PROGRAMACIÓN DE SERVICIOS Y PROCESOS – TAREA 6

📦 Tarea: Validación de Datos y Políticas de Seguridad

📖 Descripción del Problema:

La tarea de la unidad está dividida en dos actividades:

Actividad 6.1

- Crea una aplicación en Java que realice los siguientes pasos:
  - Solicita el nombre del usuario que va a utilizar la aplicación. El login tiene una longitud de 8 caracteres y está compuesto únicamente por letras minúsculas.
  - Solicita al usuario el nombre de un fichero que quiere mostrar. El nombre del fichero es como máximo de 8 caracteres y tiene una extensión de 3 caracteres.
  - Visualiza en pantalla el contenido del fichero.
  - Realiza una validación exhaustiva de los datos de entrada.
  - Lleva un registro (log) de la actividad del programa.

Actividad 6.2

  - Utilizando la aplicación desarrollada en la actividad anterior, configura las políticas de acceso para:
    - Firmar digitalmente la aplicación.

📊 Criterios de Calificación: Total 10 puntos

  - **Actividad 6.1:** 5 puntos
  - **Actividad 6.2:** 5 puntos

🧰 Recursos necesarios

  - Entorno de desarrollo Java (por ejemplo, Eclipse).

✨ Consejos y Recomendaciones

  - Lee detenidamente el contenido de la unidad antes de comenzar.
  - Valida siempre las entradas del usuario.
  - Incluye trazas de depuración o registros de actividad (log).
  - Utiliza herramientas disponibles para firmar digitalmente la aplicación.

🛠️ Indicaciones de entrega

  - Debes entregar un fichero comprimido (.zip) que contenga:
    - Explicaciones breves de cómo has realizado las actividades.
    - Capturas de pantalla que estimes oportunas.
    - El o los proyectos desarrollados.
    - Un documento (ODT, DOCX o PDF) con:

🔧 Resultados de aprendizaje y criterios de evaluación

Protege las aplicaciones y los datos definiendo y aplicando criterios de seguridad en el acceso, almacenamiento y transmisión de la información

a) Se han identificado y aplicado principios y prácticas de programación segura.
b) Se han analizado las principales técnicas y prácticas criptográficas.
c) Se han definido e implantado políticas de seguridad para limitar y controlar el acceso de los usuarios a las aplicaciones desarrolladas.
d) Se han utilizado esquemas de seguridad basados en roles.
e) Se han empleado algoritmos criptográficos para proteger el acceso a la información almacenada.
f) Se han identificado métodos para asegurar la información transmitida.
g) Se han desarrollado aplicaciones que utilicen sockets seguros para la transmisión de información.
h) Se han depurado y documentado las aplicaciones desarrolladas.

Resolución de la tarea:

Para la resolución de esta tarea se va a crear un proyecto denominado Validación, esta aplicación solicitará al usuario por consola el nombre de usuario y un nombre de archivo. Se registrarán todos los sucesos que se realicen en la aplicación. Una vez se haya validado el usuario y el archivo, se comprobará si el archivo existe, si el archivo existe se mostrará su contenido.

Estructura del proyecto

Nuestro proyecto tendrá la siguiente estructura para gestionar los archivos de forma correcta y optima.

