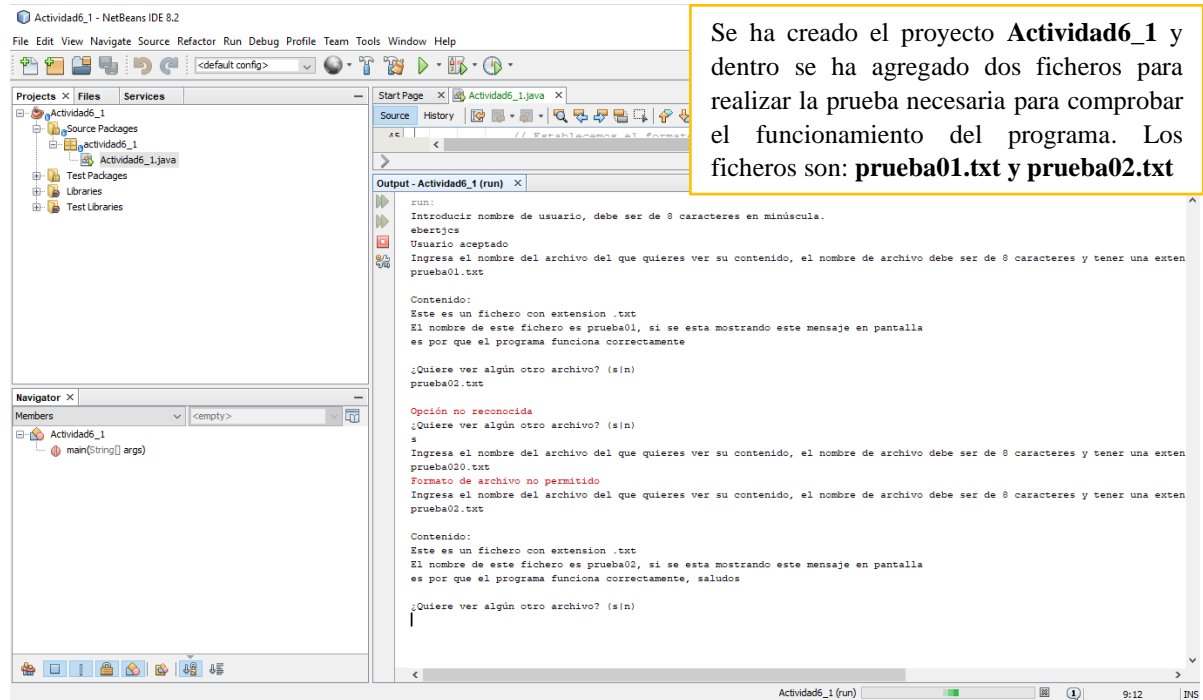


TAREA PARA PSP06

Actividad6_1:

Para desarrollar esta parte de la tarea se ha utilizado Netbeans8.2 y JDK 1.8



Código Actividad6_1.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 *
 * @author Ebert
 */
public class Actividad6_1 {

    public static void main(String[] args) {
        String user, nomArchivo, lineasArchivo;
        Pattern pat = null;
        Matcher mat = null;
        int estado = 0;
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        Logger logger = Logger.getLogger("MyLog");

        try {
            FileHandler fh = new FileHandler("registro.log", true);
            // Creamos el manejador de archivo para el registro
            logger.addHandler(fh);
            // Asociamos el manejador al logger
            logger.setUseParentHandlers(false);
            // Deshabilitamos los manejadores padres para evitar la salida a consola
            logger.setLevel(Level.ALL);
            // Establecemos el nivel de registro para registrar todos los eventos
            SimpleFormatter formatter = new SimpleFormatter();
            // Creamos un formateador simple para el registro
```

```

fh.setFormatter(formatter);
// Establecemos el formateador para el manejador de archivo

do {
    System.out.println("Introducir nombre de usuario, debe ser de 8 caracteres en
minúscula.");
    user = reader.readLine();
    logger.log(Level.INFO, "Usuario introducido: " + user);

    pat = Pattern.compile("[a-z]{8}$");
    // Patrón para verificar que el nombre de usuario tenga 8 caracteres en minúscula
    mat = pat.matcher(user);

    if (mat.matches()) {
        estado = 1;
        System.out.println("Usuario aceptado");
        do {
            System.out.println("Ingresa el nombre del archivo del que quieres ver su
contenido, "
                               + "el nombre de archivo debe ser de 8 caracteres y tener una
extensión "
                               + "de 3 caracteres (Puedes probar con prueba01.txt y
prueba02.txt):");

            nomArchivo = reader.readLine();

            logger.log(Level.INFO, "Nombre de archivo introducido: " + nomArchivo);
            // Registramos el nombre de archivo introducido

            pat = Pattern.compile("[a-zA-Z0-9]{8}\\.[a-z]{3}$");
            // Patrón para verificar el formato del nombre de archivo
            mat = pat.matcher(nomArchivo);

            if (mat.matches()) {
                estado = 2;
                try {
                    FileReader f = new FileReader(nomArchivo);
                    BufferedReader b = new BufferedReader(f);
                    logger.log(Level.INFO, "Se muestra el archivo: " + nomArchivo);

                    System.out.println("\nContenido:");

                    while ((lineasArchivo = b.readLine()) != null) {

                        System.out.println(lineasArchivo);
                    }
                    b.close();

                    do {
                        System.out.println("\n¿Quiere ver algún otro archivo?

