PROGRAMACIÓN DE
SERVICIOS Y PROCESOS

PROYECTO FINAL

0. Finalidad del programa

1. Conexión
   a) SSL
   b) Proceso de conexión Cliente / Servidor
   c) JAAS

2. Comunicación

3. Métodos
   a) Encriptación / Desencriptación
   b) Otros: HiloServidor
   c) Otros: Cliente

4. Ventana Cliente

5. Logs

Daniel Tamargo

0. Finalidad del programa

El programa se basa en la premisa de recrear un juego online donde un cliente se conectará a un servidor, iniciará sesión con sus credenciales y recibirá las reglas firmadas, una vez validadas podrá comenzar partidas donde el servidor le mandará preguntas y el cliente las contestará. Al final de cada partida el servidor informará de la puntuación final y el cliente podrá optar por jugar de nuevo.

En base a lo aprendido en clase, como mínimo el programa debería contar con lo siguiente:

- Conexión ServerSocket / Socket
- Multicliente (el Servidor delega cada conexión en un hilo)
- Generar par de claves pública / privada
- Encriptar los mensajes con ese par de claves
- Firmar y verificar las normas con ese par de claves
- Contear la puntuación de la partida
- Patrones para comprobar los datos insertados

En mi caso, además, he implementado lo siguiente:

- Conexión SSLServerSocket / SSLSocket (SSL)
- Java Authentication and Authorization Service (JAAS)
- Uso híbrido de claves (asimétrica + simétrica)
- Top puntuaciones
- Contador tiempo + animación puntos extra para el cliente
- Logs en el servidor
- Ventana administrador capaz de ver online los Logs
- El cliente encripta su contraseña para mayor privacidad

En este informe trataré de explicar y mostrar lo más relevante de la aplicación.

1. Conexión

Para dotar a la aplicación de una seguridad mucho mayor, he implementado los siguientes puntos en la aplicación

a) Secure Sockets Layer (SSL)

Para el servidor se genera un certificado y para el cliente, a través de ese certificado, se genera una clave de confianza. En este caso ambos están almacenados de forma local.

Con el certificado y la clave de confianza generados, tendremos que configurar las propiedades tanto en el Servidor como en el Cliente. De esta manera, si de alguna forma el servidor fuera suplantado y el certificado no correspondiera, el cliente lo sabría al instante y no se conectaría, evitando así exponer su información.

