

Enunciado.

La tarea de la unidad está dividida en dos actividades.

Actividad 6.1. Crea una aplicación 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.**

Para que se cumpla la condición de longitud y contenido del login, se ha creado la clase validar usuario, que comprueba que el String pasado como parámetro contenga letras minúsculas de la “a” a la “z”, y tenga una longitud entre 1 y 8 caracteres:

```
public static boolean validarUsuario(String nombre){  
  
    return nombre.matches(regex: "[a-z]{1,8}");  
}
```

Esta clase se implementa en el main validando que el formato de usuario sea el correcto. En caso de que sea incorrecto se muestra un mensaje por pantalla, y se vuelve a pedir la entrada, mediante el bucle do while:

```
public class Main {  
  
    public static void main(String[] args) throws IOException {  
  
        Scanner sc = new Scanner(source: System.in);  
        sc.useDelimiter(pattern: "\n");  
        String nombreUsuario;  
  
        System.out.println(x: "Indique el nombre del usuario");  
  
        do{  
            nombreUsuario = sc.next();  
            System.out.println(x: "");  
  
            if (!validarUsuario(nombre: nombreUsuario)) {  
                System.out.println(x: "El nombre del usuario debe componerse de 8 letras minusculas");  
                log("El nombre del usuario debe componerse de 8 letras minusculas " + nombreUsuario);  
            }  
  
        }while(!validarUsuario(nombre: nombreUsuario));  
        log("Usuario ingresado correctamente: " + nombreUsuario);  
    }  
}
```

- **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.**

Para que se cumpla la condición de longitud y contenido del nombre del archivo y su extensión, se ha creado la clase validarFichero, que comprueba que el String pasado como parámetro contenga letras minúsculas o mayúsculas de la “a” a la “z”, o números, y tenga una longitud entre 1 y 8 caracteres, y que su extensión cumpla la misma condición pero que esté compuesta por 3 caracteres.

```
public static boolean validarFichero(String nombre){  
  
    return nombre.matches(regex: "[a-zA-Z0-9]{1,8}\\.[a-zA-Z0-9]{3}");  
}
```

Esta clase se implementa en el main validando que el formato del nombre del fichero sea el correcto. En caso de que sea incorrecto se muestra un mensaje por pantalla, y se vuelve a pedir la entrada, mediante el bucle do while:

```
String nombreFichero;

System.out.println(x: "Indique el nombre del fichero");

do{
    nombreFichero = sc.next();

    if (!validarFichero(nombre: nombreFichero)) {
        System.out.println(x: "El nombre del fichero debe componerse de 8 caracteres y una extension con 3 caracteres. Ejemplo: ejemplo.txt");
        log("El nombre del fichero debe componerse de 8 caracteres y una extension con 3 caracteres " + nombreFichero);
    }

}while(!validarFichero(nombre: nombreFichero));
log("Nombre de fichero ingresado correctamente: " + nombreFichero);
```

- **Visualiza en pantalla el contenido del fichero.**

Para visualizar el contenido del fichero, se ha creado la clase leerFichero(), esta clase recoge en un BufferedReader el fichero pasado por parámetro y lo muestra línea a línea a partir de un bucle while.

```
public static void leerFichero(String nombreFichero) throws FileNotFoundException, IOException{

    BufferedReader br = new BufferedReader(new FileReader(fileName: nombreFichero));

    String linea;
    while((linea = br.readLine()) != null){

        System.out.println(x: linea);
    }

}
```

Se implementa en la clase main de manera que lance una excepción en caso de no encontrar el fichero pasado por parámetro:

```
try {
    leerFichero(nombreFichero);
} catch (IOException ex) {
    Logger.getLogger(name: Main.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
}
```

Es importante tener en cuenta que se tiene que realizar una validación de los datos de entrada y llevar un registro de la actividad del programa.

