**Apuntes Examen Junio PSP**

* **Acceso a un fichero cifrado para agregar contenido**

**Path path = Paths.get("archivo.txt");**

**byte[] bytes = "Hola mundo".getBytes();**

**if (Files.exists(path) && Files.size(path) > 0) {**

**// entra si el fichero existe y tiene contenido**

**byte[] contenidoCifrado = Files.readAllBytes(path); // lee el archivo cifrado**

**// el contenido del archivo en bytes se lo pasaríamos al cifrar o descifrar**

**}**

* **Acceso a un fichero y leer todas sus lineas**

**private static final String ARCHIVO = "frases.txt";**

**public String obtenerFrase() {**

**Path path = Paths.get(ARCHIVO);**

**if (!Files.exists(path)) {**

**return "";**

**}**

**try {**

**List<String> frases = Files.readAllLines(path);**

**if (frases.isEmpty()) {**

**return "";**

**}**

**int indice = (int)(Math.random() \* frases.size());**

**return frases.get(indice);**

**} catch (IOException e) {**

**System.err.println("Error al leer el archivo: " + e.getMessage());**

**return "";**

**}**

**}**

* **Crear un nuevo fichero o modificar el existente**

**// Si el archivo no existe, lo crea**

**Files.write(path, bytes, StandardOpenOption.CREATE);**

**// Si el archivo ya existe, agrega el nuevo contenido al final sin borrar lo anterior**

**Files.write(path, bytes, StandardOpenOption.APPEND);**

* **Cifrar y descifrar**

**private static final String ALGORITMO = "AES";**

**private static final byte[] CLAVE\_AES = "1234567890123456".getBytes();**

**private static byte[] cifrar(String datos) throws Exception {**

**Cipher cipher = Cipher.getInstance(ALGORITMO);**

**SecretKey clave = new SecretKeySpec(CLAVE\_AES, ALGORITMO);**

**cipher.init(Cipher.ENCRYPT\_MODE, clave);**

**return cipher.doFinal(datos.getBytes());**

**}**

**private static String descifrar(byte[] datosCifrados) throws Exception {**

**Cipher cipher = Cipher.getInstance(ALGORITMO);**

**SecretKey clave = new SecretKeySpec(CLAVE\_AES, ALGORITMO);**

**cipher.init(Cipher.DECRYPT\_MODE, clave);**

**return new String(cipher.doFinal(datosCifrados));**

**}**

* **Almacen concurrente**

**public class Almacen {**

**public final static int LIBRE = 0;**

**public final static int CON\_ESCRITOR = 1;**

**private int estado = LIBRE;**

**public synchronized void accesoEscribir(String dia, int cantidad) {**

**if (estado == LIBRE) {**

**estado = CON\_ESCRITOR;**

**// desencripta, lee, añade información y vuelve a encriptar**

**Cifrador.guardarReservas(dia, cantidad);**

**} else {**

**while (estado != LIBRE) {**

**try {**

**// no está libre, el hilo permanece a la espera**

**wait();**

**} catch (InterruptedException e) {**

**System.out.println(e);**

**}**

**}**

**estado = CON\_ESCRITOR;**

**// desencripta, lee, añade información y vuelve a encriptar**

**Cifrador.guardarReservas(dia, cantidad);**

**}**

**System.out.println("Hilo guarda la información en " + dia + ".txt");**

**}**

**public synchronized void escrituraFinalizada() {**

**estado = LIBRE;**

**notifyAll();**

**}**

**}**

* **Servidor HTTP**

**public class ServidorHTTP {**

**public static void main(String[] args) throws Exception {**

**ServerSocket serverSocket = new ServerSocket(12349);**

**System.out.println("Servidor iniciado en el puerto 12349 …");**

**while (true) {**

**Socket clientSocket = serverSocket.accept();**

**new Thread(new HiloServidor(clientSocket)).start();**

**}**

**}**

**}**

**}**

**class HiloServidor extends Thread {**

**private final Socket cliente;**

**public HiloServidor(Socket cliente) {**

**this.cliente = cliente;**

**}**

**@Override**

**public void run() {**

**// Lógica del hilo**

**}**

**}**

* **Servidor HTTPS**

**try (FileInputStream fis = new FileInputStream("AlmacenSSL")) {**

**//Transformar la contraseña del almacén a array**

**char[] claveKeystore = "123456".toCharArray();**

**KeyStore ks = KeyStore.getInstance("JKS");**

**ks.load(fis, claveKeystore);**

**//Crear la fábrica de claves SSL**

**KeyManagerFactory kmf =**

**KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());**

**kmf.init(ks, claveKeystore);**

**//Configurar SSLContext**

**SSLContext sslContext = SSLContext.getInstance("TLS");**

**sslContext.init(kmf.getKeyManagers(), null, null);**

**//Crear el servidor SSL**

**SSLServerSocketFactory ssf = sslContext.getServerSocketFactory();**

**SSLServerSocket sslServerSocket = (SSLServerSocket) ssf.createServerSocket(PUERTO);**

**//Habilitar protocolos seguros**

**sslServerSocket.setEnabledProtocols(new String[]{"TLSv1.2", "TLSv1.3"});**

**//Mensaje informativo en el servidor dando instrucciones de uso por parte del cliente**

**imprimeDisponible();**

**//Permite al servidor procesar múltiples peticiones de los clientes**

**while (true) {**

**//Método bloqueante que queda a la espera de la conexión de clientes**

**socCliente = sslServerSocket.accept();**

**//Procesa la peticion del cliente, recibe el socket del cliente y el almacen de usuarios**

**Thread hiloServidor**

**= new Thread(new HiloServidor(socCliente, almacen));**

**//Lanzamos el hilo para atender al cliente**

**hiloServidor.start();**

**}**

**} catch (IOException**

**| NoSuchAlgorithmException**

**| CertificateException**

**| KeyStoreException**

**| UnrecoverableKeyException**

**| KeyManagementException ex) {**

**System.out.println(ex.getMessage());**

**}**

**}**

* **Obtener un número random entre un mínimo y un máximo**

**int random = (int) (Math.random() \* (max - min + 1)) + min;**