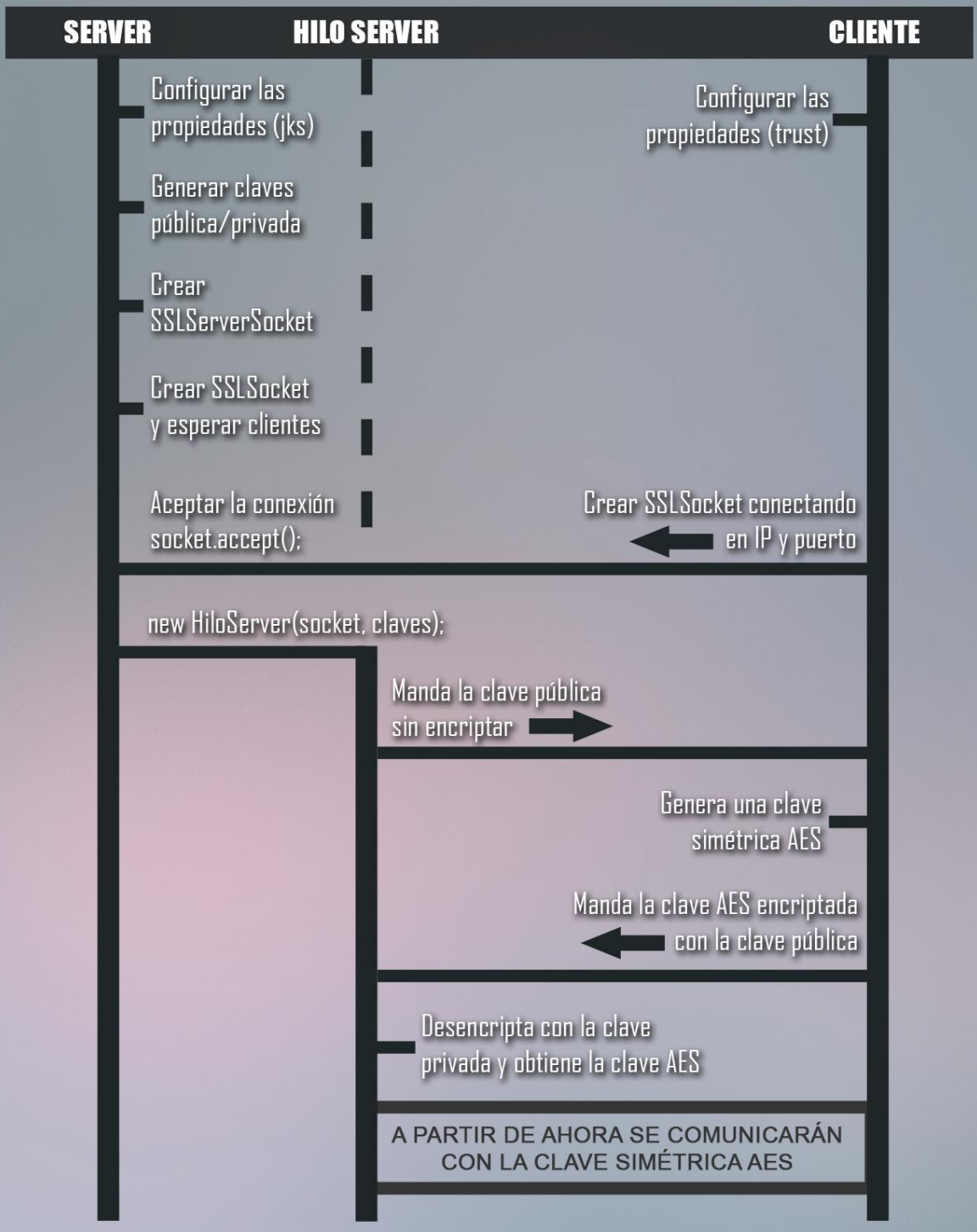
```
// Configuramos las propiedades para seleccionar qué certificado definirá la confianza con el servidor
System.setProperty("javax.net.ssl.keyStore", "./certificados/servidorAlmacenSSL.jks");
System.setProperty("javax.net.ssl.keyStorePassword", "12345Abcde");

SSLServerSocketFactory sfact = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
SSLServerSocket serverSocketSSL = (SSLServerSocket) sfact.createServerSocket( port: 6000);
```

```
// Configuramos las propiedades para que confié en el certificado que hemos ""recibido"" (realmente está en local)
System.setProperty("javax.net.ssl.trustStore", "./certificados/clienteAlmacenSSL");
System.setProperty("javax.net.ssl.trustStorePassword", "890123");

// Nos conectamos
SSLSocketFactory sfact = (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket socketSSL = (SSLSocket) sfact.createSocket( host: "localhost",  port: 6000);
System.out.println(nombre + "Conexión realizada");
```

Cabe señalar que en este proyecto hago el proceso en un sentido, pero lo más óptimo y seguro sería hacerlo en ambos sentidos, y que así el servidor también pueda corroborar la confianza en el cliente.

b) Proceso de conexión Cliente / Servidor

Ya con el protocolo SSL configurado, el cliente se conectará. Una vez conectado, el servidor le mandará su clave pública, y el cliente la usará para encriptar una clave simétrica y enviársela al Servidor. De esta manera ambos tendrán la clave simétrica y podrán comunicarse de manera rápida y segura.



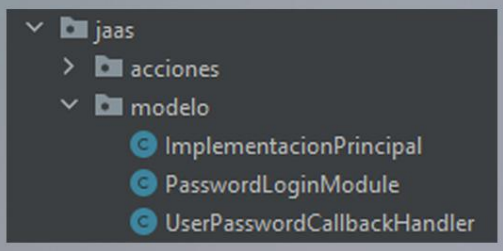
Daniel Tamargo

c) Java Authentication and Authorization Service

He implementado el protocolo JAAS para dotar al proceso de inicio de sesión de una mayor seguridad.