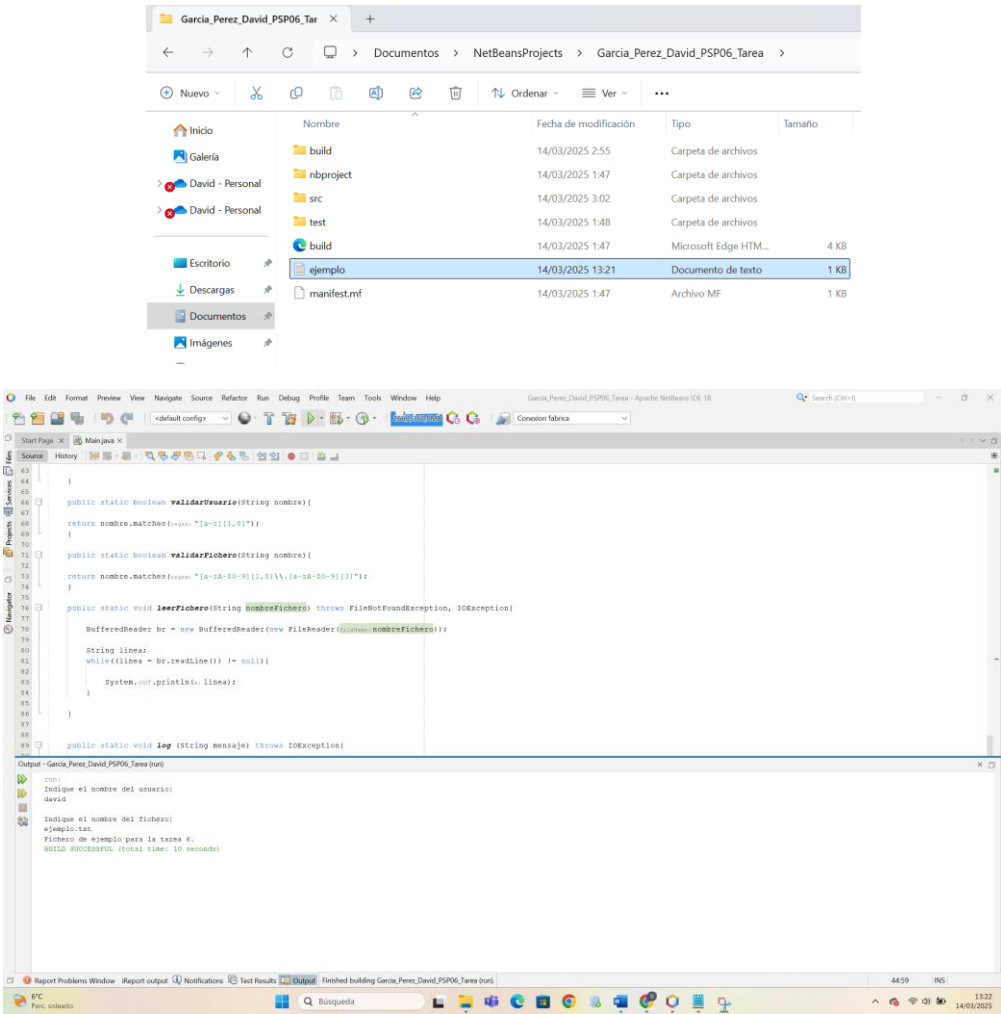
Para llevar un registro de la actividad que realiza el programa, se ha creado la clase log, que va escribiendo en un fichero registro.log, las acciones que va realizando la aplicación mediante un formato estándar que incluye la fecha y hora y a la que se le añade el mensaje pasado por parámetro:

```
public static void log (String mensaje) throws IOException{

    try (PrintWriter pw = new PrintWriter(new FileWriter(fileName: "registro.log", append: true))) {
        pw.println("[ " + LocalDateTime.now().format(formatter: DateTimeFormatter.ofPattern(pattern: "dd-MM-yyyy HH:mm:ss")) + " ] " + mensaje);
    }

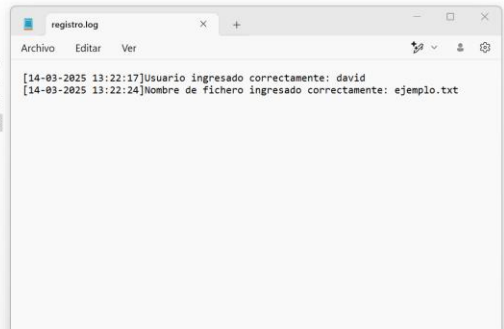
}
```

