Enunciado Práctica 2 con comentarios

# Webchat: Chat en grupo vía web

## Objetivo

Que el alumno ponga en práctica los conocimientos teóricos y prácticos adquiridos en la Parte II de la asignatura de Programación Concurrente.

La práctica podrá realizarse de forma individual o por parejas.

## Fecha y mecanismo de entrega

La entrega se realizará con fecha tope el Viernes 13 de Enero a las 11:00h de la mañana por el campus virtual. Se deberá entregar un .zip cuyo contenido será el proyecto eclipse en el que se ha realizado la práctica. Las carpetas src, target y el fichero pom.xml deberán estar en la raíz del fichero comprimido (no debe haber una carpeta "practica" o similar).

El nombre del fichero .zip tiene que ser igual al identificador del alumno en la URJC (la parte antes de la @ del correo electrónico del alumno en la URJC). En caso de dos alumnos, se incluirán ambos nombres separados por "-".

En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o alumnos):

```
<groupId>es.codeurjc.pc</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del envío del código, se realizará un examen oral de la práctica para que el profesor pueda verificar que ambos alumnos conocen el funcionamiento de la práctica entregada. Durante el examen los alumnos tendrán delante el código fuente de la práctica y se les pedirán explicaciones sobre el mismo y su ejecución.

El examen se llevará a cabo el mismo día 13 de Enero de 12:00 a 14:00 y de 15:00 a 18:00. El aula está todavía por determinar y será notificada desde el campus virtual. Los alumnos seleccionarán la franja horaria en la que desean defender la práctica usando esta página web:

https://docs.google.com/spreadsheets/d/14RQ4lV8IaQGzYoITYz1wteSBaoYrZB07bU9U790h-UI/edit?usp=sharing

## Información de interés

Se recomienda a los alumnos que visiten frecuentemente el foro de la asignatura en el campus virtual para estar informado sobre actualizaciones, consejos o cualquier otra información relacionada con la práctica.

## Enunciado

Una famosa empresa de Internet llamda **Goobook** quiere implementar un chat en grupo vía web. Para ello, sus desarrolladores han empezado a implementar un primer prototipo básico. Este prototipo básico tiene las siguientes funcionalidades:

- Un usuario puede crear un grupo.
- Un usuario puede registrarse en un grupo creado por otro usuario.
- Un usuario puede enviar un mensaje a un grupo que será recibido por el resto de usuarios registrados en ese grupo.
- Un usuario puede cerrar un grupo creado por él.

Cuando hay pocos usuarios y no escriben muy rápido el prototipo parece que funciona correctamente, pero cuando el número de usuarios crece y los mensajes son muy rápidos el prototipo empieza a tener los siguientes problemas:

- Se generan excepciones de tipo ConcurrentModificationException.
- Varios usuarios registrados en el mismo grupo no son coherentes entre sí y no ven a los mismos usuarios conectados al grupo.
- Cuando un usuario tiene una conexión muy lenta hace que otros usuarios conectados al mismo grupo tarden mucho tiempo en recibir los mensajes.
- Los mensajes no llegan ordenados en el mismo orden a todos los usuarios.

Como os podéis imaginar, estos problemas parecen estar motivados por condiciones de carrera debidas a una forma no adecuada de gestionar los aspectos concurrentes de la aplicación.

Para solucionar esos problemas la empresa ha contratado una consultoría especializada. No obstante, interesa solucionar los problemas cuanto antes y el experto no llegará hasta dentro de un mes. Casualmente estamos haciendo las prácticas en dicha empresa y aprovechando que estamos estudiando en profundidad cómo desarrollar aplicaciones concurrentes nos hemos ofrecido para intentar solucionar los problemas. Como es lógico, nuestro jefe no se fía de nosotros y piensa que no seremos capaces, pero nosotros le aseguramos que podremos hacerlo. Finalmente nuestro jefe accede, pero nos dice que si somos tan buenos, no tendremos problemas en implementar unas cuantas funcionalidades más.

A continuación se describe cómo cargar el proyecto Webchat en eclipse para que empecéis a trabajar en el código. Luego se describen los aspectos importantes de la implementación del prototipo. Seguidamente se describe en detalle cómo hacer tests que verifiquen el correcto funcionamiento de las partes de las que se compone la aplicación. Por último, se enumeran una por

una las mejoras que deben realizarse para que la aplicación funcione correctamente y se indican las nuevas funcionalidades.

## Cargar Webchat en Eclipse

El prototipo inicial de Webchat se puede descargar como un fichero .zip desde la página web de la asignatura. Una vez descargado el .zip, se tendrá que descomprimir en una carpeta dentro del workspace de eclipse.

Una vez descomprimido, tenemos que dar de alta el proyecto en el propio eclipse. Para ello, tenemos que ejecutar el comando File > Import..., luego seleccionar en la sección "Maven" la opción "Existing Maven Projects" y aparecerá un cuadro de diálogo. En ese cuadro, pulsar en el botón "Browse..." y seleccionar la carpeta en la que hemos descomprimido el .zip. En ese momento aparecerá en la lista de "Projects" el proyecto "webchat.enunciado" y pulsamos en finish.

En ese momento el proyecto se cargará en eclipse y se comenzarán a descargar en segundo plano todas las librerías necesarias para que funcione correctamente. Es decir, en el momento de importar el proyecto en Eclipse es necesario tener conexión a Internet.

## Cómo usar Webchat

El prototipo es una aplicación web que permite crear un chat en grupo. El funcionamiento es el siguiente:

- Se inicia la aplicación ejecutando la clase App (que contiene el método main). Esto se puede hacer pulsando botón derecho sobre la clase y seleccionar "Run as... > Java Application".
- Se espera hasta que aparece en la consola un mensaje que contiene algo como:

  ```
  FrameworkServlet 'dispatcherServlet': initialization completed in 10 ms
  ```

- Se abre un navegador y se conecta a la url http://127.0.0.1:8080/
- Aparecerá un cuadro de diálogo en el que se indica el nombre del chat (la aplicación permite mantener varios chats a la vez).
- Aparecerá otro cuadro de diálogo en el que se indica el nombre del usuario.
- Finalmente aparecerá la página web que contiene el chat.


- Para establecer una comunicación en el chat, bastará con abrir otra página en el navegador y

seguir los mismos pasos. Si se quiere establecer una comunicación entre varios ordenadores, el ordenador remoto tendrá que usar la IP del ordenador que contiene la aplicación (en vez de usar la IP 127.0.0.1).

## Detalles de implementación de Webchat

Webchat está formado por las siguientes clases:

- **App**: Clase principal de la aplicación. Contiene el método main que deberá ser ejecutado.
- **ChatHandler**: Clase encargada de atender las peticiones que llegan desde el navegador usando websockets.
- **Chat**: Clase que representa un chat en grupo. Contiene la lista de usuarios del chat y es la encargada de redirigir los mensajes a todos ellos.
- **ChatManager**: Clase encargada de gestionar todos los chats activos en la aplicación. Dispone de métodos para crear chats y recuperar los chats existentes.
- **WebSocketUser:** Clase encargada de gestionar a los usuarios conectados al chat. Alberga su nombre y color asignado al mismo. Tiene un método por cada mensaje que se le puede enviar a un usuario. Es decir, al invocar ese método, se le enviará ese mensaje al usuario.
- **User**: Interfaz que representa a los usuarios. Este interfaz está implementado por la clase WebSocketUser. El objetivo de este interfaz es que las clases Chat y ChatManager no estén acopladas a una implementación concreta de los usuarios. Esto es especialmente útil para los tests.

Para solucionar los problemas de concurrencia del prototipo de Webchat tan sólo es necesario modificar las clases Chat y ChatManager, así que estudiaremos esas clases con un poco más de detalle:

**Clase Chat**

Esta clase es la encargada de gestionar un chat en grupo. Tiene como atributos el nombre del chat, una referencia al ChatManager y un mapa con los usuarios asociados a ese chat. Las operaciones de esa clase son:

- Añadir, eliminar y consultar los usuarios del chat.
- Enviar un mensaje a los usuarios del chat
- Cerrar el chat

Hay que destacar que, cuando se añade o se elimina un usuario del chat, se notifica al resto de usuarios para que lo muestren en la interfaz de usuario. Para ello, se invoca el método correspondiente en la interfaz User, lo que se traducirá en el envío de un mensaje al navegador de ese usuario.

A continuación se muestra la implementación de esta clase:

```java
public class Chat {

   private String name;
   private Map<String, User> users = new HashMap<>();

   private ChatManager chatManager;


   public Chat(ChatManager chatManager, String name) {
      this.chatManager = chatManager;
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void addUser(User user) {
      users.put(user.getName(), user);
      for(User u : users.values()){
         if (u != user) {
               u.newUserInChat(this, user);
         }
      }
   }

   public void removeUser(User user) {
      users.remove(user.getName());
      for(User u : users.values()){
         u.userExitedFromChat(this, user);
      }
   }

   public Collection<User> getUsers() {
      return Collections.unmodifiableCollection(users.values());
   }

   public User getUser(String name) {
      return users.get(name);
   }

   public void sendMessage(User user, String message) {
      for(User u : users.values()){
         u.newMessage(this, user, message);
      }
   }

   public void close() {
      this.chatManager.closeChat(this);
   }
}
```

**Clase ChatManager**

Esta clase es la encargada de gestionar todos los chats y todos los usuarios de la aplicación. Para ello, básicamente tiene como atributos dos mapas con los chats y los usurios. También tiene un número máximo de chats para no sobrecargar la máquina. Las operaciones de esa clase son:

- Dar de alta un nuevo usuario

- Crear un chat (o devolver el ya creado si existe uno)
- Cerrar un chat

Hay que destacar que, cuando se añade o se elimina un chat, se notifica al resto de usuarios para que lo muestren en la interfaz de usuario. Para ello, se invoca el método correspondiente en la interfaz User, lo que se traducirá en el envío de un mensaje al navegador de ese usuario.

A continuación se muestra la implementación de esta clase:

```
public class ChatManager {

   private Map<String, Chat> chats = new HashMap<>();
   private Map<String, User> users = new HashMap<>();
   private int maxChats;

   public ChatManager(int maxChats) {
      this.maxChats = maxChats;
   }

   public void newUser(User user) {

      if(users.containsKey(user.getName())){
         throw new IllegalArgumentException("There is already a user with name \'"
                     + user.getName() + "\'");
      } else {
         users.put(user.getName(), user);
      }
   }

   public Chat newChat(String name, long timeout, TimeUnit unit) throws InterruptedException,
         TimeoutException {

      if (chats.size() == maxChats) {
         throw new TimeoutException("There is no enought capacity to create a new chat");
      }

      if(chats.containsKey(name)){
         return chats.get(name);
      } else {
         Chat newChat = new Chat(this, name);
         chats.put(name, newChat);

         for(User user : users.values()){
               user.newChat(newChat);
         }

         return newChat;
      }
   }
```

```
    public void closeChat(Chat chat) {
        Chat removedChat = chats.remove(chat.getName());
        if (removedChat == null) {
            throw new IllegalArgumentException("Trying to remove an unknown chat with name \'"
                    + chat.getName() + "\'");
        }

        for(User user : users.values()){
            user.chatClosed(removedChat);
        }
    }

    public Collection<Chat> getChats() {
        return Collections.unmodifiableCollection(chats.values());
    }


    public Chat getChat(String chatName) {
        return chats.get(chatName);
    }

    public Collection<User> getUsers() {
        return Collections.unmodifiableCollection(users.values());
    }