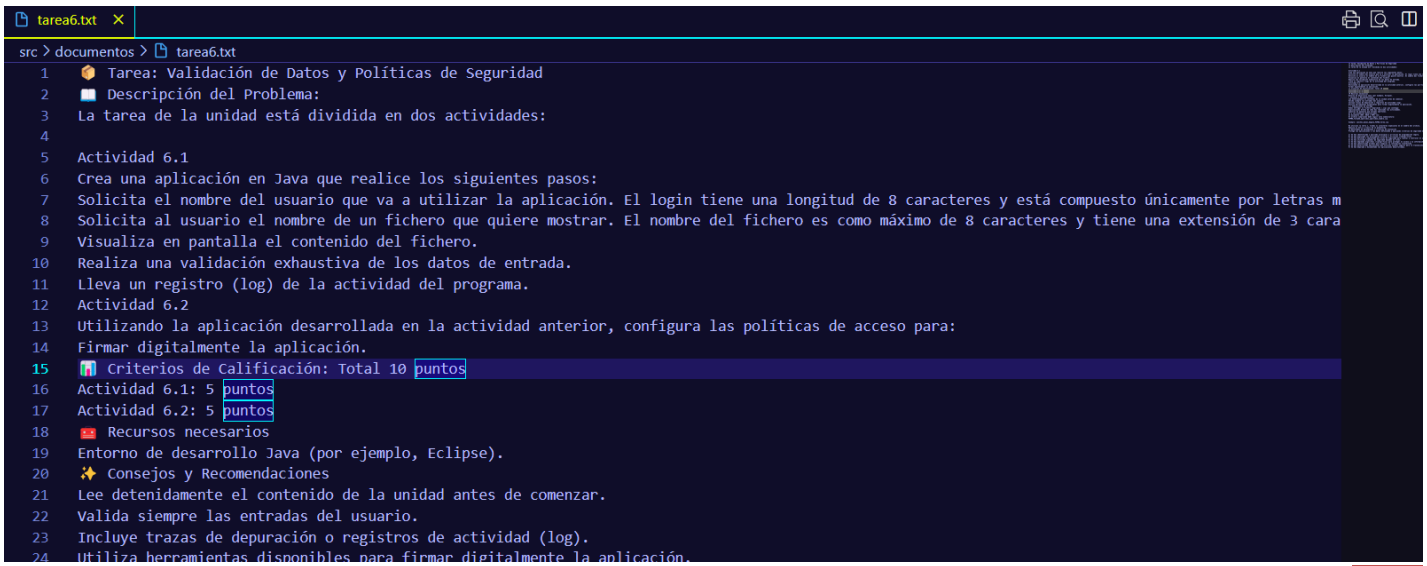
La estructura contendrá 3 directorios para gestionar los diferentes archivos necesarios. El primero directorio “documentos” almacenará los archivos a los que el usuario podrá acceder. Se creará un fichero con la descripción de la tarea.

El segundo denominado “logs” aunque se ha creado como estructura principal este directorio, se controla por para en caso de que no exista se creará el directorio. En este directorio se almacenan los registros que suceden durante la ejecución del programa.

El tercero se denomina “validación”, en este directorio se almacenarán las clases para crear la funcionalidad de la aplicación.

Archivo tarea6.txt

Se creará un archivo denominado “tarea6.txt”, este archivo debe tener un máximo de 8 caracteres para el nombre y 3 para la extensión. En este fichero se encontrará la descripción de la tarea, y cuando el usuario se haya validado correctamente, se mostrará el contenido del archivo.



Clases para el proyecto

Se crearán 3 clases Java para el proyecto, una será la clase main que permitirá ejecutar la aplicación, otra para para gestionar la lógica de la aplicación y otra para gestionar los Registros (logs) que puedan surgir durante la ejecución del programa.

A continuación, se van a describir las diferentes clases empezando por Registros.java, seguidametne DocumentHandler.java y por último, la clase main ValidacionApp.java.

Registros.java

En esta clase se va a controlar el tema de los registros, desde su creación, a la configuración y registros de todos los presos que puedan suceder en el proyecto.

En esta clase se declararán dos variables para almacenar la ruta y fichero donde se registrarán los registros, un logger, y seis variables constantes para los mensajes comunes. Con estas últimas seis variables podremos mejorar editar de forma más adecuada en caso de ser necesario los textos comunes y evitamos tener posibles errores de escritura.

```
// Se declaran las variables y objetos necesarios.
private final String DIRECTORIO_BASE;
private final String NOMBRE_ARCHIVO_LOG;
private Logger logger;
// Constantes para mensajes comunes
private final String LOG_PREFIX_ERROR = "ERROR: ";
private final String LOG_PREFIX_WARNING = "ADVERTENCIA: ";
private final String LOG_SISTEMA_CONFIGURADO = "Sistema de logging configurado correctamente";
private final String LOG_DIRECTORIO_CREADO = "Directorio creado: %s";
private final String LOG_ARCHIVO_CREADO = "Archivo de log creado: %s";
private final String LOG_ERROR_CONFIGURACION = "Error al configurar el logger: %s";
```

Se creará un constructor donde instanciaremos las variables para la ruta y para el nombre del archivo que almacenará los registros. También se instanciará el logger. Se llamará al método crearDirectorio, crearArchivoLog y configurarLogger.

```
public Registros() {
    // Se obtiene el directorio base del proyecto y se establece la ruta
    this.DIRECTORIO_BASE = System.getProperty(key:"user.dir") + "/src/logs/";
    // crear el nombre del archivo de log
    this.NOMBRE_ARCHIVO_LOG = "logger.log";
    // Se crea el logger
    this.logger = Logger.getLogger(Registros.class.getName());
    // Se llama al método para crear el directorio de logs
    crearDirectorio(DIRECTORIO_BASE);
    // Se llama al método para crear el archivo de log
    crearArchivoLog(NOMBRE_ARCHIVO_LOG);
    // Se llama al método para configurar el logger
    configurarLogger(NOMBRE_ARCHIVO_LOG);
}
```