Prueba de la aplicación:



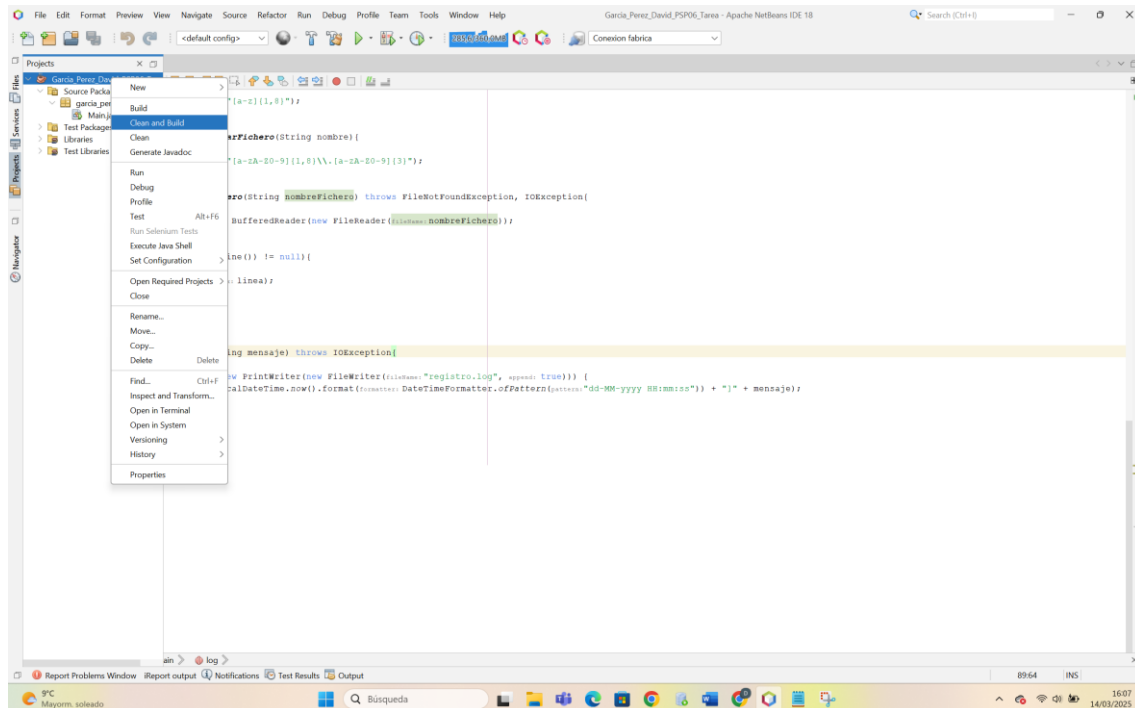
build	14/03/2025 2:55	Carpeta de archivos	
nbproject	14/03/2025 1:47	Carpeta de archivos	
src	14/03/2025 3:02	Carpeta de archivos	
test	14/03/2025 1:48	Carpeta de archivos	
build	14/03/2025 1:47	Microsoft Edge HTM...	4 KB
ejemplo	14/03/2025 13:21	Documento de texto	1 KB
manifest.mf	14/03/2025 1:47	Archivo MF	1 KB
registro	14/03/2025 13:22	Documento de texto	1 KB

build	14/03/2025 2:55	Carpeta de archivos	
nbproject	14/03/2025 1:47	Carpeta de archivos	
src	14/03/2025 3:02	Carpeta de archivos	
test	14/03/2025 1:48	Carpeta de archivos	
build	14/03/2025 1:47	Microsoft Edge HTM...	4 KB
ejemplo	14/03/2025 13:21	Documento de texto	1 KB
manifest.mf	14/03/2025 1:47	Archivo MF	1 KB
registro	14/03/2025 13:22	Documento de texto	1 KB



