Juego de Piedra-Papel-Tijera 1.0

1. Enunciado

El objetivo es tener una primera versión que implemente el **juego de Piedra-Papel-Tijera** en **modo distribuido** (con servidor y varios clientes), con una solución basada en **sockets**, todo ello implementado como **aplicaciones de consola sin interfaz gráfico**.

El juego permitirá la comunicación remota entre varios clientes. Los **clientes se agrupan de dos en dos en lo que llamaremos "salas" (una sala sólo puede contener un máximo de dos clientes).** El servidor central recoge las jugadas de ambos jugadores y reenvía el resultado correspondiente sólo a los participantes de dicha sala.

La implementación se realizará con sockets Java, de tipo TCP, **orientados a conexión**. Por lo tanto, el servidor se implementará siguiendo un modelo *push*, donde el servidor guarda registro de los clientes, y reenvía los resultados recibidos, a los participantes actuales en una sala.

Por motivos de simplicidad, los usuarios se registran con un apodo (*nickname*) y se supondrá que nunca se registran dos usuarios con el mismo.

Los clientes de una sala enviarán un texto que se corresponde con los valores:
- PIEDRA o PAPEL TIJERA o LOGOUT
El texto introducido puede estar en minúsculas o mayúsculas indistintamente. El valor *logout* se interpretará por el servidor para cerrar la conexión con dicho cliente, y por otro lado, salir de la aplicación cliente.

El servidor recoge la jugada del usuario e informa del resultado, que puede ser victoria, derrota, empate (WIN, LOSE, DRAW respectivamente) si ambos jugadores han enviado una jugada o en espera (WAITING).

En este juego:

- Si en la sala sólo hay un valor recogido actualmente, se devuelve un resultado WAITING (se está esperando al segundo jugador a que envíe su jugada).
- Si coincide la jugada de ambos jugadores el resultado es empate: DRAW
- PAPEL vs. PIEDRA -> se da como ganador a PAPEL y perdedor a PIEDRA
- PIEDRA vs. TIJERA -> se da como ganador a PIEDRA y perdedor a TIJERA
- TIJERA vs. PAPEL -> se da como ganador a TIJERA y perdedor a PAPEL
- Una vez resuelto una confrontación se vuelve a empezar de cero la partida (se borran las jugadas de ambos oponentes).

La estructura de paquetes e interfaces/enumeraciones/clases es la siguiente (ver Tabla 1):

Paquete	Módulos / Clases	Nº	Descripción

`es.ubu.lsi.client`	`GameClient`
	`GameClientImpl`
	`GameClientListener`
		`1 interfaz`
		`1 clase`
		`1 clase interna`
			Clases para el cliente.

`es.ubu.lsi.common`	`GameElement`
	`ElementType`
	`GameResult`
		`1 clase`
		`1 enumeración interna`
		`1 enumeración`
			Clases comunes.

`es.ubu.lsi.server`	`GameServer`
	`GameServerImpl`
	`ServerThreadForClient`
		`1 interfaz`
		`1 clase`
		`1 clase interna`
			Clases para el servidor.

Tabla 1: Resumen de paquetes, interfaces, enumeraciones y clases

El código fuente de `ElementType.java` y `GameResult.java` está disponible en la plataforma UBUVirtual.

Es labor de los alumnos implementar los ficheros necesarios para el correcto cierre y ensamblaje del sistema, junto con el resto de productos indicados en el apartado ***5 Normas de Entrega.***

Para su resolución se dan las siguientes indicaciones, y diagramas de clases disponibles. Es decisión del alumno **la inclusión adicional de atributos y/o métodos, los diagramas pueden omitir atributos y métodos privados.**

Dadas las facilidades que ofrecen las clases internas (*inner classes*) en Java respecto al acceso a las propiedades de sus clases externas (*outer classes*), se recomienda seguir las indicaciones dadas en el enunciado

Los mensajes se trasmiten como **objetos serializados**, por lo que para la resolución de la práctica se deben usar flujos de entrada y salida de tipo `ObjectInputStream` y `ObjectOuputStream` del paquete `java.io`.