El método crearDirectorio será privado y recibirá un String con el directorio donde se va a almacenar el fichero para los registros. Este método se llamará cada vez que se instancia el constructor de la clase para comprobar si existe la ruta donde se va a almacenar el fichero log, en caso de que no exista lo creará y si existe no hace nada.

```
private void crearDirectorio(String nombreDirectorio) {
    // Se controla las excepciones
    try {
        // Se instancia un objeto File con la ruta del directorio
        File directorio = new File(nombreDirectorio);
        // Se verifica si el directorio no existe
        if (!directorio.exists()) {
            // Se crea el directorio
            directorio.mkdirs();
            // Se registra un mensaje de información en el log
            logger.info(String.format(LOG_DIRECTORIO_CREADO, nombreDirectorio));
        }
    } catch (Exception e) {
        // Se muestra un mensaje de error en la consola y se imprime la traza de la
        // excepción
        logger.severe(LOG_PREFIX_ERROR + e.getMessage());
        e.printStackTrace();
    }
}
```

El método crearArchivoLog será privado, recibe el nombre del fichero que se va a crear, en caso de que este no exista se creará de lo contrario no hará nada.

```
private void crearArchivoLog(String nombreArchivo) {
    // Se controla las excepciones
    try {
        // Se instancia un objeto File con la ruta del archivo de log
        File archivoLog = new File(DIRECTORIO_BASE + nombreArchivo);
        // Se verifica si el archivo de log no existe
        if (!archivoLog.exists()) {
            // Se crea el archivo de log
            archivoLog.createNewFile();
            // Se muestra un mensaje de información en el log
            logger.info(String.format(LOG_ARCHIVO_CREADO, nombreArchivo));

        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

El método configurarLogger recibe un nombre del archivo para configurarlo. De este modo se insertará la información simplificada, evitando que se muestre datos no esenciales a la hora de su conulta. Este método también será privado.

```
private void configurarLogger(String nombreArchivo) {
    // Se controla las excepciones
    try {
        // Se crear el FileHandler con la ruta completa
        FileHandler fileHandler = new FileHandler(DIRECTORIO_BASE + nombreArchivo, append:true);
        // Se establece el formato del log con el formato SimpleFormatter
        fileHandler.setFormatter(new SimpleFormatter());
        // Se añade el FileHandler al logger
        logger.addHandler(fileHandler);
        // Se establece el nivel de log a ALL para registrar todos los mensajes
        logger.setLevel(Level.ALL);
        // Se desactiva el uso de los handlers del padre para evitar duplicados
        logger.setUseParentHandlers(useParentHandlers:false);
        // Se registra un mensaje de información en el log
        logger.info(LOG_SISTEMA_CONFIGURADO);
    } catch (IOException e) {
        System.err.println(LOG_PREFIX_ERROR + LOG_ERROR_CONFIGURACION.replace(target:"{}", e.getMessage()));
        e.printStackTrace();
    }
}
```

Habrá tres métodos que registrarán la información según el tipo de incidente en la ejecución de la aplicación, uno será para los registros, otro para las advertencias, estas pueden ser que el nombre introducido no es válido, y el último, será para registrar los errores.

```
/**
 * Método para escribir un mensaje en el archivo de log.
 *
 * @param mensaje Mensaje a escribir en el log.
 */
public void registroLog(String mensaje) {
    logger.info(mensaje);
}

/**
 * Método para escribir un mensaje de error en el archivo de log.
 *
 * @param mensaje Mensaje de error a escribir en el log.
 */
public void errorLog(String mensaje) {
    logger.severe(LOG_PREFIX_ERROR + mensaje);
}

/**
 * Método para escribir un mensaje de alerta en el archivo log.
 *
 * @param mensaje Mensaje de alerta a escribir en el log.
 */
public void warningLog(String mensaje) {
    logger.warning(LOG_PREFIX_WARNING + mensaje);
}
```

El último método cerrarLogger, cierra el logger y libera recursos.

```java
/**
 * Método para cerrar el logger y liberar los recursos.
 */
public void cerrarLogger() {
    for (Handler handler : logger.getHandlers()) {
        handler.close();
    }
}
```

DocumentHandler.java

En esta clase se va gestionar la lógica de la aplicación, como la validación de los datos introducidos por el usario, y la apertura y lectura del fichero al que quiere acceder.

Se instanciará un objeto de la clase Registros, que hemos creado anteriormente, de este modo se registrará todos los procesos de la ejecución de la aplicación.

En el constructor de la clase se instanciará el objeto Registros para inicializar los registros.

