Enunciado.

Ejercicio 1.

De igual manera a lo visto en el tema, ahora te proponemos un ejercicio que genere una cadena de texto y la deje almacenada en un fichero encriptado, en la raíz del proyecto hayas creado, con el nombre fichero.cifrado.

Para encriptar el fichero, utilizarás el algoritmo Rijndael o AES, con las especificaciones de modo y relleno siguientes: Rijndael/ECB/PKCS5Padding.

La clave, la debes generar de la siguiente forma:

A partir de un número aleatorio con semilla la cadena del nombre de usuario + password.

Con una longitud o tamaño 128 bits.

Para probar el funcionamiento, el mismo programa debe acceder al fichero encriptado para desencriptarlo e imprimir su contenido.

1. Generación de la clave a partir de la semilla

Comenzamos creando nuestro proyecto java con el método Main, donde definimos las variables usuario y contraseña, de tipo String. Generaremos la semilla, a partir de la cadena usuario + contraseña:

```
String usuario = "david";
String contraseña = "123456";
String semilla = usuario+ contraseña;
```

Tras esto, generamos la clave secreta usando la semilla a partir del método generarClaveSecreta(semilla) utilizando el algoritmo Rijndael.

```
SecretKey claveSecreta = generarClaveSecreta(semilla);
```

- **Método generarClaveSecreta(String semilla):**

Este método recibe un String como semilla, Usa SecureRandom con la semilla convertida en byte[] para generar valores aleatorios y genera un arreglo de 16 bytes (byte[16]), que es la longitud esperada para claves AES-128. Finalmente devuelve una clave secreta SecretKeySpec(bytes, "Rijndael").

```
public static SecretKey generarClaveSecreta(String semilla){

    SecureRandom random = new SecureRandom(seed: semilla.getBytes());
    byte[] bytes = new byte[16];
    random.nextBytes(bytes);
    return new SecretKeySpec(key:bytes, algorithm: "Rijndael");
}
```

2. Encriptar el texto

A continuación definimos un texto de prueba y lo encriptamos con el método encriptarTexto(texto, claveSecreta).

```
String texto = "Prueba de encriptado de texto";

byte[] textoEncriptado = encriptarTexto(texto, clave: claveSecreta);
```

- **Método encriptarTexto(String texto, SecretKey clave):**

Este método recibe un String con el texto a encriptar y un SecretKey con la clave de cifrado.

Usa Cipher.getInstance("Rijndael/ECB/PKCS5Padding", "BC") para definir el cifrado.

Inicializa el cifrador en **modo encriptación** (ENCRYPT_MODE).

Convierte el texto en byte[] y lo encripta con doFinal().

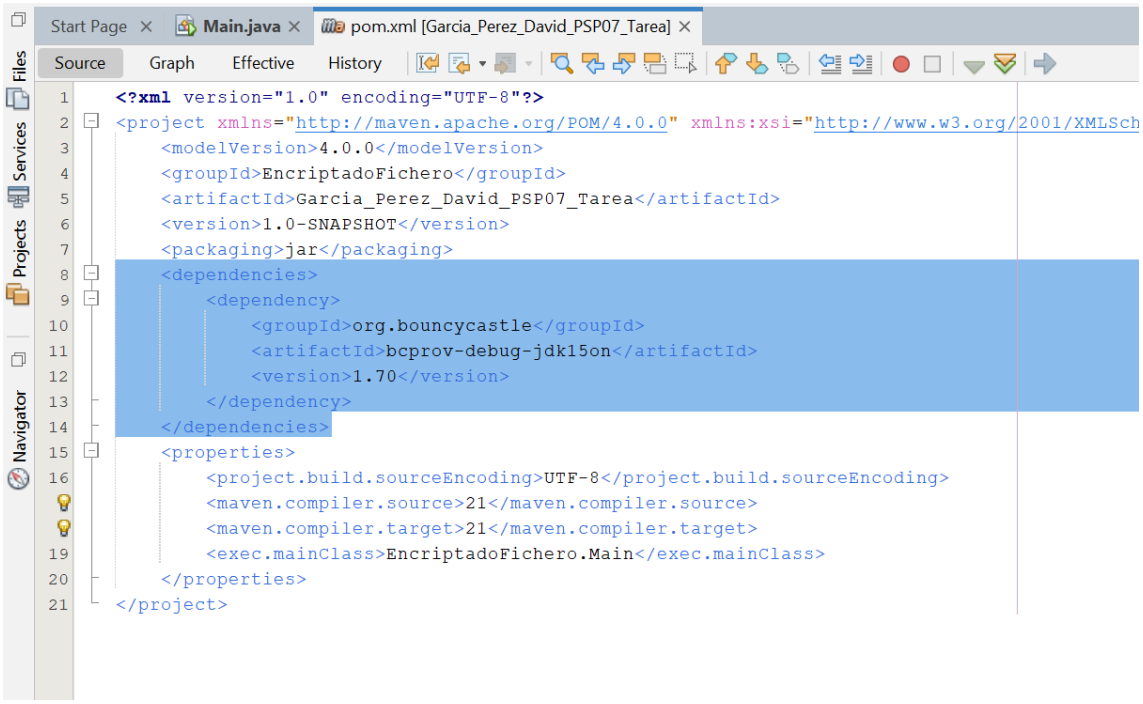
Finalmente devuelve el texto cifrado como byte[].

```
public static byte[] encriptarTexto(String texto, SecretKey clave) throws NoSuchAlgorithmException, NoSuchProviderException, NoSuchPaddingException,
        InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

    Cipher cipher = Cipher.getInstance(transformation: "Rijndael/ECB/PKCS5Padding", provider: "BC");
    cipher.init(opmode: Cipher.ENCRYPT_MODE, key:clave);
    return cipher.doFinal(input: texto.getBytes());
}
```