1.1. Paquete es.ubu.lsi.client

Comentarios respecto a la interfaz `GameClient`:

- Define la signatura de los métodos de envío de mensaje, desconexión y arranque.

Comentarios respecto a `GameClientImpl`:

- El cliente en su invocación recibe una dirección IP/nombre de máquina y un *nickname*. El puerto de conexión es siempre el **1500**. Si no se indica el equipo servidor, se toma como valor por defecto *localhost*.
  - Ej: `java es.ubu.lsi.client.GameClientImpl 10.168.168.13 minickname`
- Contiene un método `main` que arranca el hilo principal de ejecución del cliente: instancia el cliente y arranca adicionalmente (en el método `start`) un hilo adicional para **escuchar** respuestas del servidor, a través de `GameClientListener`.
- En el hilo principal se espera a la entrada de consola por parte del usuario para el **envío** de la jugada al servidor (flujo de salida, a través del método `sendElement`). Cuando se sale del bucle (Ej: `logout`) se desconecta.

Comentarios respecto a la clase (*inner class*) `GameClientListener`:

- Implementa la interfaz `Runnable`, por lo tanto, redefine el método `run` para ejecutar el hilo de **escucha** de mensajes del servidor (flujo de entrada) y mostrar los mensajes entrantes.

Por motivos de simplificación, y dado que sólo usaremos un único canal de salida (`System.out`) las salidas a pantalla (jugada enviada por el cliente vs. respuesta del servidor) puede ser que no siempre salgan ordenadas correctamente.

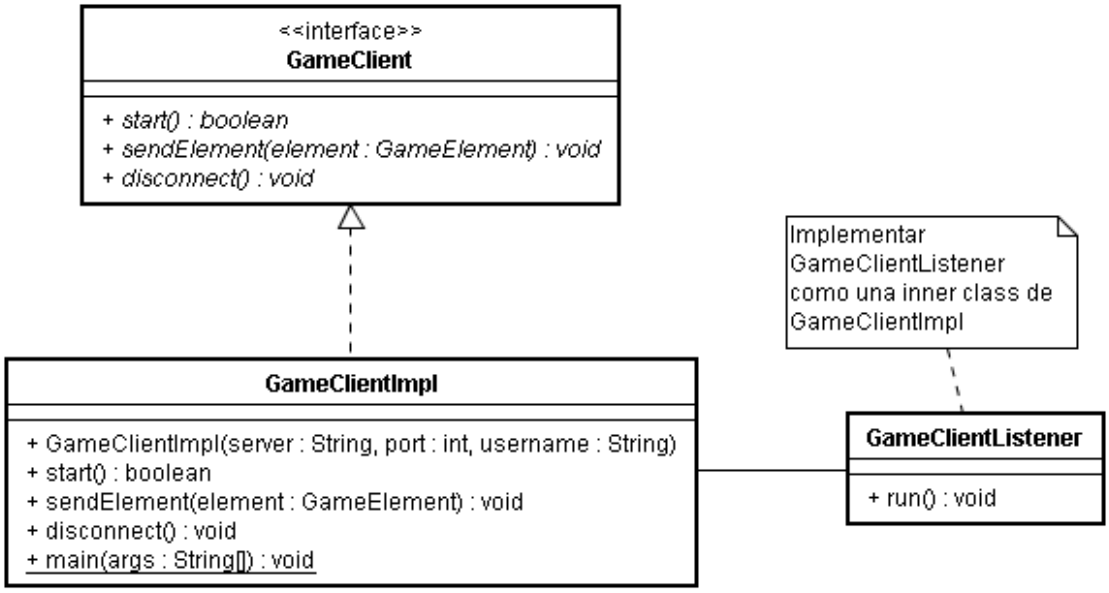

Ilustración 1. Diagrama de clases del paquete `es.ubu.lsi.client`

1.2. Paquete es.ubu.lsi.common

Comentarios respecto a la clase `GameElement`:

- Define el mensaje que se envía al servidor, incluyendo la jugada actual del jugador.

Comentarios respecto a la enumeración `ElementType`:

- Incluye las jugadas o mensaje de *logout* del cliente.

Comentarios respecto a la enumeración `GameResult`:

- Incluye las posibles respuestas del servidor.

