialog(parentComponent: null, "No puede ingresar una contraseña que contenga menos de "+numLinea+" caracteres, Ingrese nuevamente");
                    break;
                }else{
                    numeroDeLinea++;
                    continue;
                }
            }else if(numeroDeLinea == 2){
                puntuacion = numLinea * largo;
                numeroDeLinea++;
                continue;
            }else if(numeroDeLinea == 3){
                puntuacion = puntuacion + (numLinea * mayusculas);
                numeroDeLinea++;
                continue;
            }else if(numeroDeLinea == 4){
                puntuacion = puntuacion + (letrasIngresadas * numLinea);
                numeroDeLinea++;
                continue;
            }else if(numeroDeLinea == 5){
                puntuacion = puntuacion + (numerosIngresados * numLinea);
                numeroDeLinea++;
                continue;
            }else if(numeroDeLinea == 6){
                puntuacion = puntuacion + (simbolos * (numLinea*largo));
                numeroDeLinea++;
                continue;
            }else if(numeroDeLinea == 7){
                if(letrasIngresadas == largo){
                    puntuacion = puntuacion - numLinea;
                    numeroDeLinea++;
                }
            }
        }
    }
}
```

Entradas:

int largo: Un entero que representa la longitud de la contraseña.

int mayusculas: Un entero que representa la cantidad de letras mayúsculas en la contraseña.

int letrasIngresadas: Un entero que representa la cantidad total de letras ingresadas en la contraseña.

int numerosIngresados: Un entero que representa la cantidad total de números ingresados en la contraseña.

int simbolos: Un entero que representa la cantidad de símbolos en la contraseña.

File scann: Un objeto File que parece relacionarse con algún tipo de escaneo o proceso posterior.

Procesos:

El método utiliza una ruta de archivo predefinida ("C:/MEIA/puntuacion.txt") para abrir un archivo y leer su contenido.

Lee el archivo línea por línea y en cada línea realiza una serie de cálculos basados en el valor numérico de la línea.

Los cálculos se realizan en función del número de línea (de 1 a 8) y los valores de entrada proporcionados, como la longitud de la contraseña, las mayúsculas, las letras, los números y los símbolos.

La puntuación se acumula a medida que se leen las líneas del archivo de configuración.

Si se calcula una puntuación (es decir, puntuación no es igual a cero), se llama al método Resultado con el valor de puntuación calculado y el objeto File scann.

Salidas:

El método no devuelve ningún valor, pero si se calcula una puntuación (cuando puntuación no es igual a cero), llama al método Resultado con el valor de puntuación calculado y el objeto File scann como entrada.

En resumen, este método se utiliza para calcular una puntuación de contraseña en función de los criterios definidos en un archivo de configuración. Si se calcula una puntuación, se llama al método Resultado para procesar esa puntuación.

- Método Resultado:

```
public void Resultado(int puntuacion, File scann) {
    String rutaArchivo = "C:/MEIA/resultado.txt";

    try {
        FileReader fileReader = new FileReader(fileName: rutaArchivo);
        BufferedReader bufferedReader = new BufferedReader(in: fileReader);

        String linea;
        int numeroDeLinea = 1;

        while ((linea = bufferedReader.readLine()) != null) {
            String[] valores = linea.split(sep: ",");
            int inicio = Integer.parseInt(s: valores[0].trim());
            int fin = Integer.parseInt(s: valores[1].trim());
            if(numeroDeLinea == 1){
                if(puntuacion <= fin && puntuacion >= inicio){
                    JOptionPane.showMessageDialog(parentComponent:null, message:"La contraseña es insegura");
                    this.setVisible(b: false);
                }
            }
            numeroDeLinea++;
            continue;
        }else if(numeroDeLinea == 2){
            if(puntuacion <= fin && puntuacion >= inicio){
                JOptionPane.showMessageDialog(parentComponent:null, message:"La contraseña es poco segura");
                this.setVisible(b: false);
            }
        }
        numeroDeLinea++;
        continue;
        }else if(numeroDeLinea == 3){
            if(puntuacion <= fin && puntuacion >= inicio){
                JOptionPane.showMessageDialog(parentComponent:null, message:"La contraseña es segura");
            }
        }
        numeroDeLinea++;
        continue;
        }else if(numeroDeLinea == 4){
            if(puntuacion <= fin && puntuacion >= inicio){
                JOptionPane.showMessageDialog(parentComponent:null, message:"La contraseña es muy segura");
            }
        }
        numeroDeLinea++;
        continue;
    }
}
```

Entradas:

int puntuacion: Un entero que representa la puntuación de la contraseña.