    public User getUser(String userName) {
        return users.get(userName);
    }

    public void close() {}
}
```

Los detalles de implementación del resto de clases no son relevantes, aunque podéis echar un vistazo si tenéis interés en aprender cómo funcionan.

Lo más importante para solucionar los problemas de concurrencia del prototipo es saber que **las peticiones que llegan de cada usuario se ejecutan en su propio hilo**. Dicho de otra forma, si hay tres usuario en un mismo chat, y cada uno de ellos escribe un mensaje en el navegador, se llamará tres veces al método `sendMessage(User user, String message)` de la clase Chat y en cada llamada se usará un **hilo de ejecución diferente**.

## Tests en Webchat

Además de las clases del propio programa, el prototipo de Webchat dispone de varios tests que verifican el correcto funcionamiento del mismo. Los tests son mini-programas que comprueban de forma automática que una determinada funcionalidad que está implementada se comporta como se espera. Los tests son especialmente útiles para asegurarnos de que algo que tenemos implementado y funciona no "se rompe" al añadir nueva funcionalidad. Los tests de Webchat están implementados con la herramienta JUnit (http://junit.org/), que se utiliza bastante en el ámbito profesional para la implementación de tests en Java.

Cada test se implementa como un método de un clase. A continuación se muestra el test de Webchat que verifica que cuando se añade un nuevo chat, se invoca el método `newChat(...)` por cada uno de los usuarios suscritos en el `ChatManager` y el nombre del nuevo chat corresponde con el que se ha

creado.

```
@Test
public void newChat() throws InterruptedException, TimeoutException {

   //1) Crear el chat Manager
   ChatManager chatManager = new ChatManager(5);

   //2) Crear un usuario que guarda en chatName el nombre del nuevo chat
   final String[] chatName = new String[1];

   chatManager.newUser(new TestUser("user") {
      public void newChat(Chat chat) {
         chatName[0] = chat.getName();
      }
   });

   //3) Crear un nuevo chat en el chatManager
   chatManager.newChat("Chat", 5, TimeUnit.SECONDS);

   //4) Comprobar que el chat recibido en el método 'newChat' se llama 'Chat'
   assertTrue("The method 'newChat' should be invoked with 'Chat', but the value is "
            + chatName[0], Objects.equals(chatName[0], "Chat"));
}
```

Para ello, el test lo que hace es lo siguiente:

- 1) Crea un objeto de la clase `ChatManager` que permite un número máximo de 5 chats.

  ```
  ChatManager chatManager = new ChatManager(5);
  ```

- 2) Da de alta un usuario en el nuevo manager. En vez de crear un objeto de la clase `WebSocketUser`, crea un objeto de la clase `TestUser`. Esta clase implementa el interfaz `User`, pero en su implementación únicamente muestra por pantalla el método que ha sido invocado. La clase `TestUser` está especialmente diseñada para implementar tests, porque no necesita tener un servidor web real para funcionar. Se puede crear un objeto de la clase `TestUser`, pero es mucho más útil si se crea una clase anónima hija de `TestUser` y se redefine alguno de sus métodos. A continuación se muestra un fragmento de código que crea una clase anónima que hereda de `TestUser` y redefine el método `newChat(...)` para guardar en un array el nombre del chat.

  ```
  final String[] chatName = new String[1];