Realmente, JAAS está más enfocado a aplicaciones web o aplicaciones con mayor trasfondo, puesto que su finalidad es generar un contexto a través del cual poder sacar la información del usuario loggeado y comprobar su nivel de privilegios evitando así que pueda realizar acciones sin tener el permiso necesario.

La implementación del JAAS requiere de la creación de las siguientes clases:



En el paquete acciones tendríamos todas las acciones que se ejecutarían en la aplicación, y en cada acción comprobaríamos los privilegios del usuario loggeado para ver si puede ejecutarla o no. En este caso, simplemente sacaré el usuario loggeado del contexto para utilizar su información y/o guardar su puntuación.

Es de vital importancia señalarle a la aplicación cuál es el fichero de configuración que definirá el comportamiento de jaas así como los distintos tipos de usuarios que iniciarán sesión.

En la siguiente página mostraré cómo funciona el protocolo JAAS.

c) Java Authentication and Authorization Service

Cuando el Servidor recibe las credenciales del usuario, este genera un nuevo LoginContext, donde se le dirá el nombre de tipo de usuario que se va a loggear y se le pasará un objeto UserPasswordCallbackHandler con las credenciales.
Ya configurado se lanza el método login().

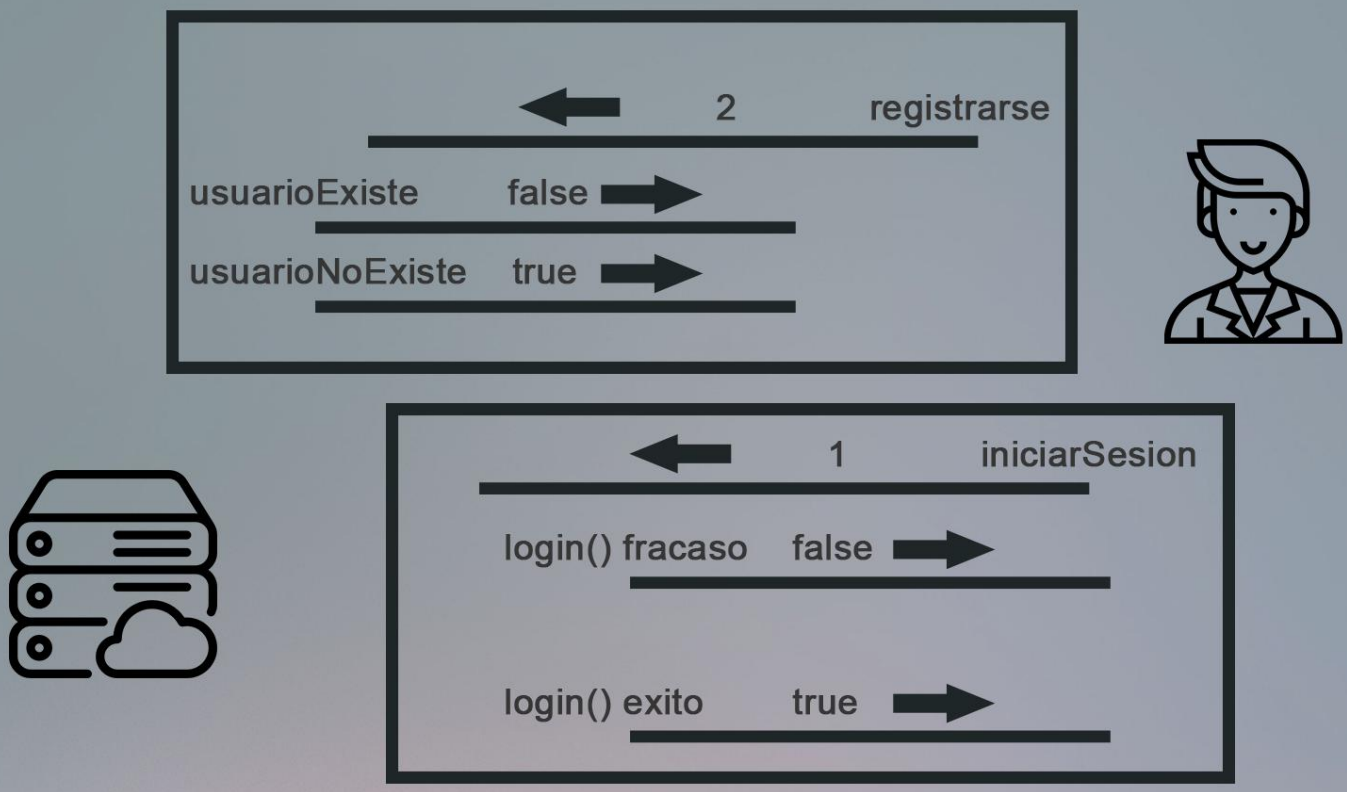
```
loginContext = new LoginContext(
        name: "Jugador",
       new UserPasswordCallbackHandler(nick, password)
);
loginContext.login();
```