Actividad 6.2. Utilizando la aplicación desarrollada en la actividad anterior, configura las políticas de acceso para:

Lo primero que debemos hacer es limpiar y construir para generar el archivo .jar de esta aplicación:



Una vez obtenido este archivo probamos su ejecución en consola y comprobamos que funciona correctamente, en este caso he tenido que añadir un `.trim()` a las entradas recogidas para que las detectara correctamente:

```
nombreUsuario = sc.next().trim();
System.out.println(" ");

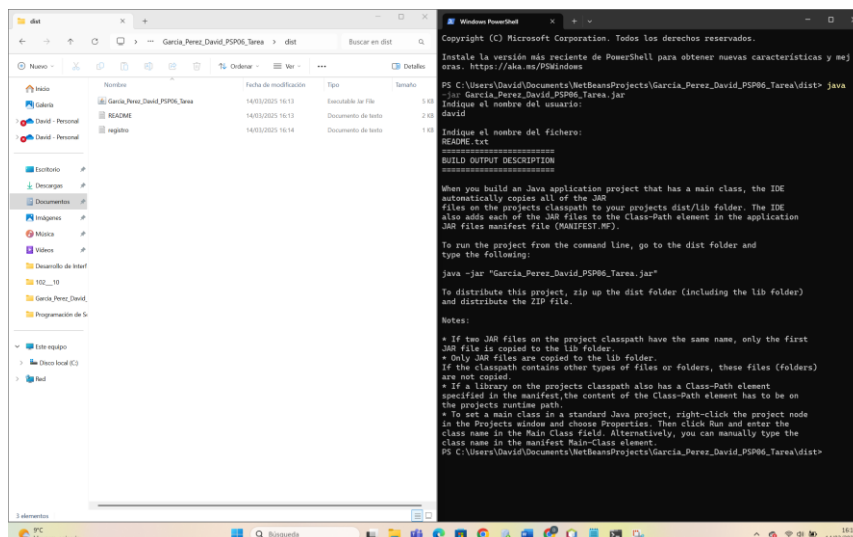
if (!validarUsuario(nombreUsuario)) {
    System.out.println("El nombre del usuario no es válido");
}

while (!validarUsuario(nombreUsuario)) {
    System.out.println("Indique el nombre del usuario correctamente:");
}

String nombreFichero;

System.out.println("Indique el nombre del fichero:");

while (true) {
    nombreFichero = sc.next().trim();
}
```



- Firmar digitalmente la aplicación.

Para crear la firma digital para esta aplicación abrimos el CMD en la carpeta donde se encuentra ubicado nuestro archivo .jar y escribimos el siguiente comando:

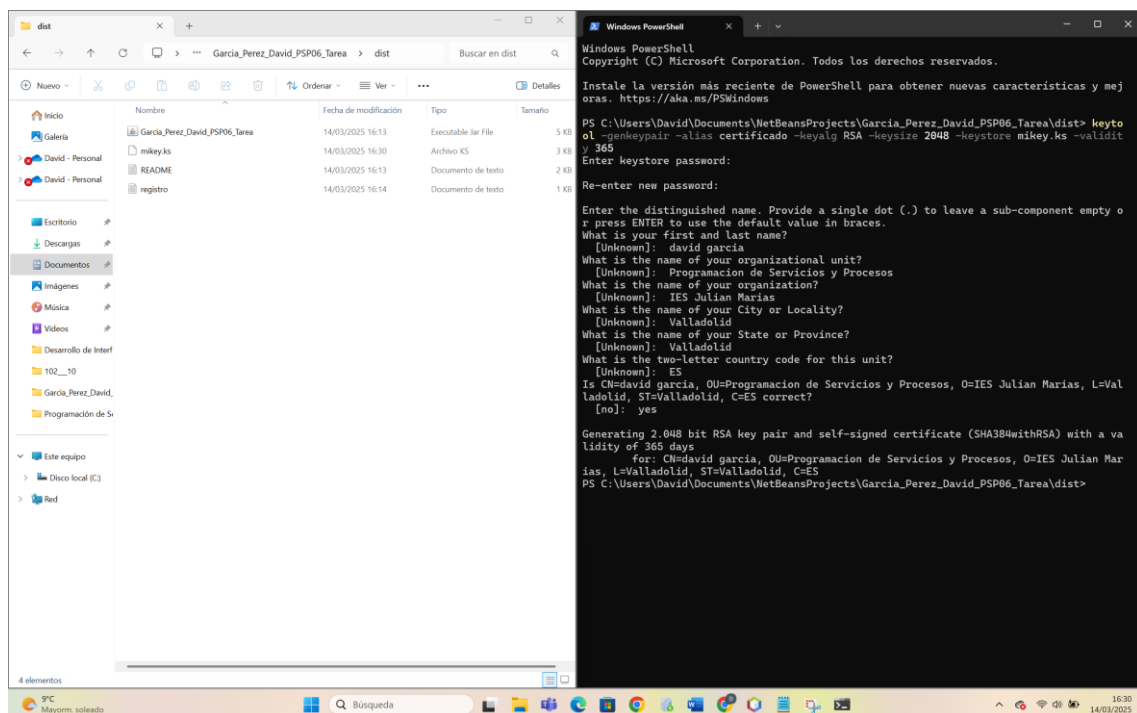
```
keytool -genkeypair -alias certificado -keyalg RSA -keysize 2048 -keystore mikey.ks -validity 365
```