(s|n)");

                        String eleccion = reader.readLine();
                        if (eleccion.equalsIgnoreCase("s")) {
                            logger.log(Level.INFO, "Se visualiza otro archivo");
                            estado = 1;
                        } else if (eleccion.equalsIgnoreCase("n")) {
                            logger.log(Level.INFO, "Sesión cerrada");
                            estado = 2;
                        } else {
                            logger.log(Level.WARNING, "Opción no reconocida");
                            System.err.println("Opción no reconocida");
                            estado = 3;
                        }
                    } while (estado == 3);

                    } catch (FileNotFoundException e) {
                        System.err.println("Archivo solicitado no existe");
                        logger.log(Level.WARNING, "Archivo solicitado no existe");
                        estado = 1;
                    }
                } else {
                    logger.log(Level.WARNING, "Formato de archivo incorrecto: " +
nomArchivo);

                    System.err.println("Formato de archivo no permitido");
                }
            } while (estado == 1);
        } else {
            logger.log(Level.WARNING, "Formato de nombre incorrecto: " + user);
            System.err.println("Formato de nombre incorrecto");
        }
    } while (estado == 0);
} catch (IOException e) {
    System.err.println(e.toString());
}
}
}

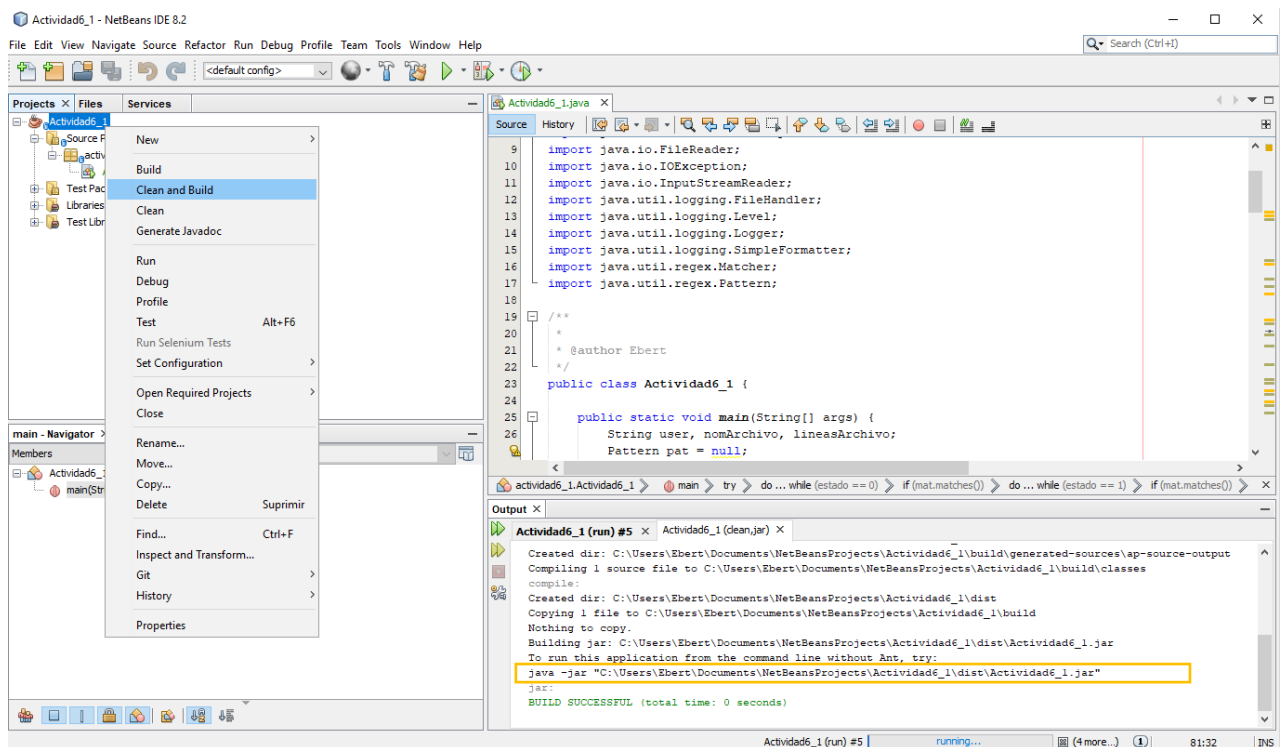
```

Actividad6_2

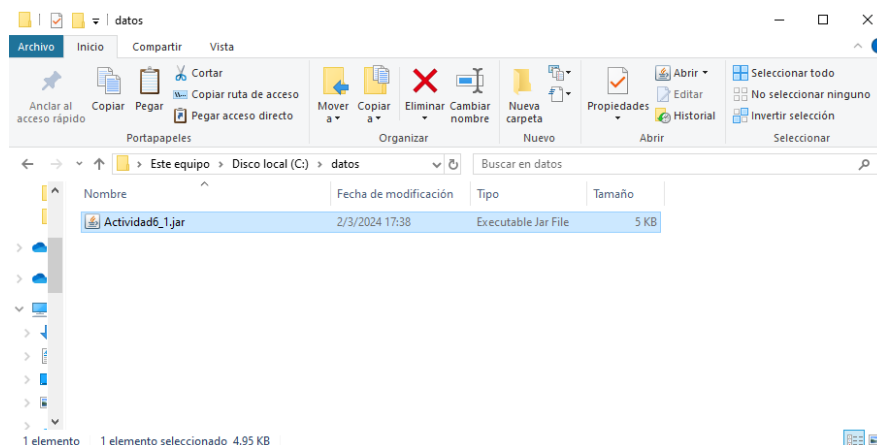
- **Firmar digitalmente la aplicación.**

Para realizar esta actividad se han realizado una serie de procesos, a continuación, intentare explicarlo paso a paso.