En el fichero jaas.config hemos definido que nuestro tipo de usuario “Jugador” utilizará la clase PasswordLoginModule que hemos creado anteriormente. Esta clase implementará el proceso de login() y sus respectivos procesos posteriores:

```
/**
 * Login
 * - comprobamos que el usuario existe
 * - si existe, cotejamos la contraseña y si es correcta, accederá
 * - si no existe o la contraseña no es correcta, no accederá
 */
public boolean login() throws LoginException {...}

/**
 *  Método que se ejecuta si el login tiene éxito
 */
public boolean commit() throws LoginException {...}

/**
 * Método que se ejecuta si el login falla
 */
public boolean abort() throws LoginException {...}

/**
 * Método que ejecutamos cuando el usuario cierra sesión Logout
 */
public boolean logout() throws LoginException {...}

/**
 * Método que limpiar la contraseña (logout, abort)
 */
private void clearPassword() {...}
```

2. Comunicación

A través del siguiente esquema representaré un ejemplo de cómo se comunica la aplicación, es decir, si el cliente selecciona una opción, cómo influye en la comunicación.



Cuando el cliente selecciona la opción registrarse, le enviará un número entero 2 seguido de todos los datos necesarios para el registro. El servidor recibe ese 2 y por tanto sabe que va a recibir dichos datos y a comprobar que el usuario no exista para registrarlo. Le devuelve un true si lo ha registrado, y un false si el usuario ya existía.

De la misma manera, cuando el usuario selecciona iniciar sesión, le envía un 1 seguido del nick y contraseña, el servidor lo comprobará y le devolverá false si el login ha fracasado y true si el login ha tenido éxito, pasando el cliente a la ventana de validdar las normas.