```java
// Se declaran las variables de la clase
private Registros registros;

// Constructor de la clase DocumentHandler
public DocumentHandler() {
    this.registros = new Registros();
}
```

Método solicitarUsuario, este método comprobará que los datos introducidos por el usuario corresponden con el formato indicado. El nombre de usuario debe tener 8 caracteres, y solo debe contener letras en minúscula.

Antes de comprobar que se cumpla la condición del nombre, limpiará la información introducida por el usuario, para evitar que se introduzcan caracteres que puedan ser peligrosos para nuestro código o base de datos.

El código se encontrará en un bucle infinito que se finalizará cuando el usuario haya introducido un nombre válido.

```java
public void solicitarUsuario(Scanner scanner) {
    // Se declara la variable usuario
    String usuario;
    // Se inicia un bucle infinito para solicitar el nombre de usuario
    while (true) {
        // Se solicita el nombre de usuario al usuario
        System.out.print(s:"Introduce tu nombre de usuario (8 caracteres, solo letras minúsculas): ");
        // Se lee la entrada del usuario y se eliminan caracteres peligrosos
        usuario = limpiarEntrada(scanner.nextLine());
        // Se valida el nombre de usuario
        if (usuario.matches(regex:"[a-z]{8}")) {
            // Se indica que el nombre introducido es v
            System.out.println(String.format(format:"El nombre que ha introducido es válido: ", usuario));
            // Si el nombre de usuario es válido, se registra en el log
            registros.registroLog("Usuario válido: " + usuario);
            // Se sale del bucle
            break;
            // Si el nombre de usuario no es válido, se muestra un mensaje de error
        } else {
            System.out.println(
                    x:"El nombre de usuario no es válido. Debe tener 8 caracteres y solo letras minúsculas.");
            // Se registra el intento de usuario inválido en el log
            registros.warningLog("Intento de usuario inválido: " + usuario);
        }
    }

}
```

El Método solicitarArchivo, recibe por parte del usuario el nombre del archivo y este método comprobará si cumple con los requisitos indicados. Una vez comprobado que el nombre introducido es válido se devolverá para leer el archivo, y mostrarlo por consola.

También se limpiará el nombre del archivo, para evitar que el usuario haya introducido caracteres peligrosos para nuestro código y base de datos.

El código estará dentro de un bucle infinito que se mantendrá ejecutado hasta que el usuario introduzca un nombre de archivo correcto.

```java
public String solicitarArchivo(Scanner scanner) {
    // Se declara la variable nombreArchivo
    String nombreArchivo;
    // Se inicia un bucle infinito para solicitar el nombre del archivo
    while (true) {
        // Se solicita el nombre del archivo al usuario
        System.out.println(
                x:"El nombre del archivo debe tener como máximo 8 caracteres, un punto y 3 caracteres para la extensión. ejemplo: archivo.txt");
        System.out.print(s:"Introduce el nombre del archivo: ");
        // Se lee la entrada del usuario y se eliminan caracteres peligrosos
        nombreArchivo = limpiarEntrada(scanner.nextLine());
        // Se valida el nombre del archivo
        if (validarAccesoArchivo(nombreArchivo)) {
            // Si el nombre del archivo es válido, se registra en el log
            registros.registroLog("Nombre de archivo válido: " + nombreArchivo);
            // Se sale del bucle
            break;
        } else {
            // Se registra el intento de archivo inválido en el log
            registros.warningLog("Intento de archivo inválido: " + nombreArchivo);
        }
    }
    // Se devuelve el nombre del archivo validado
    return nombreArchivo;
}
```

Una vez el usuario haya introducido los datos válidos se llamará al método mostrarContendioArchivo. Este método comprobará si existe el archivo indicado por el usuario, esto se hará mediante el otro método que es llamado por este. Una vez se haya confirmado que existe, leerá el archivo y mostrará por pantalla.