File scann: Un objeto File que parece estar relacionado con algún proceso posterior.

Procesos:

El método utiliza una ruta de archivo predefinida ("C:/MEIA/resultado.txt") para abrir un archivo y leer su contenido.

Lee el archivo línea por línea y en cada línea realiza una serie de cálculos basados en los valores numéricos de inicio y fin en esa línea.

Compara la puntuación calculada con los valores de inicio y fin de cada línea para determinar en qué rango se encuentra la puntuación.

Muestra un mensaje emergente (JOptionPane.showMessageDialog) que indica la seguridad de la contraseña en función del rango en el que se encuentra la puntuación.

Si la contraseña se considera "segura" o "muy segura", no oculta la ventana actual. Sin embargo, si la contraseña se considera "insegura" o "poco segura", oculta la ventana actual (this.setVisible(false)).

Salidas:

El método no devuelve ningún valor, pero muestra un mensaje emergente que indica la seguridad de la contraseña en función del rango de puntuación y, en ciertos casos, oculta la ventana actual.

LIBRERIAS UTILIZADAS EN EL ACCESO

java.io.BufferedReader;

Función: Proporciona la capacidad de leer texto de un flujo de entrada de caracteres. Se utiliza comúnmente para leer líneas de un archivo de texto.

java.io.BufferedWriter;

Función: Proporciona la capacidad de escribir texto en un flujo de salida de caracteres. Se utiliza para escribir líneas de texto en un archivo de texto.

java.io.File;

Función: Representa un archivo o un directorio en el sistema de archivos. Se utiliza para interactuar con archivos y directorios en el sistema.

java.io.FileNotFoundException;

Función: Esta excepción se lanza cuando un intento de abrir un archivo no se puede realizar porque el archivo no existe en la ubicación especificada.

java.io.FileReader;

Función: Permite la lectura de caracteres de un archivo. Se utiliza junto con BufferedReader para leer archivos de texto.

java.io.FileWriter;

Función: Permite escribir caracteres en un archivo. Se utiliza junto con BufferedWriter para escribir en archivos de texto.

java.io.IOException;

Función: Representa una excepción que se lanza cuando ocurre un error durante operaciones de entrada/salida. Se utiliza para manejar errores de E/S.

`java.text.DateFormat;`

Función: Proporciona la capacidad de formatear y analizar fechas y horas en una representación legible por humanos. Se utiliza para convertir entre objetos Date y cadenas de fecha.

`java.text.ParseException;`

Función: Representa una excepción que se lanza cuando ocurre un error al analizar una cadena en un objeto Date. Se utiliza para manejar errores de análisis de fechas.

`java.text.SimpleDateFormat;`

Función: Permite formatear y analizar fechas y horas en un formato específico. Se utiliza para definir un formato personalizado para las fechas.

`java.util.Date;`

Función: Representa una fecha y hora en Java. Se utiliza para trabajar con fechas y horas en aplicaciones Java.

`java.util.Scanner;`

Función: Permite leer datos de entrada de diferentes tipos desde la consola o desde otros flujos de entrada. Se utiliza para la entrada de usuario y lectura de archivos.

`javax.swing.JOptionPane;`

Función: Proporciona la capacidad de mostrar cuadros de diálogo y ventanas emergentes en aplicaciones de interfaz de usuario de Java. Se utiliza para mostrar mensajes y recibir entrada del usuario.

`java.util.regex.Matcher;`

Función: Se utiliza para buscar coincidencias de patrones en cadenas de texto utilizando expresiones regulares. Permite realizar operaciones avanzadas de búsqueda y reemplazo en cadenas.

`java.util.regex.Pattern;`

Función: Se utiliza para compilar y trabajar con expresiones regulares. Permite definir patrones de búsqueda utilizando notación de expresiones regulares.