3. Métodos

a) Encriptación / Desencriptación

Algunos de los métodos utilizados para encriptar y desencriptar tipos de datos como Strings, ints, o ArrayLists de Strings

```java
public byte[] encriptarArrayListString(SecretKey claveAES, ArrayList<String> arrayList)
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DataOutputStream out = new DataOutputStream(baos);
    for (String element : arrayList) {
        out.writeUTF(element);
    }

    byte[] bytes = baos.toByteArray();
    Cipher aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.ENCRYPT_MODE, claveAES);
    return aesCipher.doFinal(bytes);
```

```java
public ArrayList<String> desencriptarArrayListString(SecretKey claveAES, byte[] mensaje)
    ArrayList<String> datos = new ArrayList<>();
    Cipher aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.DECRYPT_MODE, claveAES);
    byte[] bytes = aesCipher.doFinal(mensaje);

    ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
    DataInputStream in = new DataInputStream(bais);
    while (in.available() > 0) {
        String element = in.readUTF();
        datos.add(element);
    }

    return datos;
```

```java
public byte[] encriptarMensaje(SecretKey claveAES, String mensaje)
    Cipher aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.ENCRYPT_MODE, claveAES);
    return aesCipher.doFinal(mensaje.getBytes());
```

```java
public String desencriptarMensaje(SecretKey claveAES, byte[] mensaje)
    Cipher aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.DECRYPT_MODE, claveAES);
    return new String(aesCipher.doFinal(mensaje));
```

```java
public byte[] encriptarInt(SecretKey claveAES, int dato)
    Cipher aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.ENCRYPT_MODE, claveAES);
    return aesCipher.doFinal(intToByteArray(dato));
```

```java
public byte[] intToByteArray(int dato) {
    byte[] resultado = new byte[4];
    resultado[0] = (byte) ((dato & 0xFF000000) >> 24);
    resultado[1] = (byte) ((dato & 0x00FF0000) >> 16);
    resultado[2] = (byte) ((dato & 0x0000FF00) >> 8);
    resultado[3] = (byte) ((dato & 0x000000FF));
    return resultado;
```

```java
public int desencriptarInt(SecretKey claveAES, byte[] mensaje) t
    Cipher aesCipher = Cipher.getInstance("AES");
    aesCipher.init(Cipher.DECRYPT_MODE, claveAES);
    return ByteBuffer.wrap(aesCipher.doFinal(mensaje)).getInt();
```

Daniel Tamargo

3. Métodos

b) Otros: HiloServidor

Algunos de los métodos por la clase HiloServidor

```java
/**
 * Método utilizado para cargar y devolver un ArrayList de Strings que
 * contendrá el top 10 de puntuaciones + el top y la puntuación del jugador loggeado
 */
public ArrayList<String> topPuntuaciones() {
    ArrayList<String> lineasTopPuntuaciones = new ArrayList<>();

    ArrayList<Usuario> usuarios = LeerFicheros.leerUsuarios();
    HashMap<Usuario, Integer> listaSinOrdenar = new HashMap<>();

    for (Usuario usuario: usuarios) {
        listaSinOrdenar.put(usuario, usuario.getPuntuacion());
    }

    LinkedHashMap<Usuario, Integer> listaOrdenada = new LinkedHashMap<>();
    listaSinOrdenar.entrySet() Set<Map<K, V>.Entry<Usuario, Integer>>
            .stream() Stream<Map<K, V>.Entry<Usuario, Integer>>
            .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
            .forEachOrdered(x -> listaOrdenada.put(x.getKey(), x.getValue()));

    int n = 0;
    String tuPuntuacion = "";
    for (Usuario value : listaOrdenada.keySet()) {
        if (n < 10) {
            lineasTopPuntuaciones.add(String.format("%-18s%3d\n", value.getNick(), value.getPuntuacion()).replace( oldChar: ' ', newChar: '.'));
        }
        if (value.getNick().equalsIgnoreCase(nickJugador)) {
            tuPuntuacion = "Tu puntuación: " + value.getPuntuacion() + "\n" +
                    "Eres el top " + (n + 1);
        }
        n++;
    }

    for (int i = 0; i < (10 - n); i++) {
        lineasTopPuntuaciones.add("\n");
    }
    for (int i = 0; i < 3; i++) {
        lineasTopPuntuaciones.add("\n");
    }

    lineasTopPuntuaciones.add(tuPuntuacion);
    return lineasTopPuntuaciones;
}
```

```java
/**
 * Devuelve true si la puntuación es mayor que la que ya tenía el usuario
 *
 * Utilizará getPrincipals() sacando el sujeto que realizó el login (JAAS)
 */
public boolean cotejarPuntuaciones(int puntuacion) {
    boolean mayorPuntuacion = false;
    if (loginContext != null) {
        Subject sujeto = loginContext.getSubject();
        Set<Principal> principales = sujeto.getPrincipals();
        for (Principal principale : principales) {
            ImplementacionPrincipal principal = (ImplementacionPrincipal) principale;
            if (principal.getUsuario().getPuntuacion() < puntuacion) {
                EscribirFicheros.modificarPuntuacionUsuario(principal.getUsuario().getNick(), puntuacion);
                principal.getUsuario().setPuntuacion(puntuacion);
                mayorPuntuacion = true;
                break;
            }
        }
    }
    return mayorPuntuacion;
}
```