```java
public void mostarContendioArchivo(String nombreArchivo) {
    // Se obtiene la ruta segura del archivo
    Path rutaArchivo = obtenerRutaSegura(nombreArchivo);
    // Se verifica si la ruta es nula
    if (rutaArchivo == null) {
        return;
    }
    // Se intenta leer el contenido del archivo
    try (BufferedReader br = Files.newBufferedReader(rutaArchivo)) {
        System.out.println(x:"\nContenido del archivo:");
        String linea;
        // Se lee línea por línea el contenido del archivo
        while ((linea = br.readLine()) != null) {
            System.out.println(linea);
        }
        // Se registra en el log que el archivo se ha leído correctamente
        registros.registroLog("Archivo leído correctamente: " + nombreArchivo);
    } catch (IOException e) {
        // Si ocurre un error al leer el archivo, se muestra un mensaje de error y se
        // registra en el log
        System.err.println("Error al leer el archivo: " + e.getMessage());
        registros.warningLog("Error al leer el archivo: " + e.getMessage());
    }
}
```

Para evitar que el usuario introducir caracteres peligrosos, se ha desarrollado un método que eliminará esos caracteres a través del método limpiarEntrada.

```java
private String limpiarEntrada(String input) {
    return input.replaceAll(regex:"[<>\"'&]", replacement:"");
}
```

Se ha desarrollado un método para obtener una ruta segura para acceder al archivo, además verificará que el archivo existe. En caso de no poder generar la ruta se devolverá null.

```
private Path obtenerRutaSegura(String nombreArchivo) {
    // Se obtiene la ruta base del proyecto
    String directorioBase = System.getProperty(key:"user.dir") + "/src/documentos/";
    // Se crea la ruta del archivo
    Path rutaArchivo = Paths.get(directorioBase, nombreArchivo).normalize();
    // Se verifica si la ruta es válida
    if (rutaArchivo.toAbsolutePath().startsWith(directorioBase)) {
        return rutaArchivo;
    } else {
        return null;
    }
}
```

También habrá un método para validar el formato, si existe y el acceso del al archivo. Este método obtendrá una ruta segura, si la ruta es null se finaliza el proceso y se devuelve false. También se comprobará si se tiene permisos de lectura para el archivo.

```
private boolean validarAccesoArchivo(String nombreArchivo) {

    // Se compureba el formato
    if (!nombreArchivo.matches(regex:"^[a-zA-Z0-9]{1,8}\\.[a-zA-Z0-9]{3}$")) {
        System.out.println(x:"El formato del nombre del archivo introducido no es válido.");
        return false;
    }

    // Se obtiene la ruta segura del archivo
    Path rutaArchivo = obtenerRutaSegura(nombreArchivo);
    // Se verifica si la ruta es nula
    if (rutaArchivo == null) {
        return false;
    }
    // Se registra en el log la verificación del archivo
    registros.registroLog("Verificando archivo en: " + rutaArchivo.toAbsolutePath());

    // Se verifica si el archivo existe
    if (!Files.exists(rutaArchivo)) {
        // Si el archivo no existe, se mostrará un mensaje informado de que no existe el
        // archivo.
        System.out.println("El archivo no existe en la ruta: " + rutaArchivo);
        // Si el archivo no existe, se registra en el log y se devuelve false
        registros.warningLog("El archivo no existe: " + rutaArchivo);
        return false;
    }

    // Se verifica si el archivo tiene permisos de lectura
    if (!Files.isReadable(rutaArchivo)) {
        // Si no se tiene permisos de lectura, se mostrará un mensaje
        System.out.println("El archivo no tiene permisos de lectura: " + rutaArchivo);
        // Si no tiene permisos de lectura, se registra en el log y se devuelve false
        registros.warningLog("El archivo no tiene permisos de lectura: " + rutaArchivo);
        return false;
    }
    // Si el archivo es accesible, se registra en el log y se devuelve true
    registros.registroLog("El archivo es accesible: " + nombreArchivo);
    return true;
}
```

Si se cumplen todos los requisitos se devolverá true para continuar con el proceso de la aplicación.

Por último, se ha desarrollado un método para finalizar el proceso de registros. Este método llamá al método de registros que cierra el fichero para los registros.

```
public void cerrarRegistro() {
    // Se llama al método que finaliza el proceso de registro.
    this.registros.cerrarLogger();

}
```

ValidacionApp.java (Clase Main)

Por último, tendremos una clase para llamar a los métodos de la clase DocumentHandler.java. En esta clase se instanciará un Objeto de la clase DocumentoHandler para controlar para poder llamar a los métodos públicos de la clase.

En el método main se instanciará un objeto Scanner para que poder solicitar a los usuarios los datos. Seguidamente se ira llamando a los métodos que controlarán los datos introducidos por el usuario.

Al finalizar todo el proceso se va a cerrar el fichero del registro y el scanner.