- En primer lugar, he realizado un **Clean and Build** en el proyecto **Actividad6_1**, esto ha generado un fichero **.jar** de nombre **Actividad6_1.jar** en el directorio **dist** que se encuentra dentro de nuestro proyecto

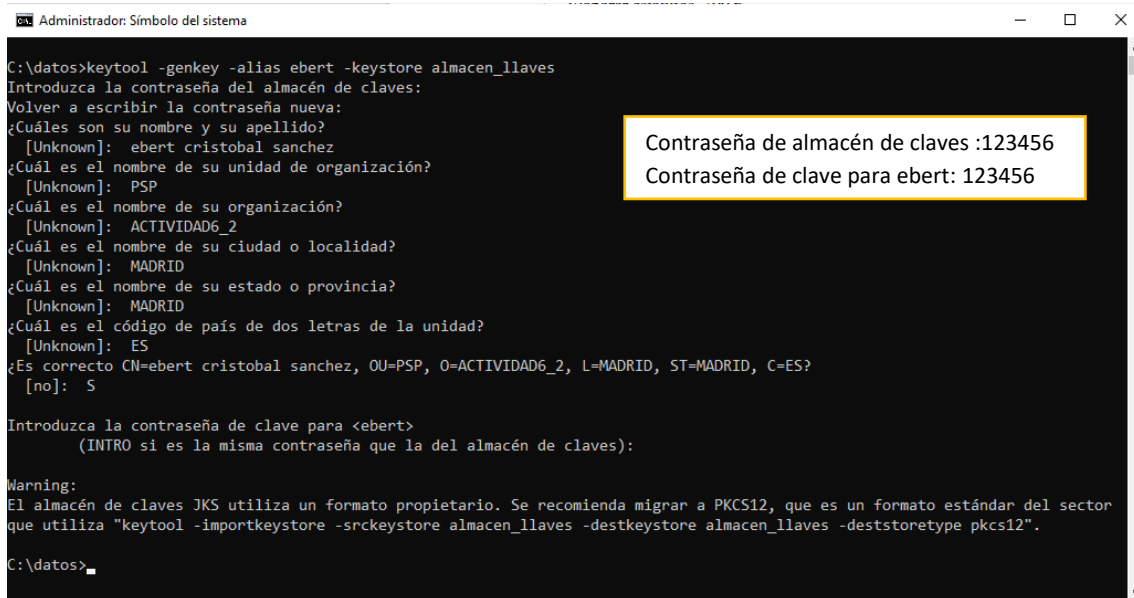


- Posteriormente he creado el directorio **datos** en la **unidad C** y he copiado el fichero **Actividad6_1.jar** que se generó en el directorio **dist**. Recomiendo copiar el fichero y no generar el archivo **Actividad6_1.jar** desde la ruta **c:\datos** porque el fichero no se genera correctamente y en mi caso me estuvo dando un error porque que en **META-INF/MANIFEST.MF** no había generado correctamente el **Main-Class**



- Ahora voy a generar las claves, se va a crear un par de claves (una clave privada y una clave pública) para firmar el fichero, para ello utilizare el siguiente comando:

```
keytool -genkey -alias ebert -keystore almacen_llaves
```



```
C:\datos>keytool -genkey -alias ebert -keystore almacen_llaves
Introduzca la contraseña del almacén de claves:
Volver a escribir la contraseña nueva:
¿Cuáles son su nombre y su apellido?
[Unknown]: ebert cristobal sánchez
¿Cuál es el nombre de su unidad de organización?
[Unknown]: PSP
¿Cuál es el nombre de su organización?
[Unknown]: ACTIVIDAD6_2
¿Cuál es el nombre de su ciudad o localidad?
[Unknown]: MADRID
¿Cuál es el nombre de su estado o provincia?
[Unknown]: MADRID
¿Cuál es el código de país de dos letras de la unidad?
[Unknown]: ES
¿Es correcto CN=ebert cristobal sánchez, OU=PSP, O=ACTIVIDAD6_2, L=MADRID, ST=MADRID, C=ES?
[no]: S
Introduzca la contraseña de clave para <ebert>
(INTRO si es la misma contraseña que la del almacén de claves):