Daniel Tamargo

3. Métodos

c) Otros: Cliente

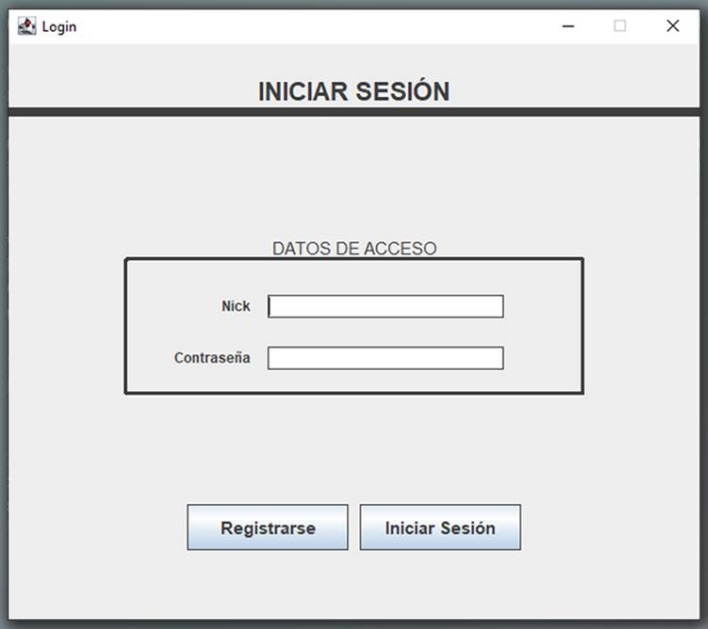
Algunos de los métodos por la clase Cliente

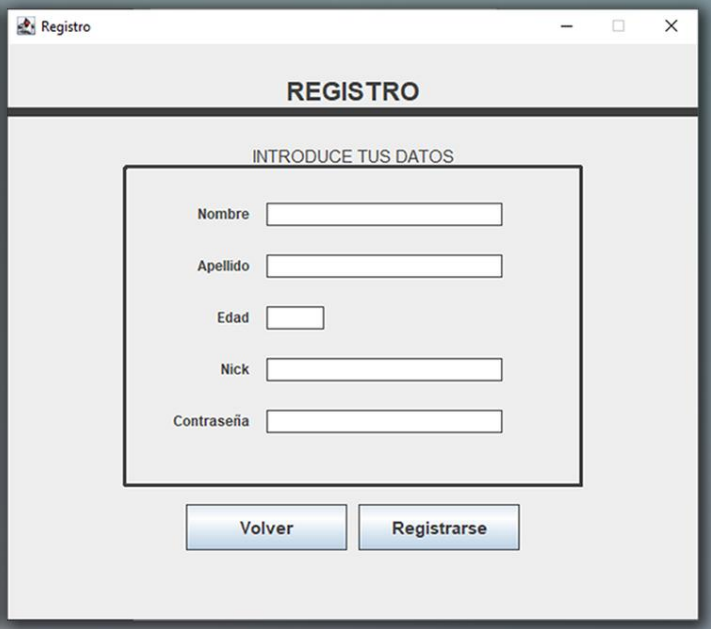
```java
// PUNTUACIONES
public void volcarDatosTextPane(JTextPane textPane, ArrayList<String> listaStrings, int tipo) {
    textPane.setText("");

    StyledDocument doc = textPane.getStyledDocument();

    Style style = textPane.addStyle( nm: "PlaylistStyle",  parent: null);
    StyleConstants.setForeground(style, Color.DARK_GRAY);

    String nums = "1234567890";

    try {
        boolean resetearColor = false;
        for (String str : listaStrings) {
            if (str.contains(".")) {
                if ((tipo == 2 && str.substring(0, str.indexOf('.')).equalsIgnoreCase(nickJugador))) {
                    StyleConstants.setForeground(style, Color.BLUE);
                    resetearColor = true;
                }
            }
            if ((tipo == 2 && str.contains("Tu puntuación"))
                    || (tipo == 1 && nums.contains(str.substring(0, 1)))) {
                StyleConstants.setForeground(style, Color.BLUE);
                resetearColor = true;
            }
            doc.insertString(doc.getLength(), str, style);

            if (resetearColor) {
                StyleConstants.setForeground(style, Color.DARK_GRAY);
                resetearColor = false;
            }
        }
    } catch (Ba
}
```