```java
private static final DocumentHandler documentHandler = new DocumentHandler();

/**
 * Método principal que ejecuta el programa.
 *
 * @param args argumentos de línea de comandos (no utilizados)
 */
Run | Debug
public static void main(String[] args) {
    // Se crea un objeto Scanner para leer la entrada del usuario
    Scanner scanner = new Scanner(System.in);
    // Se solicita el nombre de usuario al usuario
    documentHandler.solicitarUsuario(scanner);
    // Se solicita el nombre del archivo al usuario
    String nombreArchivo = documentHandler.solicitarArchivo(scanner);
    // Se lee el contenido del archivo y se muestra en la consola
    documentHandler.mostarContendioArchivo(nombreArchivo);
    // Se llama al método para finalizar el fichero de registros.
    documentHandler.cerrarRegistro();
    // Se cierra el objeto Scanner
    scanner.close();
}
```

Ejecución de la App

Con esto ya tendríamos todo preparado para nuestra aplicación, a continuación, se muestra el uso de la aplicación.

```
C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion> cmd /C ""C:\Program File
rea_6\Validacion\bin validacion.ValidacionApp "
may 07, 2025 6:17:20 P.◆M. validacion.Registros crearArchivoLog
INFO: Archivo de log creado: logger.log
Introduce tu nombre de usuario (8 caracteres, solo letras min◆sculas):
```

La aplicación esta informando que ha creado el archivo para registrar los datos y nos esta solicitando el nombre del usuario. Ahora vamos a introducir un nombre que no cumpla con el requisito indicado.

```
may 07, 2025 6:17:20 P.◆M. validacion.Registros crearArchivoLog
INFO: Archivo de log creado: logger.log
Introduce tu nombre de usuario (8 caracteres, solo letras min◆sculas): Diogenes
El nombre de usuario no es v◆lido. Debe tener 8 caracteres y solo letras min◆sculas.
Introduce tu nombre de usuario (8 caracteres, solo letras min◆sculas):
```

Como se puede observar, aunque el nombre tiene 8 letras, como una es mayúscula ya no se cumple y devuelve el error y solicita de nuevo el nombre. Ahora vamos a introducir el nombre correcto

```
Introduce tu nombre de usuario (8 caracteres, solo letras min◆sculas): Diogenes
El nombre de usuario no es v◆lido. Debe tener 8 caracteres y solo letras min◆sculas.
Introduce tu nombre de usuario (8 caracteres, solo letras min◆sculas): diogenes
El nombre que ha introducido es v◆lido:
El nombre del archivo debe tener como m◆ximo 8 caracteres, un punto y 3 caracteres para la extensi◆n. ejemplo: archivo.txt
Introduce el nombre del archivo:
```

Como se puede observar nos ha validado el nombre y ahora nos esta solicitando el nombre del archivo.

A continuación, se va a introducir el nombre del archivo sin cumplir de diferentes formas el formato indicado.

```
El nombre del archivo debe tener como m♦ximo 8 caracteres, un punto y 3 caracteres para la extensi♦n. ejemplo: archivo.txt
Introduce el nombre del archivo: smi
El formato del nombre del archivo introducido no es v♦lido.
El nombre del archivo debe tener como m♦ximo 8 caracteres, un punto y 3 caracteres para la extensi♦n. ejemplo: archivo.txt
Introduce el nombre del archivo: demandade
El formato del nombre del archivo introducido no es v♦lido.
El nombre del archivo debe tener como m♦ximo 8 caracteres, un punto y 3 caracteres para la extensi♦n. ejemplo: archivo.txt
Introduce el nombre del archivo: demandade.txt
El formato del nombre del archivo introducido no es v♦lido.
El nombre del archivo debe tener como m♦ximo 8 caracteres, un punto y 3 caracteres para la extensi♦n. ejemplo: archivo.txt
Introduce el nombre del archivo:
```

Ahora se va a proceder a introducir un nombre de archivo que existe dentro de la carpeta documentos.