`javax.swing.Icon;`

Función: Representa un icono gráfico que se puede mostrar en componentes de la interfaz de usuario. Se utiliza para mostrar imágenes o íconos en ventanas y diálogos de la interfaz de usuario.

`javax.swing.ImageIcon;`

Función: Permite cargar imágenes desde archivos y crear objetos `Icon` a partir de esas imágenes. Se utiliza junto con `Icon` para mostrar imágenes en la interfaz de usuario

LIBRERIAS UTILIZADAS EN CREAR UN NUEVO USUARIO

`BufferedReader`:

Función: Proporciona la capacidad de leer texto de un flujo de entrada de caracteres. Se utiliza comúnmente para leer líneas de un archivo de texto.

`BufferedWriter`:

Función: Proporciona la capacidad de escribir texto en un flujo de salida de caracteres. Se utiliza para escribir líneas de texto en un archivo de texto.

`File`:

Función: Representa un archivo o un directorio en el sistema de archivos. Se utiliza para interactuar con archivos y directorios en el sistema.

`FileNotFoundException`:

Función: Esta excepción se lanza cuando un intento de abrir un archivo no se puede realizar porque el archivo no existe en la ubicación especificada.

`FileReader`:

Función: Permite la lectura de caracteres de un archivo. Se utiliza junto con `BufferedReader` para leer archivos de texto.

`FileWriter`:

Función: Permite escribir caracteres en un archivo. Se utiliza junto con `BufferedWriter` para escribir en archivos de texto.

`PrintWriter`:

Función: Proporciona capacidades extendidas de escritura de texto a un flujo de caracteres. Se utiliza para escribir en archivos y otros flujos de caracteres.

`Date`:

Función: Representa una fecha y hora en Java. Se utiliza para trabajar con fechas y horas en aplicaciones Java.

`Scanner`:

Función: Permite leer datos de entrada de diferentes tipos desde la consola o desde otros flujos de entrada. Se utiliza para la entrada de usuario y lectura de archivos.

Pattern:

Función: Se utiliza para compilar y trabajar con expresiones regulares. Permite definir patrones de búsqueda utilizando notación de expresiones regulares.

Matcher:

Función: Se utiliza para buscar coincidencias de patrones en cadenas de texto utilizando expresiones regulares. Permite realizar operaciones avanzadas de búsqueda y reemplazo en cadenas.

Icon:

Función: Representa un icono gráfico que se puede mostrar en componentes de la interfaz de usuario. Se utiliza para mostrar imágenes o íconos en ventanas y diálogos de la interfaz de usuario.

ImageIcon:

Función: Permite cargar imágenes desde archivos y crear objetos Icon a partir de esas imágenes. Se utiliza junto con Icon para mostrar imágenes en la interfaz de usuario.

JFileChooser:

Función: Proporciona una ventana de diálogo que permite a los usuarios seleccionar archivos o directorios. Se utiliza para permitir al usuario seleccionar archivos desde el sistema de archivos.

JOptionPane:

Función: Proporciona la capacidad de mostrar cuadros de diálogo y ventanas emergentes en aplicaciones de interfaz de usuario de Java. Se utiliza para mostrar mensajes y recibir entrada del usuario.

FileNameExtensionFilter:

Función: Se utiliza para filtrar los tipos de archivos que se muestran en un cuadro de diálogo de selección de archivos, permitiendo seleccionar solo archivos con extensiones específicas.

SimpleDateFormat:

Función: Permite formatear y analizar fechas y horas en un formato específico. Se utiliza para definir un formato personalizado para las fechas.

ArrayList:

Función: Implementa una lista dinámica en Java que puede contener elementos de cualquier tipo. Se utiliza para almacenar y manipular colecciones de elementos.

List:

Función: Define una interfaz que representa una colección ordenada de elementos. Se utiliza para trabajar con colecciones de datos genéricos en Java.

LIBRERIAS UTILIZADAS PARA MENUADMIN:

File:

Función: Representa un archivo o un directorio en el sistema de archivos. Se utiliza para interactuar con archivos y directorios en el sistema.

FileNotFoundException:

Función: Esta excepción se lanza cuando un intento de abrir un archivo no se puede realizar porque el archivo no existe en la ubicación especificada.