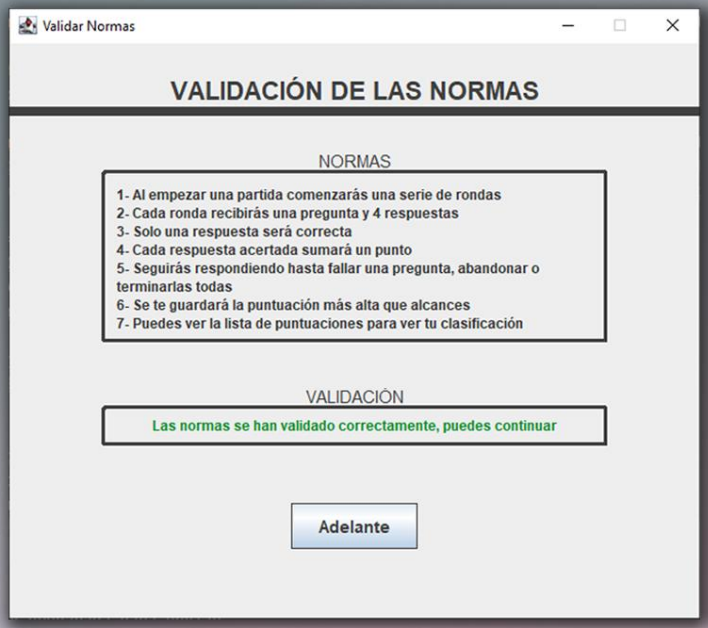
```java
public byte[] encriptarContrasenya() throws UnsupportedEncodingException, BadPaddingException, IllegalBlockSi
    // ENCRIPTAMOS LA CONTRASEÑA PARA QUE EL SERVIDOR NO PUEDA SABER CUÁL ES (SEGURIDAD DEL CLIENTE)
    // Vamos a crear una clave DES que SIEMPRE será la misma para que SIEMPRE dé el mismo resultado
    // UTF-8 es el por defecto de los algoritmos, pero mejor asegurar la consistencia
    final String utf8 = "utf-8";
    String contrasenyaCifradora = "ContraseñaSuperSecretaParaEncriptarContraseñas";
    byte[] keyBytes = Arrays.copyOf(contrasenyaCifradora.getBytes(utf8),  newLength: 24);
    SecretKey claveCifrarContrasenya = new SecretKeySpec(keyBytes,  algorithm: "DESede");

    // El vector debe tener una longitud de 8 bytes
    String vector = "ABCD1234";
    IvParameterSpec iv = new IvParameterSpec(vector.getBytes(utf8));

    // Creamos en encriptador
    Cipher encrypt = Cipher.getInstance("DESede/CBC/PKCS5Padding");
    encrypt.init(Cipher.ENCRYPT_MODE, claveCifrarContrasenya, iv);

    // Preparamos la contraseña encriptada para que el servidor no pueda saber cuál es la contraseña 'pura'
    byte[] bytesContrasenya = String.valueOf(tContrasenya.getPassword()).getBytes(utf8);
    return encrypt.doFinal(bytesContrasenya);
}
```