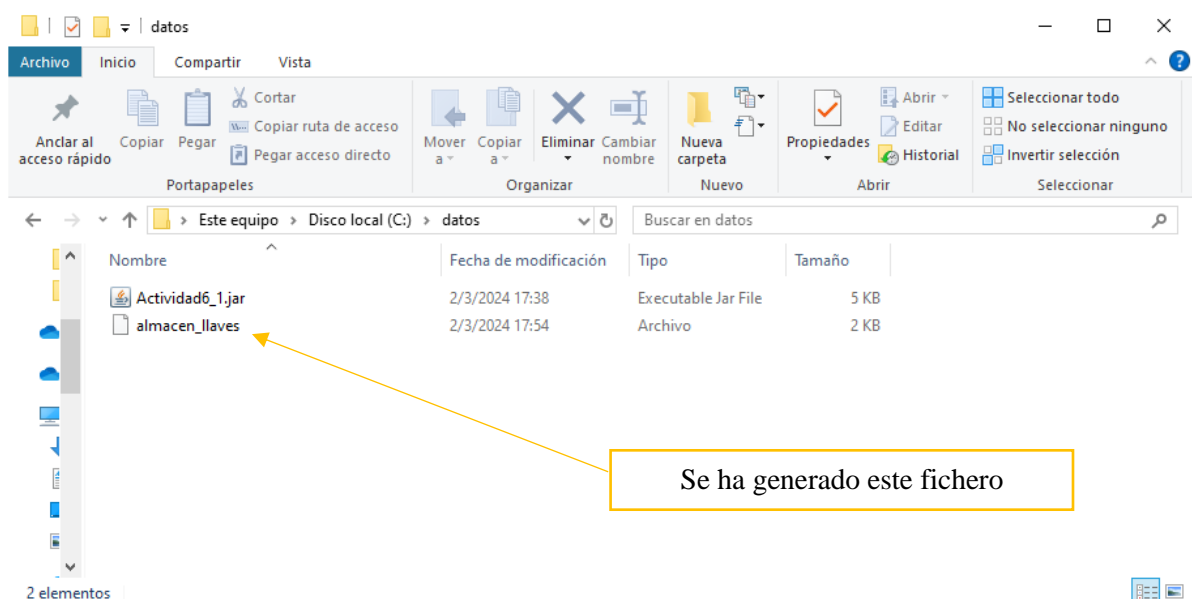
Warning:
El almacén de claves JKS utiliza un formato propietario. Se recomienda migrar a PKCS12, que es un formato estándar del sector
que utiliza "keytool -importkeystore -srckeystore almacen_llaves -destkeystore almacen_llaves -deststoretype pkcs12".

C:\datos>
```

donde:

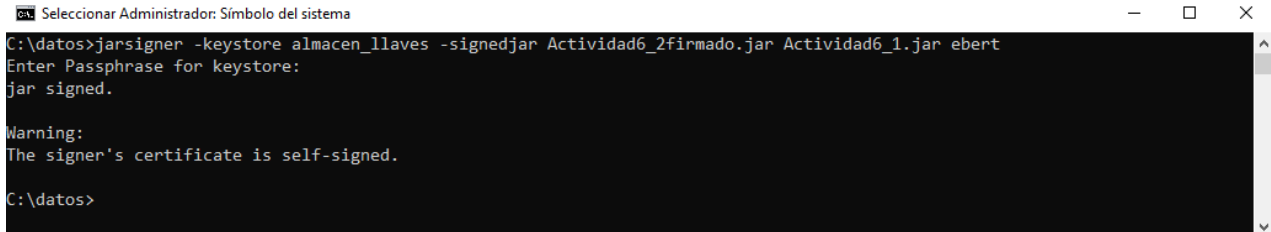
-alias ebert: Indica el alias que se va a utilizar para referirnos al keystore, que es donde se van almacenar las llaves generadas.

-keystore almacen_llaves: Indica el nombre del keystore que se está creando o utilizando



- Ahora se va a firmar el fichero **actividad6_1.jar** utilizando el certificado creado en el paso anterior. Para ello voy a utilizar el siguiente comando:

```
jarsigner -keystore almacen_llaves -signedjar  
Actividad6_2firmado.jar Actividad6_1.jar ebert
```

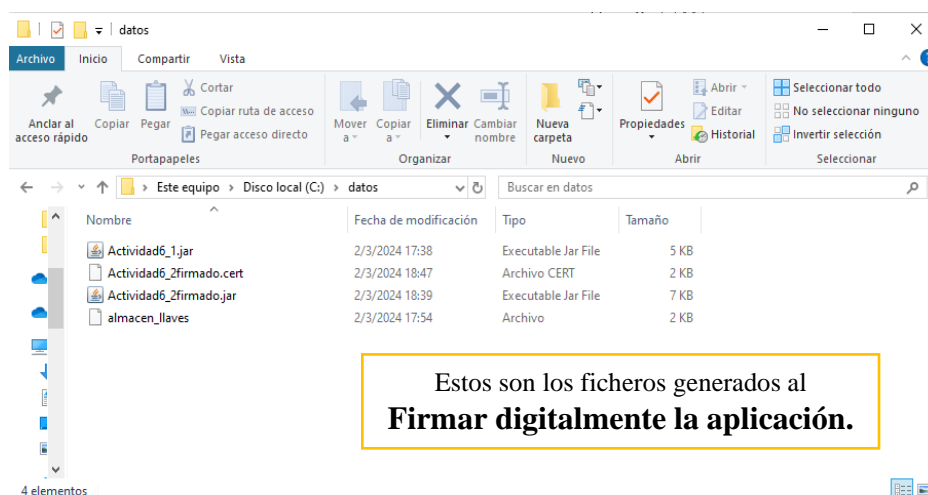
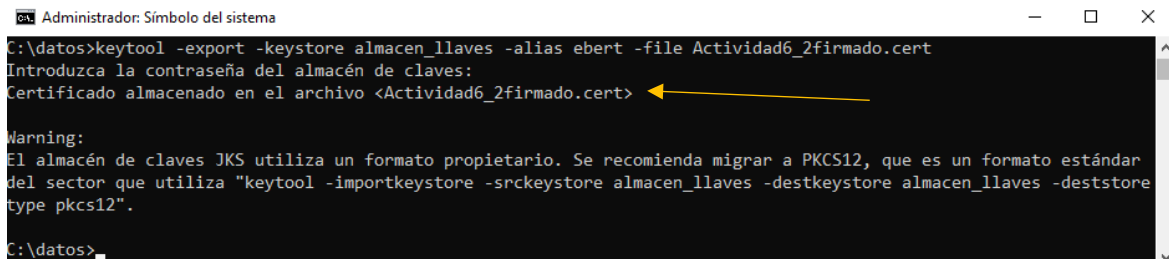


donde:

- keystore almacen_llaves:** Es el keystore creado anteriormente.
- signedjar Actividad6_2firmado.jar:** Indica el fichero jar firmado.
- Actividad6_1.jar:** Es el fichero jar a firmar.
- ebert:** Es el alias que se ha creado anteriormente

- Por último, Exportamos la llave pública del certificado ejecutando el siguiente comando:

```
keytool -export -keystore almacen_llaves -alias ebert -file  
Actividad6_2firmado.cert
```

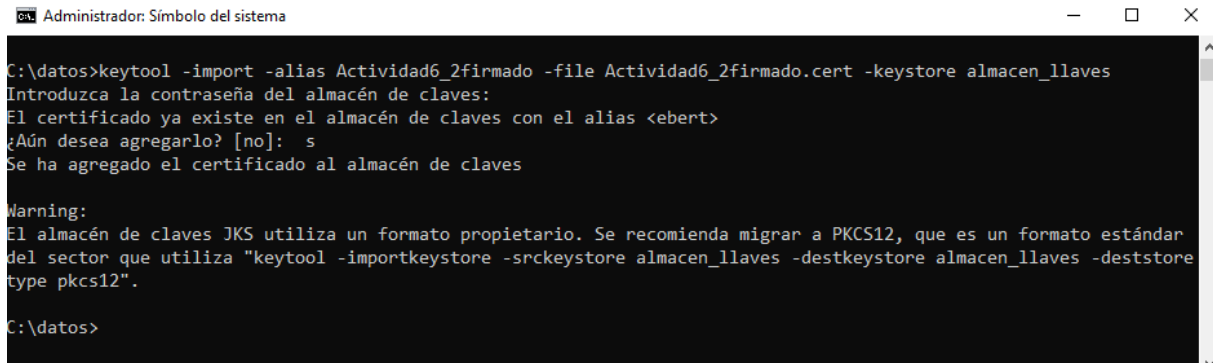


- Que sólo pueda leer los datos del directorio c:/datos.

Para realizar la tarea de este punto y para que la aplicación funcione sin ningún tipo de problemas se van a ejecutar una serie de comandos.

- Primero para que la aplicación funcione correctamente se ejecutara el siguiente comando:

```
keytool -import -alias Actividad6_2firmado -file
Actividad6_2firmado.cert -keystore almacen_llaves
```



```

C:\datos>keytool -import -alias Actividad6_2firmado -file Actividad6_2firmado.cert -keystore almacen_llaves
Introduzca la contraseña del almacén de claves:
El certificado ya existe en el almacén de claves con el alias <ebert>
¿Aún desea agregarlo? [no]: s
Se ha agregado el certificado al almacén de claves

Warning:
El almacén de claves JKS utiliza un formato propietario. Se recomienda migrar a PKCS12, que es un formato estándar
del sector que utiliza "keytool -importkeystore -srckeystore almacen_llaves -destkeystore almacen_llaves -deststore
type pkcs12".

