PSP\Utilidades\src\utilidades\Utilidades.java

```java
package utilidades;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.StandardOpenOption;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.security.GeneralSecurityException;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import java.util.List;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import org.mindrot.jbcrypt.BCrypt;

public class Utilidades {

    public static void main(String[] args) throws FileNotFoundException, IOException, NoSuchAlgorithmException,
            CertificateException, KeyStoreException, UnrecoverableKeyException, KeyManagementException {

        /* *************************** MANEJO DE FICHEROS *************************** */
        // ACCESO A  ESCRIBIR
```

```java
Path pathBytes = Paths.get("archivoBytes.txt");
Path pathString = Paths.get("archivo.txt");

Files.write(pathBytes, "Línea a guardar\n".getBytes()); // sobrescribe
Files.write(pathBytes, "Otra línea\n".getBytes(), StandardOpenOption.APPEND); // añade

Files.writeString(pathString, "Texto de prueba\n"); // sobrescribe
Files.writeString(pathString, "Otra línea\n", StandardOpenOption.APPEND); // añade

// ACCESO A LEER
byte[] bytes = Files.readAllBytes(pathBytes);
String contenido = new String(bytes); // transforma bytes a cadena
System.out.println("**** Lectura y transformación de bytes ****");
System.out.println(contenido);

// ACCESO A LEER TODAS LAS LINEAS DEL ARCHIVO Y SACARLO EN FORMA DE LISTA
List<String> lineas = Files.readAllLines(Path.of("archivo.txt"));
System.out.println("**** Lectura en forma de lista ****");
for (String linea : lineas) {
    System.out.println(linea);
}

// LECTURA Y ESCRITURA
byte[] data = Files.readAllBytes(pathString); // Leer como bytes
Files.write(Path.of("copia.txt"), data); // Escribir a copia

// COMPROBACIÓN DE EXISTENCIA Y CREACIÓN DE ARCHIVOS
try {
    if (Files.exists(pathString)) {
        System.out.println("El archivo ya existe.");
    } else {
        Files.createFile(pathString);
        System.out.println("Archivo creado correctamente.");
    }
} catch (IOException ex) {
    System.out.println(ex.getMessage());
}
```

```java
/* ************************** LIBRERIA BCRYPT ************************** */
String texto = "Este texto va a ser encriptado con BCrypt";
String textoHasheado = BCrypt.hashpw(texto, BCrypt.gensalt(12)); // hashea el texto
boolean sonIguales = BCrypt.checkpw(texto, textoHasheado); // comprueba
System.out.println("El texto hasheado es: " + textoHasheado);


/* ************************** SERVIDOR HTTPS ************************** */
// Cargar el almacén de claves (keystore)
KeyStore keyStore = KeyStore.getInstance("JKS");
try (FileInputStream keyFile = new FileInputStream("AlmacenSSL")) {
    keyStore.load(keyFile, "123456".toCharArray());
}

// Inicializar el gestor de claves con el keystore
KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance("SunX509");
keyManagerFactory.init(keyStore, "123456".toCharArray()); // Usa la misma contraseña

// Inicializar el contexto SSL con el gestor de claves
SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(keyManagerFactory.getKeyManagers(), null, null);

// Declara objeto tipo Factory para crear socket SSL servidor
SSLServerSocketFactory factory = sslContext.getServerSocketFactory();

// Crea un socket servidor seguro
SSLServerSocket socketServidorSsl = (SSLServerSocket) factory.createServerSocket(8444);
System.out.println("Servidor SSL escuchando en el puerto " + 8444);

while (true) {
    // Acepta conexiones de clientes
    SSLSocket socketSsl = (SSLSocket) socketServidorSsl.accept();
    System.out.println("Cliente conectado");
    // Crea un nuevo hilo para manejar al cliente con una clase anónima
    Thread hiloCliente = new Thread() {
        @Override
```

```
            public void run() {
                // Aquí va la lógica para manejar al cliente
            }
        };
        hiloCliente.start();
    }
}

/* ********************** CIFRADO Y DESCIFRADO ********************** */
private static final String CLAVE = "1234567890123456";
private static final String ALGORITMO = "AES";

private static byte[] cifrar(String datos) throws Exception {
    Cipher cipher = Cipher.getInstance(ALGORITMO);
    SecretKeySpec keySpec = new SecretKeySpec(CLAVE.getBytes(), ALGORITMO);
    cipher.init(Cipher.ENCRYPT_MODE, keySpec);
    return cipher.doFinal(datos.getBytes());
}

private static String descifrar(byte[] datosCifrados) throws Exception {
    Cipher cipher = Cipher.getInstance(ALGORITMO);
    SecretKeySpec keySpec = new SecretKeySpec(CLAVE.getBytes(), ALGORITMO);
    cipher.init(Cipher.DECRYPT_MODE, keySpec);
    return new String(cipher.doFinal(datosCifrados));
}

private static final int ENCRIPTAR = Cipher.ENCRYPT_MODE;
private static final int DESENCRIPTAR = Cipher.DECRYPT_MODE;

//cifrar o descifrar archivo
public static byte[] procesarDatos(byte[] datos, int modo) {
    try {
        Cipher cipher = Cipher.getInstance(ALGORITMO);
        SecretKeySpec keySpec = new SecretKeySpec(CLAVE.getBytes(), ALGORITMO);
        cipher.init(modo, keySpec);
        return cipher.doFinal(datos);
    } catch (GeneralSecurityException ex) {
```

```
            System.err.println("Error al procesar: " + ex.getMessage());
            return null;
        }
    }

    /* ************** CONSTRUIR Y ENVIAR UNA RESPUESTA HTTP ************** */
    private void enviarRespuesta(PrintWriter writer, int codigo, String contenido) {
        String statusLine;

        switch (codigo) {
            case 200 ->
                statusLine = "HTTP/1.1 200 OK";
            case 400 ->
                statusLine = "HTTP/1.1 400 Bad Request";
            case 404 ->
                statusLine = "HTTP/1.1 404 Not Found";
            default -> {
                statusLine = "HTTP/1.1 500 Internal Server Error";
                contenido = "<h1>Error desconocido</h1>";
            }
        }

        writer.println(statusLine + "\n"
                + "Content-Type: text/html; charset=UTF-8\n"
                + "Content-Length: " + contenido.length() + "\n"
                + "\n"
                + contenido);
    }
}
```