FileWriter:

Función: Permite escribir caracteres en un archivo. Se utiliza junto con PrintWriter para escribir en archivos de texto.

IOException:

Función: Representa una excepción que se lanza cuando ocurre un error durante operaciones de entrada/salida. Se utiliza para manejar errores de E/S.

PrintWriter:

Función: Proporciona capacidades extendidas de escritura de texto a un flujo de caracteres. Se utiliza para escribir en archivos y otros flujos de caracteres.

ArrayList:

Función: Implementa una lista dinámica en Java que puede contener elementos de cualquier tipo. Se utiliza para almacenar y manipular colecciones de elementos.

List:

Función: Define una interfaz que representa una colección ordenada de elementos. Se utiliza para trabajar con colecciones de datos genéricos en Java.

Scanner:

Función: Permite leer datos de entrada de diferentes tipos desde la consola o desde otros flujos de entrada. Se utiliza para la entrada de usuario y lectura de archivos.

JFileChooser:

Función: Proporciona una ventana de diálogo que permite a los usuarios seleccionar archivos o directorios. Se utiliza para permitir al usuario seleccionar archivos desde el sistema de archivos.

JOptionPane:

Función: Proporciona la capacidad de mostrar cuadros de diálogo y ventanas emergentes en aplicaciones de interfaz de usuario de Java. Se utiliza para mostrar mensajes y recibir entrada del usuario.

FileNameExtensionFilter:

Función: Se utiliza para filtrar los tipos de archivos que se muestran en un cuadro de diálogo de selección de archivos, permitiendo seleccionar solo archivos con extensiones específicas.

LIBRERIAS UTILIZADAS PARA MENU USUARIO:

BufferedImage:

Función: Representa una imagen en memoria y permite el procesamiento de imágenes. Se utiliza para cargar y manipular imágenes en aplicaciones Java.

File:

Función: Representa un archivo o un directorio en el sistema de archivos. Se utiliza para interactuar con archivos y directorios en el sistema.

FileWriter:

Función: Permite escribir caracteres en un archivo. Se utiliza junto con PrintWriter para escribir en archivos de texto.

IOException:

Función: Representa una excepción que se lanza cuando ocurre un error durante operaciones de entrada/salida. Se utiliza para manejar errores de E/S.

PrintWriter:

Función: Proporciona capacidades extendidas de escritura de texto a un flujo de caracteres. Se utiliza para escribir en archivos y otros flujos de caracteres.

ArrayList:

Función: Implementa una lista dinámica en Java que puede contener elementos de cualquier tipo. Se utiliza para almacenar y manipular colecciones de elementos.

List:

Función: Define una interfaz que representa una colección ordenada de elementos. Se utiliza para trabajar con colecciones de datos genéricos en Java.

Scanner:

Función: Permite leer datos de entrada de diferentes tipos desde la consola o desde otros flujos de entrada. Se utiliza para la entrada de usuario y lectura de archivos.

ImageIO:

Función: Proporciona funcionalidad para leer y escribir imágenes en diferentes formatos. Se utiliza para cargar y guardar imágenes en aplicaciones Java.

Icon:

Función: Representa un icono gráfico que se puede mostrar en componentes de la interfaz de usuario. Se utiliza para mostrar imágenes o íconos en ventanas y diálogos de la interfaz de usuario.

ImageIcon:

Función: Permite cargar imágenes desde archivos y crear objetos Icon a partir de esas imágenes. Se utiliza junto con Icon para mostrar imágenes en la interfaz de usuario.

JFileChooser:

Función: Proporciona una ventana de diálogo que permite a los usuarios seleccionar archivos o directorios. Se utiliza para permitir al usuario seleccionar archivos desde el sistema de archivos.

JOptionPane:

Función: Proporciona la capacidad de mostrar cuadros de diálogo y ventanas emergentes en aplicaciones de interfaz de usuario de Java. Se utiliza para mostrar mensajes y recibir entrada del usuario.

FileNameExtensionFilter:

Función: Se utiliza para filtrar los tipos de archivos que se muestran en un cuadro de diálogo de selección de archivos, permitiendo seleccionar solo archivos con extensiones específicas.