Donde:

- **keytool**: Es una herramienta de línea de comandos de Java utilizada para gestionar claves y certificados.
- **-genkeypair**: Indica que se generará un **par de claves** (pública y privada).
- **-alias certificado**: Define un **alias** para identificar el par de claves dentro del almacén (certificado en este caso).
- **-keyalg RSA**: Especifica el algoritmo de cifrado a utilizar, en este caso **RSA**.
- **-keysize 2048**: Especifica el tamaño de la clave en bits. Un tamaño de **2048 bits** es un estándar seguro para RSA.
- **-keystore mikey.ks**: Define el nombre del archivo donde se almacenará el par de claves (el almacén de claves). En este caso, el archivo se llamará **mikey.ks**.
- **-validity 365**: Indica que el certificado generado tendrá una validez de **365 días**.

A continuación, añadimos la contraseña, en este caso se ha seleccionado una sencilla: 123456, y rellenamos los campos que nos pide.

Tras esto confirmamos los datos introducidos con yes y se nos generará el archivo mikey.ks:



A continuación procedemos a firmar el archivo `Garcia_Perez_David_PSP06_Tarea.jar` con el certificado almacenado en el keystore `mikey.ks` a través del comando:

`jarsigner -keystore mikey.ks Garcia_Perez_David_PSP06_Tarea.jar certificado`

Donde:

- ***jarsigner***: Es una herramienta de Java que se usa para firmar digitalmente archivos .jar y verificar firmas.
- ***-keystore mikey.ks***: Especifica el almacén de claves (keystore) que contiene el certificado usado para la firma.
- ***Garcia_Perez_David_PSP06_Tarea.jar***: El archivo JAR que se va a firmar.
- ***certificado***: Es el alias del certificado dentro del almacén de claves (keystore). Este alias se usará para firmar el JAR.

```
PS C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP06_Tarea\dist> jarsigner -keystore mikey.ks Garcia_Perez_David_PSP06_Tarea.jar certificado
Enter Passphrase for keystore:

jar signed.

Warning:
The signer's certificate is self-signed.
POSIX file permission and/or symlink attributes detected. These attributes are ignored when signing and are not protected by the signature.
PS C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP06_Tarea\dist>
```

Para verificar que todo está correcto ejecutamos el siguiente comando:

`jarsigner -verify -verbose -certs Garcia_Perez_David_PSP06_Tarea.jar`

Este comando verifica la firma digital de un archivo JAR y proporciona detalles sobre la verificación. Aunque nos proporciona el error de que está autofirmado, podemos determinar que la firma es válida aunque no confiable:

```
Windows PowerShell
PS C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP06_Tarea\dist> jarsigner -verify -verbose -certs Garcia_Perez_David_PSP06_Tarea.jar

s 367 Fri Mar 14 16:32:58 CET 2025 META-INF/MANIFEST.MF

>>> Signer
X.509, CN=david garcia, OU=Programacion de Servicios y Procesos, O=IES Julian Marías, L=Valladolid, ST=Valladolid, C=ES
Signature algorithm: SHA384withRSA, 2048-bit key
[Certificate is valid from 14/3/25, 16:38 to 14/3/26, 16:38]
[Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]

t)

407 Fri Mar 14 16:32:58 CET 2025 META-INF/CERTIFIC.SF
1690 Fri Mar 14 16:32:58 CET 2025 META-INF/CERTIFIC.RSA
0 Fri Mar 14 16:13:50 CET 2025 META-INF/
0 Fri Mar 14 16:13:50 CET 2025 garcia_perez_david_psp06_tarea/
4131 Fri Mar 14 16:13:50 CET 2025 garcia_perez_david_psp06_tarea/Main.class

>>> Signer
X.509, CN=david garcia, OU=Programacion de Servicios y Procesos, O=IES Julian Marías, L=Valladolid, ST=Valladolid, C=ES
Signature algorithm: SHA384withRSA, 2048-bit key
[Certificate is valid from 14/3/25, 16:38 to 14/3/26, 16:38]
[Invalid certificate chain: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target]

t)

s = signature was verified
m = entry is listed in manifest
k = at least one certificate was found in keystore

- Signed by "CN=david garcia, OU=Programacion de Servicios y Procesos, O=IES Julian Marías, L=Valladolid, ST=Valladolid, C=ES"
Digest algorithm: SHA-384
Signature algorithm: SHA384withRSA, 2048-bit key

jar verified.

Warning:
This jar contains entries whose certificate chain is invalid. Reason: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid c
ertification path to requested target
This jar contains entries whose signer certificate is self-signed.
This jar contains signatures that do not include a timestamp. Without a timestamp, users may not be able to validate this jar after any of the signer certificates expire (as early
as 2026-03-14).
POSIX file permission and/or symlink attributes detected. These attributes are ignored when signing and are not protected by the signature.
The signer certificate will expire on 2026-03-14.
PS C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP06_Tarea\dist>
```

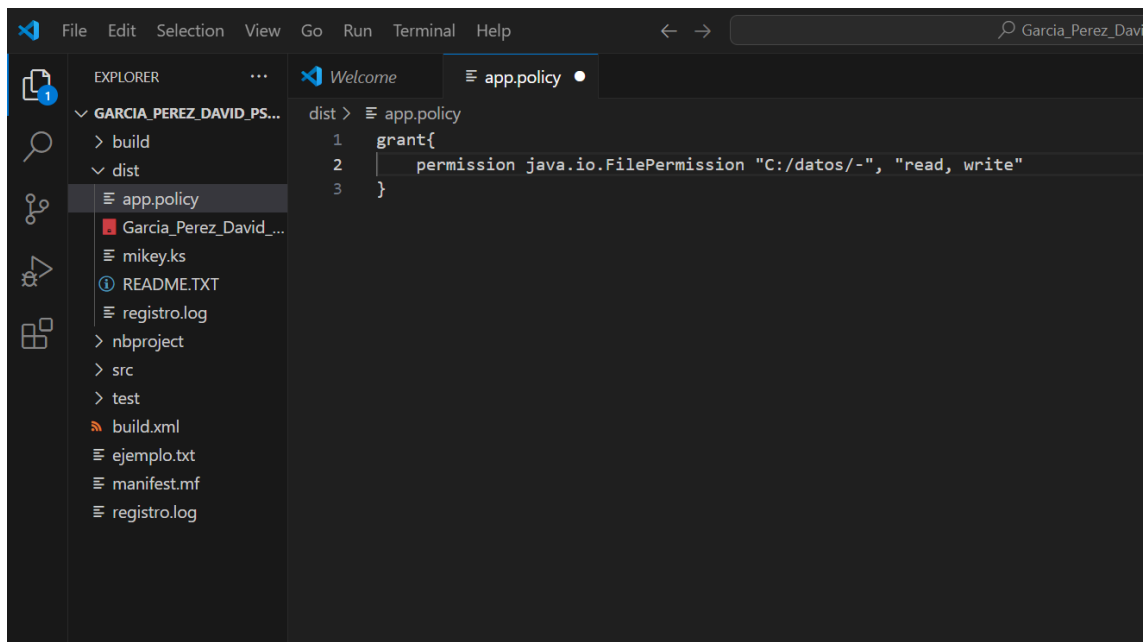
- **Que sólo pueda leer los datos del directorio c:/datos.**

