Chat 1.0

1. Enunciado

El objetivo es tener una primera versión que implemente un chat en modo texto (tanto servidor como cliente), con una solución basada en sockets.

El chat permitirá la comunicación remota entre varios clientes, utilizando un servidor central que recoge los mensajes, y reenvía los mismos a todos los participantes.

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 mensajes recibidos, uno a uno, a todos los participantes actuales.

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

Se manejarán dos tipos básicos de mensaje:
- *texto libre*: texto introducido por el cliente para retrasmitir a todos los otros clientes conectados
- *logout*: comando para cerrar la conexión y salir de la aplicación cliente.

Por otro lado, un usuario (cliente) tendrá la posibilidad de bloquear los mensajes de otro usuario incluso si el usuario a bloquear no está todavía conectado. Para realizar este bloqueo se implementará el comando "*ban*", pudiendo volver a aceptar sus mensajes con el comando "*unban*". Cuando un usuario haya sido bloqueado, simplemente no se mostrarán sus mensajes. Ejemplo:

- *ban usuariobaneado*
- *unban usuariounbaneado*

Cuando un usuario bloquea a otro, se deberá comunicar este hecho al servidor a través de un mensaje del tipo "*usuarioactual ha baneado a usuariobaneado*".

La estructura de paquetes y módulos/clases es la siguiente (ver Tabla 1):

Paquete	Módulos / Clases	Nº	Descripción

`es.ubu.lsi.client`	`ChatClient`
	`ChatClientImpl`
	`ChatClientListener`
		`1 interfaz`
		`1 clase`
		`1 clase interna`
			Clases para el cliente.

`es.ubu.lsi.common`	`ChatMesage` `MessageType`	`1 clase` `1 clase interna`	Clases comunes.

`es.ubu.lsi.server`	`ChatServer`
	`ChatServerImpl`
	`ServerThreadForClient`
		`1 interfaz`
		`1 clase`
		`1 clase interna`
			Clases para el servidor.

Tabla 1: Resumen de paquetes y módulos

El código fuente de `ChatMessage.java` y `MessageType.java` están ya disponibles[1].

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 ***4*** 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.**

[1]Disponibles en UBUVirtual.

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 `ChatClient`:

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

Comentarios respecto a `ChatClientImpl`:

- 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.ChatClientImpl 10.168.168.13 jpseco`
- 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 a través de `ChatClientListener`.
- En el hilo principal se espera a la entrada de consola por parte del usuario para el envío del mensaje (flujo de salida, a través del método `sendMessage`). Cuando se sale del bucle (Ej: `logout`) se desconecta.

Comentarios respecto a `ChatClientListener`:

- 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.

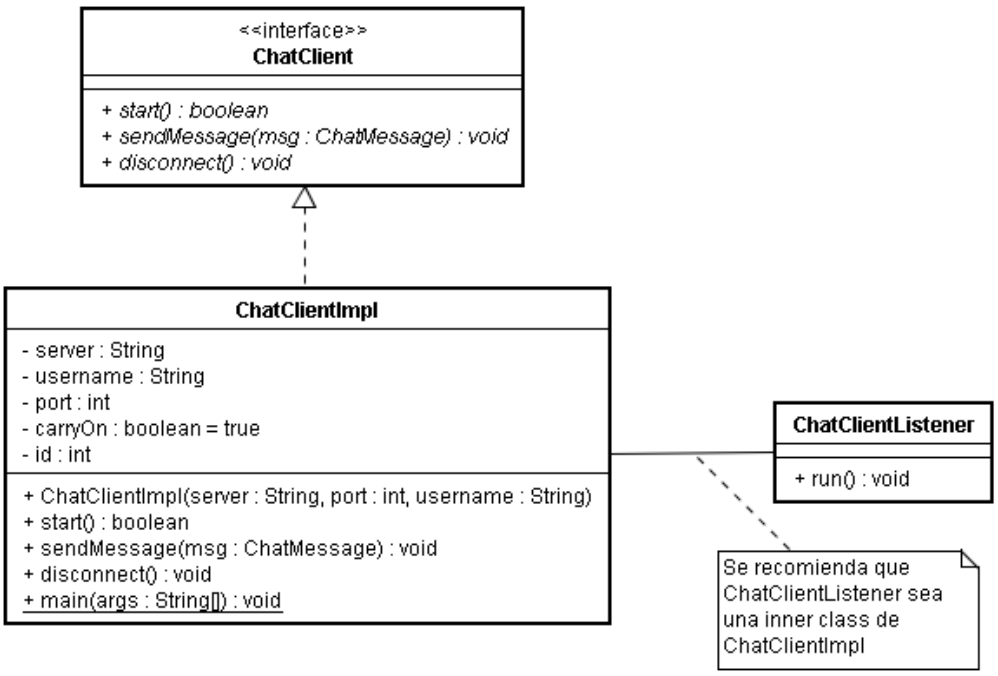

Figura 1. Diagrama de clases del paquete `es.ubu.lsi.client`

1.2. Paquete es.ubu.lsi.common

Comentarios respecto a la clase `ChatMessage`:

- Mensaje que se envía y recibe en el chat. Encapsula el identificador del cliente que genera el mensaje, el tipo y en algunos casos el texto a mostrar.

Comentarios respecto a la interfaz `MessageType`:

- Enumeración interna del mensaje. Define mensajes de tipo texto y *logout* (para cerrar sesión). El tercer tipo de mensaje (*shutdown*) se define para usos futuros, pero no se requiere su uso en esta práctica.