Para que funcione el provider debemos agregar el proveedor de seguridad **Bouncy Castle** y añadir al pom.xml las dependencias correspondientes y limpiar y construir el proyecto:

```
Security.addProvider(new BouncyCastleProvider());
```



3. Crear el fichero cifrado

Tras esto, procedemos a crear el fichero.cifrado donde guardaremos el texto encriptado, para ello utilizaremos el método escribirFichero(fichero, textoEncriptado):

```
//Generar el fichero con el texto encriptado
String fichero = "fichero.cifrado";
escribirFichero(fichero, bytes: textoEncriptado);
System.out.println("Texto encriptado almacenado en " + fichero + ".");
```

- **Método escribirFichero(fichero, byte[] bytes):**

Este método recibe un String con el nombre del archivo y un byte[] con los datos a escribir.

Tras esto usa FileOutputStream para escribir los bytes en el archivo utilizando un try-with-resources para cerrar el flujo automáticamente.

```
public static void escribirFichero(String fichero, byte[] bytes) throws FileNotFoundException, IOException {

    try (FileOutputStream fos = new FileOutputStream(name: fichero)) {
        fos.write(b: bytes);
    }
}
```

4. Desencriptar y mostrar el texto del fichero encriptado

Finalmente para poder desencriptar y leer el archivo encriptado procedemos a obtener la cadena de bytes que compone el texto encriptado mediante el método leerFichero(fichero), para posteriormente desencriptarlo con el método desencriptarTexto(bytesFichero, claveSecreta) y mostrarlo por consola:

```
//Desencriptar texto
byte[] bytesFichero = leerFichero(fichero);
String textoDesencriptado = desencriptarTexto(bytes: bytesFichero, clave: claveSecreta);
System.out.println("El texto desencriptado es: \n" + textoDesencriptado);
```

- **Método leerFichero(String fichero):**

Este método recibe un String con el nombre del archivo a leer, y usa FileInputStream para leer todos los bytes del archivo y devolver el contenido en un byte[].

```
public static byte[] leerFichero(String fichero) throws FileNotFoundException, IOException {

    try (FileInputStream fis = new FileInputStream(name: fichero)) {
        return fis.readAllBytes();
    }
}
```

- **Método desencriptarTexto(byte[] bytes, SecretKey clave):**

Este método recibe un byte[] con el texto cifrado y un SecretKey con la clave de desencriptación. Después Configura un Cipher en modo DECRYPT_MODE y usa doFinal(bytes) para desencriptar convirtiendo los bytes desencriptados en un String y devolviendo este.

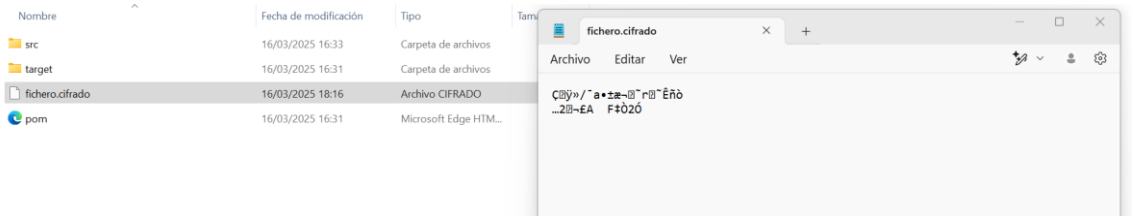
```
public static String desencriptarTexto(byte[] bytes, SecretKey clave) throws NoSuchAlgorithmException, NoSuchProviderException,
        NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

    Cipher cipher = Cipher.getInstance(transformation: "Rijndael/ECB/PKCS5Padding", provider: "BC");
    cipher.init(opmode: Cipher.DECRYPT_MODE, key:clave);
    return new String(bytes: cipher.doFinal(input: bytes));

}
```

5. Prueba de la aplicación

Al ejecutar el programa comprobamos que se nos ha generado en el directorio de este el fichero.encriptado, que si lo abrimos contendrá el texto cifrado en bytes, por lo que será ininteligible:



Pero por consola, este texto se mostrará desencriptado, pudiendo leer correctamente su contenido:

```
--------------------------------[ jar ]---------------------------------

--- resources:3.3.0:resources (default-resources) @ Garcia_Perez_David_PSP07_Tarea ---
skip non existing resourceDirectory C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP07_Tarea\src\main\resources

--- compiler:3.10.1:compile (default-compile) @ Garcia_Perez_David_PSP07_Tarea ---
Changes detected - recompiling the module!
Compiling 1 source file to C:\Users\David\Documents\NetBeansProjects\Garcia_Perez_David_PSP07_Tarea\target\classes

--- exec:3.1.0:exec (default-cli) @ Garcia_Perez_David_PSP07_Tarea ---
Texto encriptado almacenado en fichero.cifrado.
El texto desencriptado es:
Prueba de encriptado de texto
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  1.759 s
Finished at: 2025-03-16T18:16:09+01:00
------------------------------------------------------------------------
```