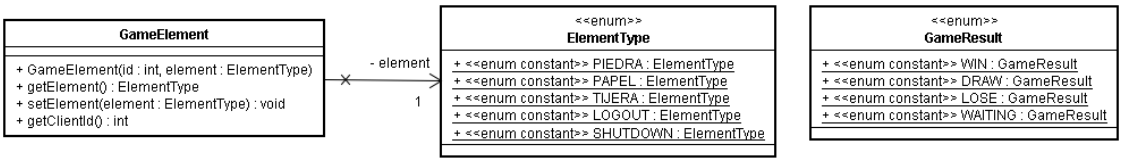

Ilustración 2. Diagrama de clases del paquete `es.ubu.lsi.common`.

1.3. Paquete es.ubu.lsi.server

Comentarios respecto a la interfaz `GameServer`:

- Define la signatura de los métodos de arranque, multidifusión y eliminación de clientes.

Comentarios respecto a la clase `GameServerImpl`:

- Por defecto el servidor se ejecuta en el puerto 1500. en su invocación no recibe argumentos.
  - Ej: `java es.ubu.lsi.server.GameServerImpl`

- Contiene un método `main` que arranca el hilo principal de ejecución del servidor: instancia el servidor y arranca (en el método `startup`).
- En el método `startup` se implementa el bucle con el servidor de sockets (`ServerSocket`), esperando y aceptado peticiones. Ante cada petición entrante y aceptada, se instancia un nuevo `ServerThreadForClient` y se arranca el hilo correspondiente para que cada cliente tenga su hilo independiente asociado en el servidor (con su socket, flujo de entrada y flujo de salida). Es importante ir guardando un registro de los hilos creados para poder posteriormente realizar el *push* de los mensajes y un apagado correcto.
- El método `broadcastRoom` envía el resultado sólo a los clientes de una determinada sala (flujo de salida).
- El método `remove` elimina un cliente de la lista.
- El método `shutdown` cierra los flujos de entrada/salida del servidor y el socket correspondiente a cada cliente.

Comentarios respecto a la clase `ServerThreadForClient`:

- La clase (*inner class*) extiende de `Thread`.

- En el método `run` espera en un bucle a los mensajes **recibidos** de cada cliente (flujo de entrada), realizándose la operación correspondiente (a través de los métodos de la clase externa `GameServer`).