```
El nombre del archivo debe tener como m♦ximo 8 caracteres, un punto y 3 caracteres para la extensi♦n. ejemplo: archivo.txt
Introduce el nombre del archivo: tarea6.txt

Contenido del archivo:
? Tarea: Validaci♦n de Datos y Pol♦ticas de Seguridad
? Descripci♦n del Problema:
La tarea de la unidad est♦ dividida en dos actividades:

Actividad 6.1
Crea una aplicaci♦n en Java que realice los siguientes pasos:
Solicita el nombre del usuario que va a utilizar la aplicaci♦n. El login tiene una longitud de 8 caracteres y est♦ compuesto ♦nicamente por letras min♦sculas.
Solicita al usuario el nombre de un fichero que quiere mostrar. El nombre del fichero es como m♦ximo de 8 caracteres y tiene una extensi♦n de 3 caracteres.
Visualiza en pantalla el contenido del fichero.
Realiza una validaci♦n exhaustiva de los datos de entrada.
Lleva un registro (log) de la actividad del programa.
Actividad 6.2
Utilizando la aplicaci♦n desarrollada en la actividad anterior, configura las pol♦ticas de acceso para:
Firmar digitalmente la aplicaci♦n.
? Criterios de Calificaci♦n: Total 10 puntos
Actividad 6.1: 5 puntos
```

Como se puede observar en la imagen anterior se ha leído todo el archivo y se ha mostrado por consola.

Veamos el registro de los datos introducidos dentro.

```
may 07, 2025 6:17:20 P.♦M. validacion.Registros configurarLogger
INFO: Sistema de logging configurado correctamente
may 07, 2025 6:19:27 P.♦M. validacion.Registros warningLog
WARNING: ADVERTENCIA: Intento de usuario inv♦lido: Diogenes
may 07, 2025 6:20:49 P.♦M. validacion.Registros registroLog
INFO: Usuario v♦lido: diogenes
may 07, 2025 6:22:37 P.♦M. validacion.Registros registroLog
INFO: Verificando archivo en: C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion\src\documentos\archivosdelatarea.t
may 07, 2025 6:22:37 P.♦M. validacion.Registros warningLog
WARNING: ADVERTENCIA: El archivo no existe: C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion\src\documentos\archi
may 07, 2025 6:22:37 P.♦M. validacion.Registros warningLog
WARNING: ADVERTENCIA: Intento de archivo inv♦lido: archivosdelatarea.txt
may 07, 2025 6:37:58 P.♦M. validacion.Registros configurarLogger
INFO: Sistema de logging configurado correctamente
may 07, 2025 6:38:02 P.♦M. validacion.Registros registroLog
INFO: Usuario v♦lido: diogenes
may 07, 2025 6:38:12 P.♦M. validacion.Registros registroLog
INFO: Verificando archivo en: C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion\src\documentos\archivospara.txt
may 07, 2025 6:38:12 P.♦M. validacion.Registros warningLog
WARNING: ADVERTENCIA: El archivo no existe: C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion\src\documentos\archi
may 07, 2025 6:38:12 P.♦M. validacion.Registros warningLog
WARNING: ADVERTENCIA: Intento de archivo inv♦lido: archivospara.txt
may 07, 2025 6:42:00 P.♦M. validacion.Registros configurarLogger
INFO: Sistema de logging configurado correctamente
may 07, 2025 6:42:05 P.♦M. validacion.Registros registroLog
INFO: Usuario v♦lido: diogenes
may 07, 2025 6:42:08 P.♦M. validacion.Registros registroLog
INFO: Verificando archivo en: c:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion\src\documentos\archivospara.txt
may 07, 2025 6:42:08 P.♦M. validacion.Registros warningLog
WARNING: ADVERTENCIA: El archivo no existe: c:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion\src\documentos\archi
may 07, 2025 6:42:08 P.♦M. validacion.Registros warningLog
WARNING: ADVERTENCIA: Intento de archivo inv♦lido: archivospara.txt
may 07, 2025 6:43:27 P.♦M. validacion.Registros configurarLogger
INFO: Sistema de logging configurado correctamente
may 07, 2025 6:43:36 P.♦M. validacion.Registros registroLog
INFO: Usuario v♦lido: diogenes
may 07, 2025 6:43:53 P.♦M. validacion.Registros registroLog
INFO: Verificando archivo en: C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion\src\documentos\diogenesde
may 07, 2025 6:43:53 P.♦M. validacion.Registros warningLog
```

Resolución del apartado 2 de la tarea.

Abrimos el powerShell desde Visual Studio, y a través de los siguientes comandos generaremos el certificado y podremos firma nuestra aplicación.

Lo primero que vamos a realizar es compilar nuestra aplicación:

```
PS C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion> javac -d out src\validacion\*.java
```

El siguiente paso será crear el archivo .jar para ejecutar nuestra aplicación