1.3. Paquete es.ubu.lsi.server

Comentarios respecto a la interfaz `ChatServer`:

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

Comentarios respecto a la clase `ChatServerImpl`:

- Por defecto el servidor se ejecuta en el puerto 1500. en su invocación no recibe argumentos.
  - Ej: `java es.ubu.lsi.server.ChatServerImpl`
- 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 `broadcast` envía el mensaje recepcionado a todos los clientes (flujo de salida).
- El método `remove` elimina un cliente de la lista.
- El método `shutdown` cierra los flujos de entrada/salida y el socket correspondiente de cada cliente.

Comentarios respecto a la clase `ChatServerThreadForClient`:

- La clase extiende de `Thread`.

- En el método `run` se 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, `ChatServer`). A la finalización de su ejecución se debe eliminar al propio cliente de la lista de clientes activos.

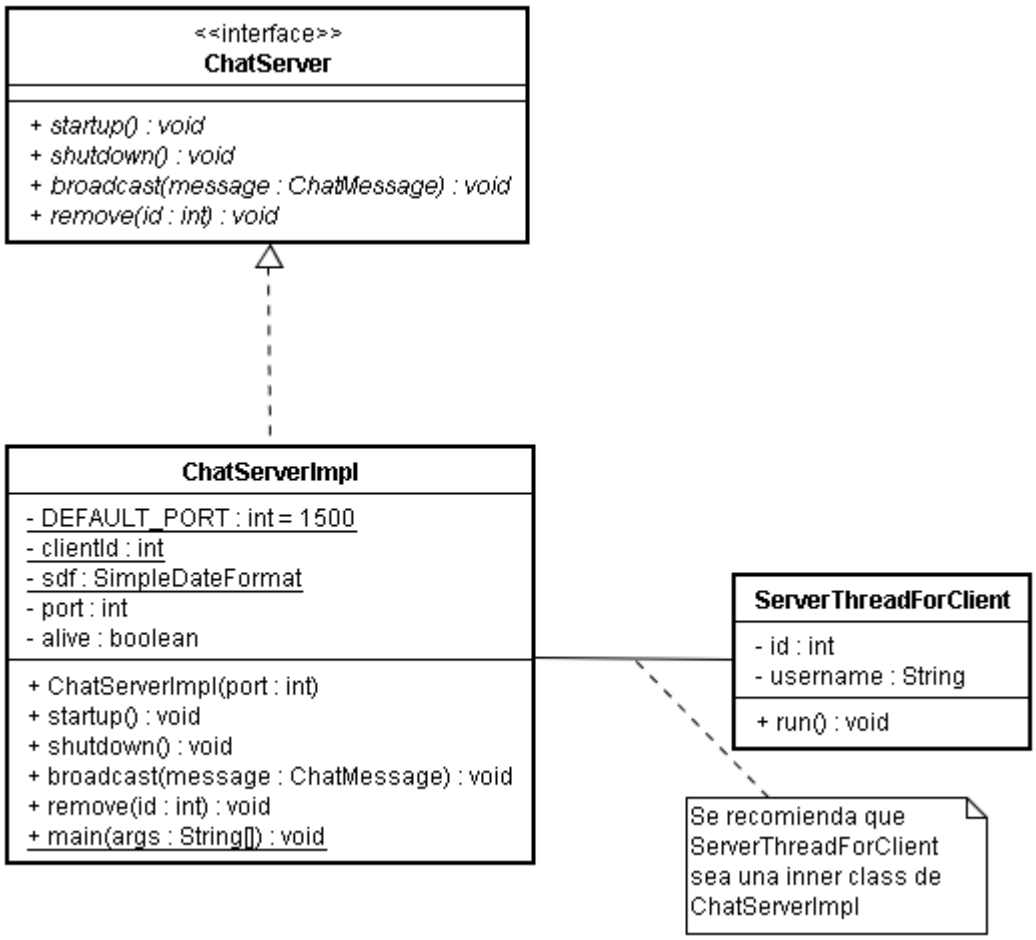

Figura 2: Diagrama de clases del paquete `es.ubu.lsi.server`

2. Normas de Entrega

Fecha límite de entrega:

- **26 de marzo de 2024 (martes) hasta las 21:30 horas**

Formato de entrega:

- Tal y cómo hemos hablado en clase, la entrega se realizará a través de un repositorio de github.
- Cualquier entrega fuera del repositorio de github, no será tenida en cuenta.
- En lo que respecta a fechas de entrega, al estar trabajando con github, se evaluará la práctica entregada cuyo commit, sea más próximo a la fecha de entrega, a no ser que el alumno especifique lo contrario dentro de la entrega, en cualquier caso nunca se tendrán en cuenta fuentes o documentos entregados cuyo cómic sea posterior a la última fecha de entrega permitida.
- En caso de participar varios miembros en el equipo, debe constar en el repositorio de git la participación con los comités correspondientes de cada uno de ellos, debéis recordar que en esta asignatura uno de los objetivos es prepararos para seguir el flujo de despliegue continuo e integración continua con trabajo en equipo.
- **Salvo en casos excepcionales, 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`).

Correción, defensa de la práctica y comentarios adicionales.

- La práctica se realizará por grupos de **dos personas máximo o bien 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.**
- Para aclarar cualquier duda utilizar 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 (>50 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.
- No se corrigen prácticas que no compilen, ni que contengan errores graves en ejecución con una valoración de cero.
- No se admite código no comentado con la especificación para **javadoc** con una valoración de cero.