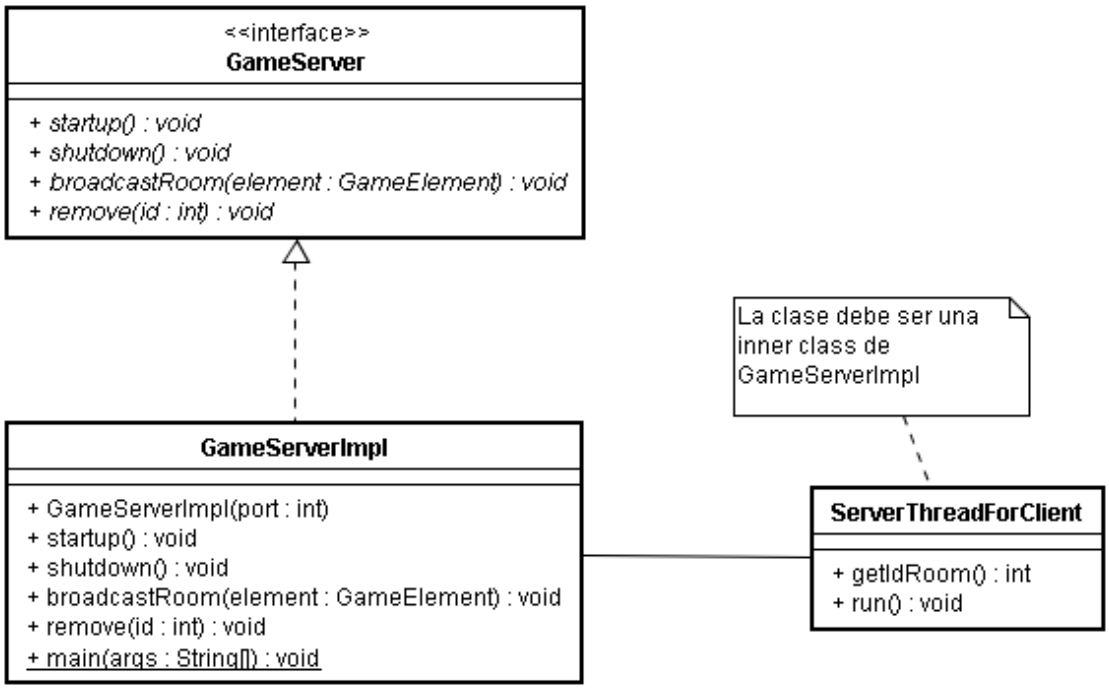

Ilustración 3. Diagrama de clases del paquete `es.ubu.lsi.server`.

2. Normas de Entrega

Fecha límite de entrega:

- **25 de marzo de 2022 (viernes) hasta las 23:30 horas**

Formato de entrega:

- Se enviará un fichero `.zip` a través de la plataforma **UBUVirtual** completando la tarea previa.
- El fichero comprimido seguirá el siguiente formato de nombre (sin tildes ni espacios en blanco):

  `NombrePrimerApelido-NombrePrimerApellido.zip`

  Ej: si los alumnos son Pio López y Blas Pérez, su fichero se llamará:

  `PioLopez-BlasPerez.zip`
- **Aunque la práctica se haga por parejas, se enviará por parte de cada uno de los dos integrantes.**
- **NO se admiten envíos posteriores a la fecha y hora límite, ni a través de otro medio que no sea la entrega de la tarea en UBUVirtual. Si no se respetan las normas de envío la calificación es cero.**

Dado que el desarrollo **se debe realizar con Maven**, la estructura de directorios y ficheros será la configuración por defecto con el arquetipo `maven-archetype-quickstart`. **Se entregará un único fichero `.zip` con la estructura comprimida del proyecto, después de realizar un `mvn clean`.**

Dentro del proyecto se añadirá un fichero `build.xml` para su ejecución con ANT que permita la generación de la documentación *javadoc* del proyecto en la carpeta `doc` (target con `name="javadoc"` utilizando el task `javadoc`).

Por otro lado se incluirán dos scripts (`.bat`) para la ejecución de servidor y cliente(s), con los correspondientes ajustes necesarios suponiendo que ejecutamos desde el directorio raíz del proyecto Maven:

- `runServer`
- `runClient`

Corrección, defensa de la práctica y comentarios adicionales.

- La práctica se realizará **individualmente**, pudiéndose solicitar **una defensa** de las mismas en horario de prácticas **La no realización de la defensa, si se solicita, supondría una calificación de cero en dicha práctica. Se debe tener en cuenta que lo visto en prácticas, también puede ser preguntado en las correspondientes pruebas de teoría.**
- Para aclarar cualquier duda, dado que el enunciado puede dar lugar a interpretaciones y deja abierto cierto margen de decisión, utilizad el foro habilitado. En caso de no aclararse las dudas en el foro, dirigirse al profesor en horario de tutorías.
- No se deben modificar los ficheros fuente proporcionados. En caso de ser necesario, por errores en el diseño / implementación de los mismos, se notificará para que sea el responsable de la asignatura quien publique en UBUVirtual la corrección y/o modificación. Se modificará el número de versión del fichero en correspondencia con la fecha de modificación y se publicará un listado de erratas.

Criterios de valoración.

- La práctica es obligatoria, entendiendo que su no presentación en la fecha marcada, supone una calificación de cero sobre el total de **1 punto** (sobre 10 del total de la asignatura) que se valora la práctica.

- Se valorará negativamente métodos con un número de líneas grande (>30 líneas) sin contar comentarios. Se debe evitar en la medida de lo posible la repetición de código.
- No se admiten ficheros cuya estructura y contenido no se adapte a lo indicado, con una valoración de cero.
- Se valorará el aspecto más o menos "profesional" que se dé a la aplicación.
- No se corrigen prácticas que no compilen, ni que contengan errores graves en ejecución (impidiendo básicamente jugar) con una valoración de cero.
- No se admite código no comentado con la especificación para **javadoc,** con una valoración de cero.