  TestUser user = new TestUser("user") {
     public void newChat(Chat chat) {
        chatName[0] = chat.getName();
     }
  }
  ```

  En este fragmento se puede ver cómo se puede redefinir un método para que guarde en una variable del test la información que recibe como parámetro. Hay que notar que la variable tiene que ser final (no se puede cambiar su valor), así que usamos un pequeño truco, declarar la variable como un array de una posición y hacer que la clase anónima escriba el valor en dicha posición. De esa forma, ese valor estará disponible para el propio test.

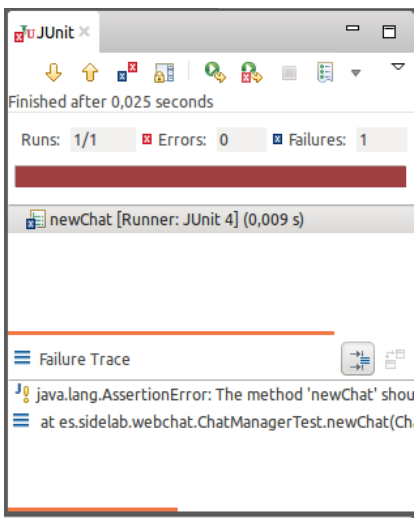
- 3) Crea un nuevo chat en el ChatManager:

```
chatManager.newChat("Chat", 5, TimeUnit.SECONDS);
```

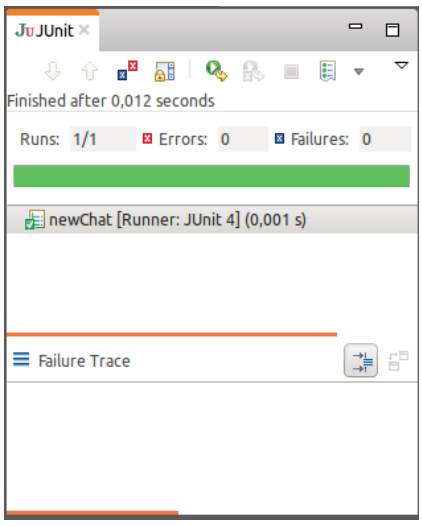
- 4) Por último, el test comprueba que el nombre del chat recibido en el método newChat es igual que el nombre que se pasa cuando se crea el nuevo Chat. Para ello, se utiliza el método estático assertTrue(...) que verifica que la expresión que se le pasa como parámetro es true. En caso de que no lo sea, el test falla indicando el mensaje que se le pasa como primer parámetro:

```
assertTrue("The method 'newChat' should be invoked with 'Chat', but the value is "
          + chatName[0], Objects.equals(chatName[0], "Chat"));
```

Para ejecutar este test en eclipse, basta con hacer botón derecho sobre el nombre del método y seleccionar la opción "Run as... > JUnit Test". Durante la ejecución, aparecerá una vista en eclipse que muestra el resultado de los tests. El color verde significa que los tests han "pasado", es decir, que han comprobado que el comportamiento del código es el esperado. En caso de que los tests fallen, se mostrará el color rojo con el mensaje que indica qué ha ido mal.




Test con fallo




Test correcto

Se considera que un test "no pasa", cuando se da alguna de las siguientes situaciones:

- Se eleva una excepción no capturada durante su ejecución
- Se invoca el método Assert.fail
- Se invoca el método Assert.assertTrue con una expresión que se evalúa a false.

Se considera que un test pasa en cualquier otro caso. Es decir, si finaliza correctamente y todos los asserts ejecutados son correctos.

Es importante tener en cuenta algunos aspectos de JUnit cuando estamos implementando tests para **código concurrente**:

- Si el test termina sin excepción, ni llamada a fail, ni llamada a assertTrue con una expresión falsa, se considera el test correcto. Es decir, los tests no esperan a que cualquier hilo creado haya terminado. Si deseamos que nuestro test se espere, tendremos que implementarlo de forma explícita en el test.
- Para que un test falle, se debe llamar al método fail o al método assertTrue en el hilo que ejecuta el propio test. Si se llama a estos métodos en otro hilo se provocará que esos hilos finalicen su ejecución, pero eso no implica que el test falle. Es decir, si dentro de un hilo detectamos alguna situación que indica que el test no debe pasar (es decir, debe finalizar con error o fallo), entonces habrá que comunicar dicha situación al hilo del test para que invoque el método adecuado.

Por último, cabe destacar que aunque un test se implementa en un método. Se suelen crear varias clases agrupando los tests relacionados entre sí. No hay una regla fija, pero cuando la clase empieza a tener más de cinco métodos de test, quizás convenga dividirla en dos separando los tests por grupos relacionados entre sí.

## Mejoras a implementar en Webchat

A continuación se describen una por una las mejoras que deben realizarse en el prototipo inicial de Webchat para evitar que tenga problemas de concurrencia:

- **Mejora 1:** Implementar un test que verifique que no hay errores de concurrencia en las clases ChatManager y Chat. Este test tiene que realizar lo siguiente:
  - Crear un ChatManager con un tope de 50 chats.
  - Ejecutar 4 hilos en paralelo simulando 4 usuario concurrentes.
  - Cada hilo creará un objeto TestUser y lo registrará en el ChatManager.
  - A continuación, el hilo repetirá 5 veces las siguientes acciones:
    - Creará un chat llamado “chat”+número de iteración (método manager.newChat)
    - Se registrará en ese chat (método chat.addUser(user))
    - Mostrará por pantalla todos los usuarios de ese chat.

  El test debería fallar en cuanto uno de los usuarios encuentren un error. Dicho de otro modo, en cuando en alguno de los hilos se eleve una excepción, el test debería fallar y no continuar su ejecución.

  Por otro lado, hasta que los errores de concurrencia no se solucionen, este test debería fallar elevando una excepción de tipo ConcurrentModificationException.
- **Mejora 2:** Realizar los cambios necesarios en las clases ChatManager y Chat para tengan el mismo comportamiento que en el prototipo pero que sean thread-safe, es decir, que sus métodos puedan ser invocados por varios hilos de ejecución. Cuando se implemente esta mejora, el test implementado en la Mejora 1 no debería fallar.
- **Mejora 3:** Uno de los problemas que tiene la implementación actual es que un usuario con

una conexión a Internet lenta hace que los mensajes se retrasen y tarden mucho en llegar a otros usuarios. Esto se debe a que en el prototipo, la notificación al resto de usuarios del chat se realiza secuencialmente en el mismo hilo de ejecución en el que se ejecuta la acción del usuario. Para evitar este problema, la notificación a los usuarios deberá realizarse en paralelo y en hilos diferentes al hilo que realiza la acción. No obstante, los usuarios (objetos que implementan el interfaz User) no son objetos thread-safe. Por este motivo, cuando se invoque un método en un usuario, no podrá haber otro hilo invocando métodos en ese usuario. Por último, hay que asegurarse de que las operaciones se ejecutan en el usuario en el orden en que fueron recibidas. Es decir, si hay que enviar dos mensajes a un usuario, tienen que llegarle en el orden en el que se han recibido en el servidor.

- **Mejora 4:** Para asegurarnos de que la mejora 3 se ha implementado correctamente, se deberá crear dos tests que verifiquen lo siguiente:
  - **Test de notificaciones en paralelo:** Se deberá implementar un test que compruebe que las notificaciones a los usuarios se ejecutan en hilos diferentes al hilo que ejecutó la acción (crear un chat, añadir un usuario, etc...) y estos a su vez en paralelo.