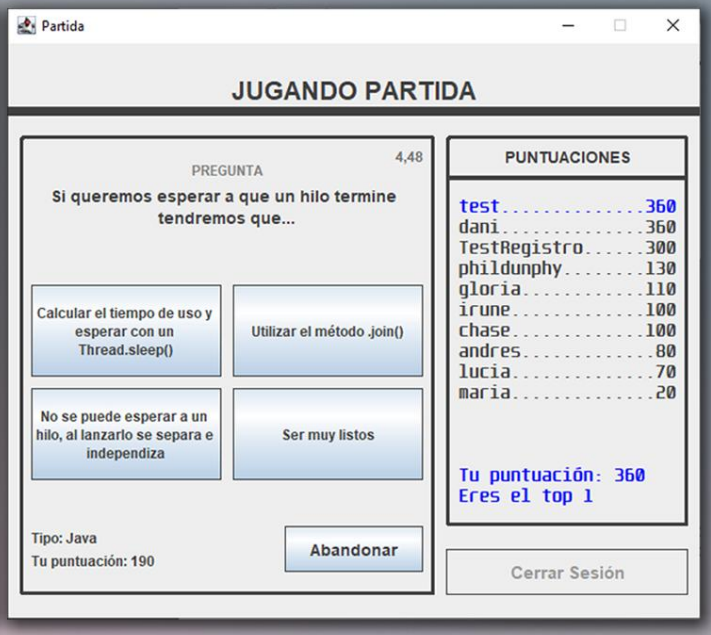
Daniel Tamargo

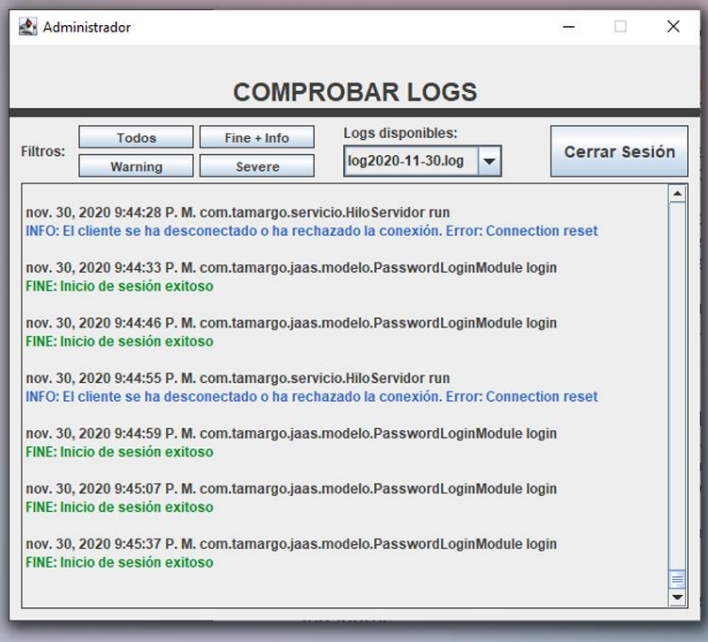
4. Ventana Cliente











Como he generado todos los Labels, Buttons, etc por código, realmente es una ventana donde reutilizo un panel en el que voy plasmando diferentes elementos e información

Daniel Tamargo

5. Logs

Utilizando la clase Logger, podremos guardar todos los logs que deseemos.

En mi caso he creado una clase GuardarLogs que inicializa el Logger y permite utilizarlo desde cualquier otra clase.

El log permite cualquier nivel, así, generaré un único log por día donde guardaré las notificaciones de los errores, avisos e información.

La propia clase GuardarLogs.java tiene un método que comprobará cuántos logs hay guardados y, a través de una variable que define el número máximo de logs que se pueden guardar, comprobará que no se están guardando más logs de los debidos, eliminando los logs sobrantes. Como los logs están ordenados por nombre y he utilizado la fecha en formato inglés, se eliminarán los logs más antiguos simplemente eliminando de arriba a abajo hasta quedarme solo con el número de logs que deseo.

A través de la ventana especial de administrador podremos visualizar de forma clara y elegante cada uno de los logs con cada registro, pudiendo filtrar en base al nivel del registro.

Aquí muestro un ejemplo de cómo guardo un Log desde el HiloServidor

```
case 3 -> { // PARTIDA
    System.out.println(this.nombre + "Comenzando una nueva partida con " + nickJugador);
    GuardarLogs.logger.log(Level.FINE, msg: "Comenzando una nueva partida con el usuario " + nickJugador);
    buclePartida(claveAES, objOS, objIS);
}
```