C:\datos>

```

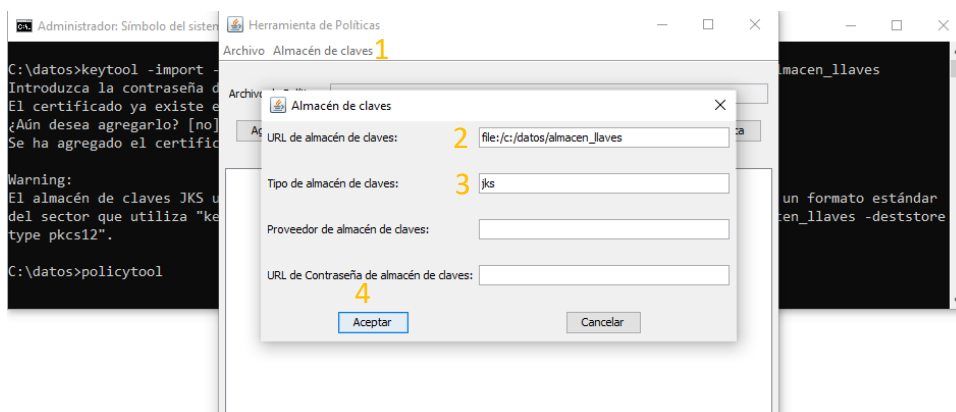
donde:

-import -alias Actividad6_2firmado: Indica que se va a importar el certificado con el alias **Actividad6_2firmado**.

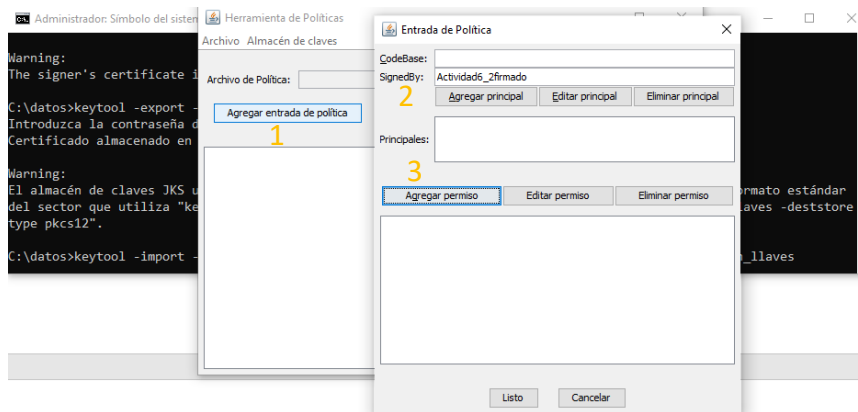
-keystore almacen_llaves: Indica el keystore donde se guarda el certificado.

- Ahora se va a realizar la configuración necesaria para sólo pueda leer los datos del directorio c:/datos y se va a ejecutar el comando: policytool

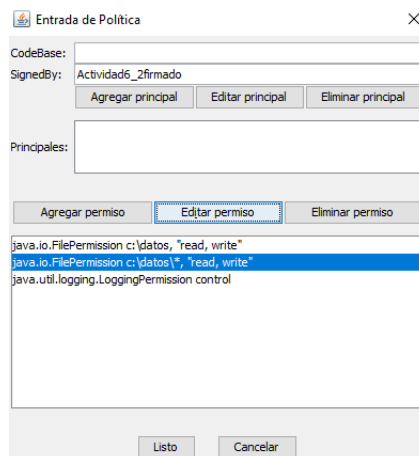
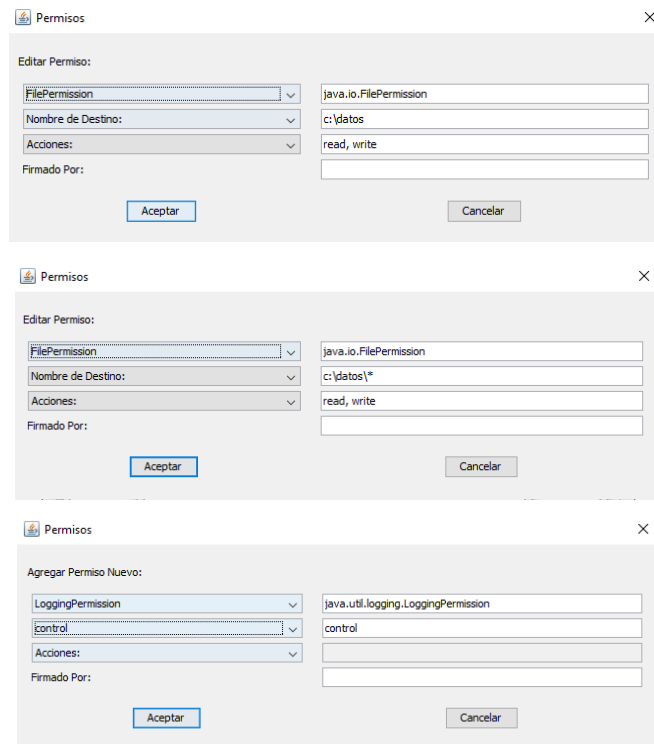
Configurando el almacén de claves



Agregando entrada de políticas

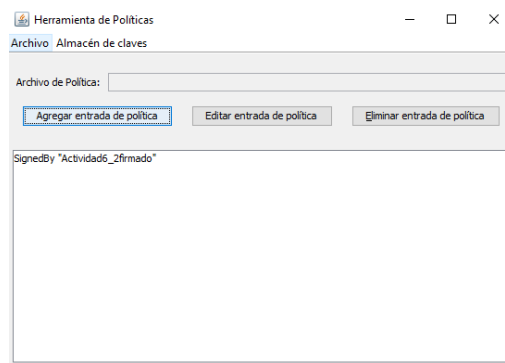


Agregando permisos, en este caso vamos agregar 3 permisos

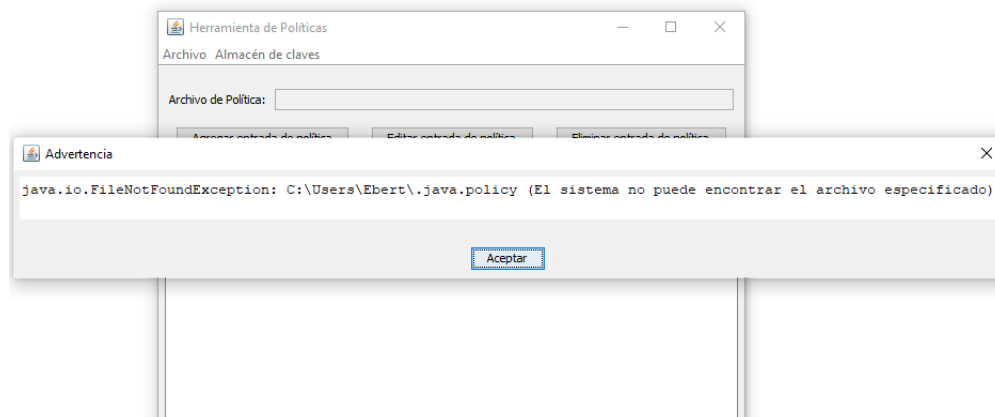


Se han agregado los permisos
c:\datos. Presionamos en el
botón Listo

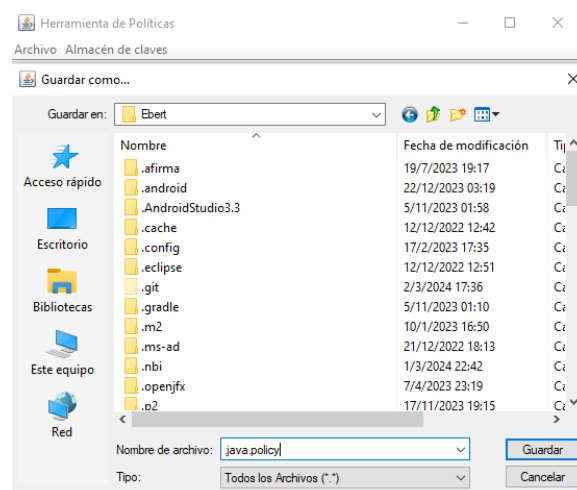