```
PS C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion> jar cfe ValidacionApp.jar validacion.ValidacionApp -C out .
```

Ahora crearemos un keystore. Esto sirve para crear un certificado, en este caso será de 365 días de duración.

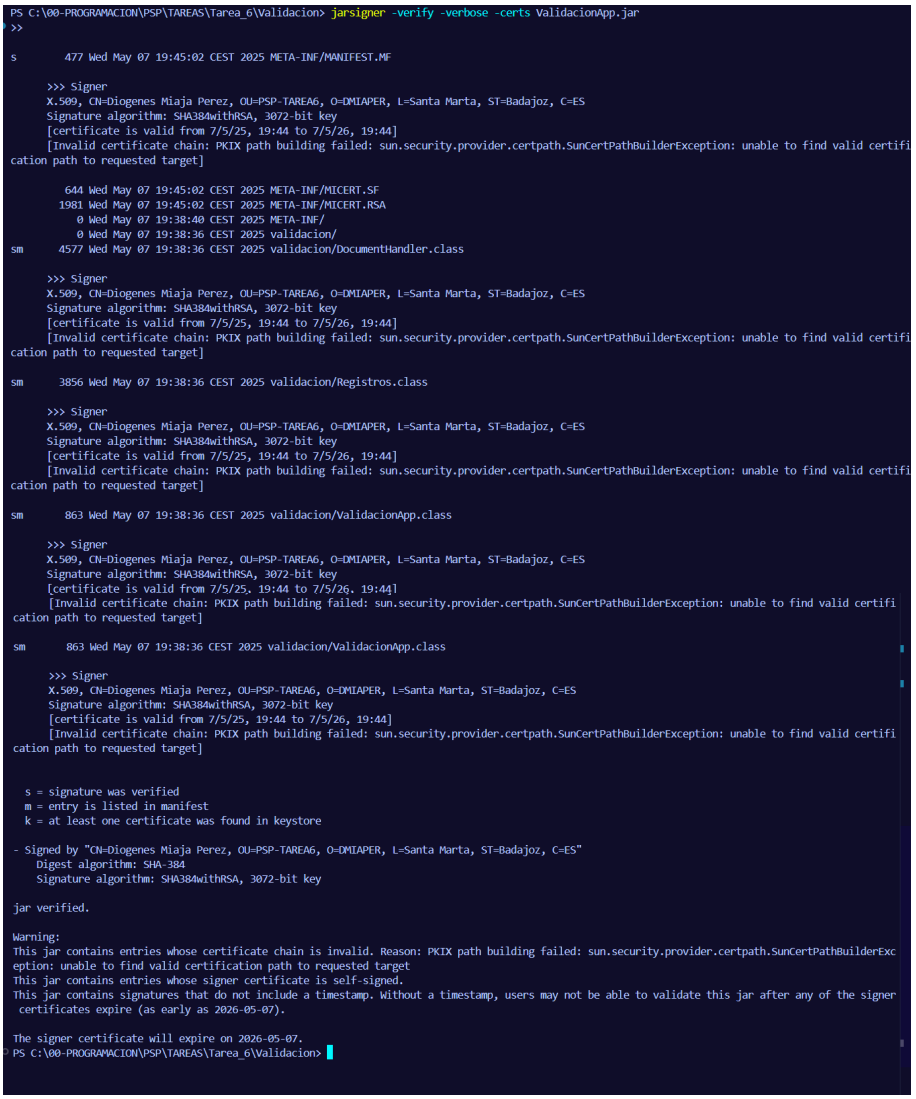
```
PS C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion> keytool -genkeypair -alias micert -keyalg RSA -keystore mi_keystore.jks -storepass 123456 -validity 365
>>
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
What is your first and last name?
  [Unknown]:  Diogenes Miaja Perez
What is the name of your organizational unit?
  [Unknown]:  PSP-TAREA6
What is the name of your organization?
  [Unknown]:  DMIAPER
What is the name of your City or Locality?
  [Unknown]:  Santa Marta
What is the name of your State or Province?
  [Unknown]:  Badajoz
What is the two-letter country code for this unit?
  [Unknown]:  ES
Is CN=Diogenes Miaja Perez, OU=PSP-TAREA6, O=DMIAPER, L=Santa Marta, ST=Badajoz, C=ES correct?
  [no]:  yes
```

Como podemos observar en la imagen anterior se han introducido los datos de identificación del certificado.

Tras la creación del certificado, se va proceder a la firma del fichero.

```
PS C:\00-PROGRAMACION\PSP\TAREAS\Tarea_6\Validacion> jarsigner -keystore mi_keystore.jks -storepass 123456 ValidacionApp.jar micert
>>
jar signed.
```

Por último, se va a comprobar si está firmado el archivo jar.



Al final de la comprobación nos indica cuando expira la certificación.