  - **Test de orden de los mensajes:** Se deberá implementar un test que compruebe que las notificaciones para un usuario en concreto se ejecutan en el orden en que fueron realizadas. Para forzar que las notificaciones tienen que "esperar" a que se haya terminado de procesar la anterior, se puede simular con un Thread.sleep(500) un tiempo grande de procesamiento.
- **Mejora 5:** Además, una vez verificado que los métodos de los usuarios se ejecutan en hilos diferentes, se deberá verificar que funcionan correctamente. Es decir, se deberá implementar un test para cada una de las siguientes situaciones:
  - La creación de un nuevo chat.
  - La eliminación de un chat existente.
  - El alta de un nuevo usuario en un chat.
  - La salida de un usuario de un chat.
  - El envío de un mensaje a un chat.

  En cada uno de estos tests se deberá verificar que la información que le llega al usuario es correcta. Por ejemplo, si un usuario envía un mensaje a un chat, deberá verificarse que el resto de usuarios del chat reciben ese mensaje (y no otro).

## Nuevas funcionalidades en Webchat

Nuestro jefe ha quedado gratamente sorprendido de que hayamos solucionado todos los problemas de concurrencia del prototipo de Webchat. Por este motivo, nos pide que le implementemos unas cuantas funcionalidades extra al prototipo. A continuación se describen esas funcionalidades:

- **Funcionalidad 1: Tiempo de espera máximo al crear el chat:** Cuando se intente crear un chat pero ya se haya alcanzado el número máximo de chats permitidos, el hilo se debera

esperar como máximo el tiempo indicado en el método. En el momento en que algún chat se cierre, el método creará el chat y retornará. En caso de que se alcance el timeout indicado sin la posiblidad de crear un chat, entonces se deberá lanzar la excepción TimeoutException. Para verificar que la funcionalidad se ha implementado correctamente, hay que implementar dos tests, uno para el caso de que salte el timeout y otro para el caso de que no se puede crear el chat inmediatamente, es decir, que se ha alcanzado el número máximo, pero antes de que salte el timeout, otro chat se cierra y deja hueco a la creación que está a la espera.

- **Funcionalidad 2: Cálculo periódico de estadísticas.** Para que los administradores de Webchat puedan tener información sobre el uso de la aplicación web, necesitan que se calculen de forma periódica unas estadísticas de uso. En concreto, se necesita la siguiente información en un momento determinado: Número de chats, Número total de usuarios conectados, número medio de usuarios por chat, chat con más número de usuarios y chat con menos número de usuarios. Estos estadísticos tendrán que calcularse cada 3 segundos. Además, para aprovechar la potencia de cómputo disponible, dichos estadísticos tendrán que calcularse como varias tareas que se ejecutan en paralelo. Es muy importante que los estadísticos sean coherentes entre sí. Por ejemplo, si durante el cálculo de los estadísticos se añade un nuevo chat, no es razonable que el número total no le haya tenido en cuenta pero sí se haya tenido en cuenta para calcular en chat con menor número de usuarios. Para esta funcionalidad no es necesario incluir un test que verifique el correcto funcionamiento de la misma.

## Se pide

Implementar todas y cada una de las mejoras y las nuevas funcionalidades en la aplicación Webchat. Si no se implementa alguna de ellas la práctica puede considerarse suspensa.

En la evaluación el aspecto más importante es que el programa no presente problemas concurrente como la posibilidad de condiciones de carrera o el uso de objetos no thread-safe desde varios hilos. Además, de forma complementaria, también se valorarán las buenas prácticas en la implementación de programas Java en general y concurrentes en particular. En concreto, se considerará especialmente:

- El nombre de las variables, parámetros, atributos, clases y métodos deberá reflejar lo más fielmente posible el uso que se va a realizar del elemento. Por ejemplo, no tiene sentido que un parámetro se llame “param1”.
- Siempre que sea posible, en vez de crear un nuevo hilo con un tiempo de vida corto, es mejor reutilizar hilos con un pool de Threads.
- Cuando sea necesario implementar algún tipo de sincronización o comunicación entre hilos, se deberá usar la herramienta de más alto nivel que permitan implementar los requisitos solicitados.

En la defensa presencial se deberán argumentar todas las decisiones que se han tomado durante la realización de la práctica. Pese a que se puede implementar en parejas, ambos integrantes tienen que poder contestar a cualquier pregunta sobre el código.

**IMPORTANTE:** Todas las prácticas que sean "razonablemente parecidas" pueden considerarse como copias y ambas prácticas serán consideradas suspensas. El enunciado es lo suficientemente abierto como para que parejas trabajando de forma independiente no puedan implementar prácticas parecidas sin haberse ayudado "demasiado" entre ellas.