Como podemos ver ya tenemos agregada nuestra entrada de política

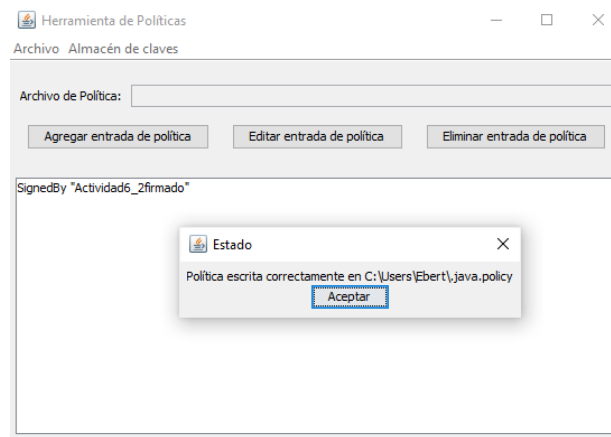


Ahora vamos a presionar en **Archivo** y luego donde dice **Ver log de advertencia** y nos aparece un mensaje que dice que en una determinada ruta no se ha encontrado el archivo especificado.

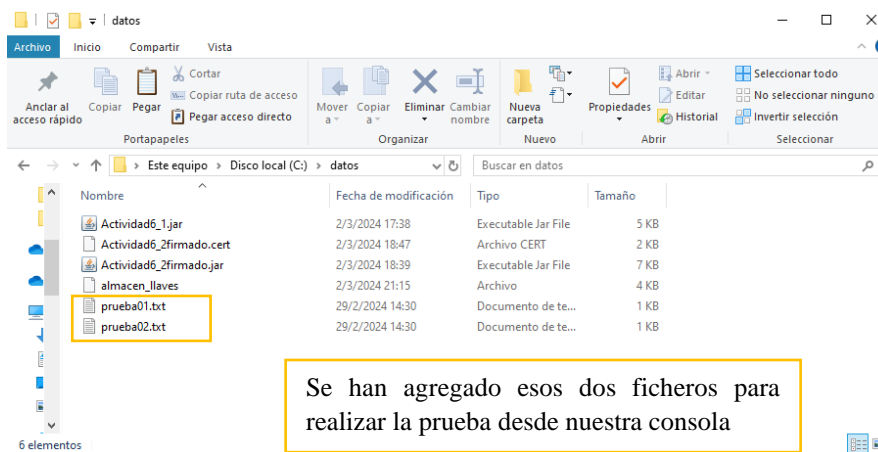


Por último, vamos a presionar nuevamente en **Archivo** y luego en **Guardar como** y buscamos la ruta que nos indica en la advertencia, en mi caso **c:\Users\Ebert** y aquí vamos a guardar nuestro archivo de política que hemos creado, con el nombre **.java.policy**



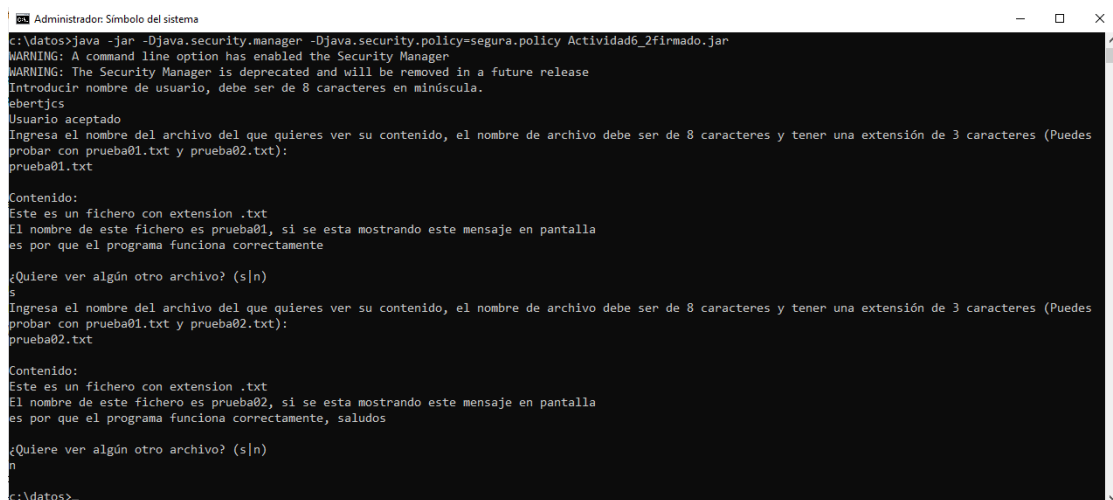


- Para finalizar vamos a comprobar que todo funciona correctamente para el ejercicio planteado en esta tarea



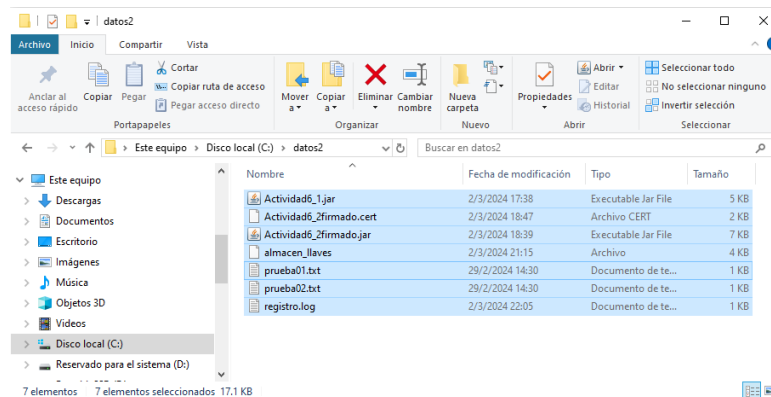
- Ejecutamos por consola desde la ruta **c:\datos** el siguiente comando:

```
java -jar -Djava.security.manager -Djava.security.policy=segura.policy Actividad6_2firmado.jar
```



Como podemos ver en la imagen el ejercicio funciona correctamente.

Para comprobar que solo deba funcionar desde **c:\datos**, he creado otro directorio **c:\datos2** he copiado todos los ficheros generados en **c:\datos** quedando de la siguiente forma



Ahora desde consola vamos a probar a ejecutar:

```
java -jar -Djava.security.manager -Djava.security.policy=segura.policy  
Actividad6_2firmado.jar
```

```
Seleccionar Administrador: Símbolo del sistema
c:\datos2>java -jar -Djava.security.manager -Djava.security.policy=segura.policy Actividad6_2firmado.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
Exception in thread "main" java.security.AccessControlException: access denied ("java.io.FilePermission" "registro.log.lck" "write")
    at java.base/java.security.AccessControlContext.checkPermission(AccessControlContext.java:488)
    at java.base/java.security.AccessController.checkPermission(AccessController.java:1071)
    at java.base/java.lang.SecurityManager.checkPermission(SecurityManager.java:411)
    at java.base/java.lang.SecurityManager.checkWrite(SecurityManager.java:833)
    at java.base/sun.nio.fs.WindowsChannelFactory.open(WindowsChannelFactory.java:302)
    at java.base/sun.nio.fs.WindowsChannelFactory.newFileChannel(WindowsChannelFactory.java:168)
    at java.base/sun.nio.fs.WindowsFileSystemProvider.newFileChannel(WindowsFileSystemProvider.java:114)
    at java.base/java.nio.channels.FileChannel.open(FileChannel.java:309)
    at java.base/java.nio.channels.FileChannel.open(FileChannel.java:369)
    at java.logging/java.util.logging.FileHandler.openFiles(FileHandler.java:512)
    at java.logging/java.util.logging.FileHandler.<init>(FileHandler.java:342)
    at actividad6_1.Actividad6_1.main(Actividad6_1.java:34)
c:\datos2>
```

Como podemos ver en la imagen no nos deja ejecutar la aplicación.