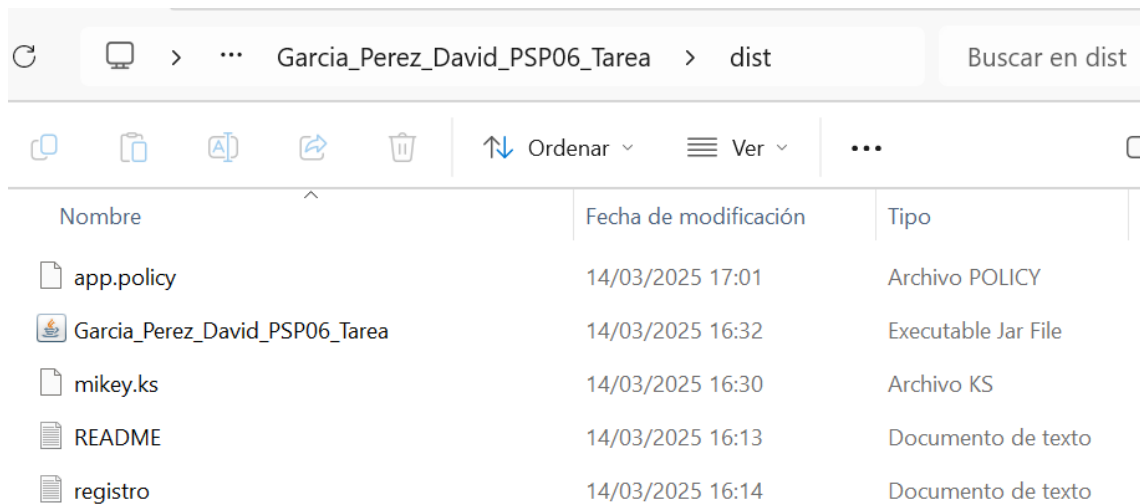
Para que este programa solo pueda leer los datos del directorio c:/datos crearemos un archivo de políticas de seguridad: app.policy. Cuya misión es otorgar permisos a las aplicaciones Java que se ejecutan en un entorno restringido, como un SecurityManager. En este archivo incluimos el siguiente código:

```
grant {  
  
    permission java.io.FilePermission "C:/datos/-", "read, write";  
  
};
```

Donde:

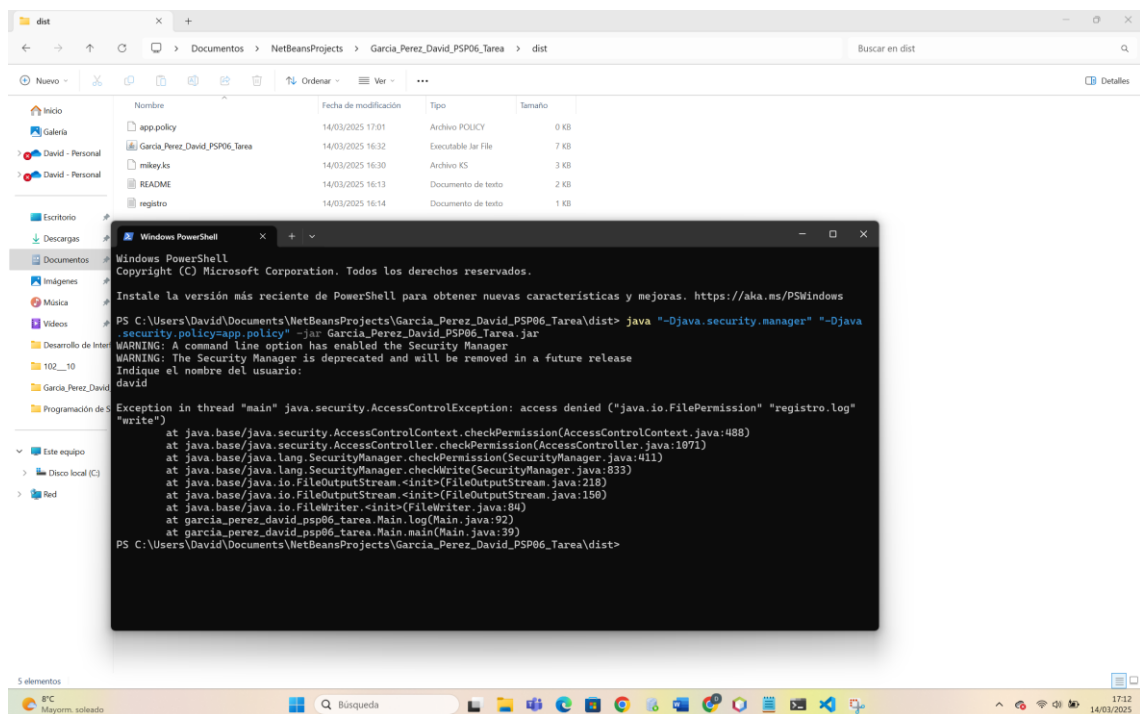
- **grant { ... }:** Define un bloque de permisos que Java aplicará a los programas que usen este archivo de políticas.
- **permission java.io.FilePermission:** Especifica que se está concediendo un permiso relacionado con el acceso a archivos.
- **"C:/datos/-":** El "C:/datos/-" indica que el permiso se aplica a **todos los archivos y subdirectorios** dentro de C:/datos/. El - al final significa que se incluye recursivamente cualquier archivo o directorio dentro de C:/datos/.
- **"read, write":** Especifica que el programa tendrá permisos de **lectura y escritura** en esa ubicación.





Nombre	Fecha de modificación	Tipo
app.policy	14/03/2025 17:01	Archivo POLICY
Garcia_Perez_David_PSP06_Tarea	14/03/2025 16:32	Executable Jar File
mikey.ks	14/03/2025 16:30	Archivo KS
README	14/03/2025 16:13	Documento de texto
registro	14/03/2025 16:14	Documento de texto

Para confirmar que el archivo de políticas de seguridad funciona, intentamos abri el jar en el entorno de un SecurityManager en esta carpeta y no da el siguiente error:



```
PS C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP06_Tarea\dist> java -Djava.security.manager "-Djava.security.policy=app.policy" -jar Garcia_Perez_David_PSP06_Tarea.jar
WARNING: A command line option has enabled the Security Manager
WARNING: The Security Manager is deprecated and will be removed in a future release
Indique el nombre del usuario:
david
Exception in thread "main" java.security.AccessControlException: access denied ("java.io.FilePermission" "registro.log" "write")
    at java.base/java.security.AccessControlContext.checkPermission(AccessControlContext.java:488)
    at java.base/java.security.AccessController.checkPermission(AccessController.java:1971)
    at java.base/java.lang.SecurityManager.checkPermission(SecurityManager.java:411)
    at java.base/java.lang.SecurityManager.checkWrite(SecurityManager.java:833)
    at java.base/java.io.FileOutputStream.<init>(FileOutputStream.java:218)
    at java.base/java.io.FileOutputStream.<init>(FileOutputStream.java:150)
    at java.base/java.io.PrintWriter.<init>(PrintWriter.java:84)
    at Garcia_Perez_David_PSP06_Tarea.Main.log(Main.java:92)
    at Garcia_Perez_David_PSP06_Tarea.Main.main(Main.java:39)
PS C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP06_Tarea\dist>
```


Por otro lado, creamos la carpeta C:/datos e incluimos el archivo jar, el archivo .txt y el archivo .policy. Como vemos en este caso sí se ejecuta la aplicación ya que estamos dentro de la ruta indicada en el archivo.policy:

