**PROGRAMACIÓN CONCURRENTE**

PRÁCTICA 6

Adrián Rabadán Jurado

Teresa Rodríguez Ferreira

**Ejercicio 1:**

En este apartado creamos la clase Cliente, que se inscribe al servidor cada vez que se crea uno nuevo. De esta manera podrá comunicarse con todos los demás clientes. El método difundir permite al cliente enviar un mensaje y que sea leído por todos los demás, ya que el servidor contiene una lista de todos los clientes. Hemos creado una lista de hasta 100 clientes porque en los demás apartados podríamos ejecutar tantos clientes como queramos. Cada cliente ve el mensaje que ha enviado y todos los mensajes de los demás clientes, por tanto un mensaje enviado por el propio cliente aparecerá dos veces. Utilizamos el método callback para imprimir los mensajes. La ejecución en este apartado se hará creando a mano los clientes.

**Ejercicio2:**

En el segundo ejercicio se nos pide utilizar un contador para numerar los mensajes. Este contador lo guardamos en la clase Server, y cuando llamamos a callback en el método difundir, le pasamos este número para que se imprima por pantalla. Además creamos de forma automática 5 clientes, que enviarán 100 mensajes cada uno. Esto lo hacemos con un for en la propia clase Server, por lo tanto todos los mensajes se mostrarán en la consola del servidor. En este ejercicio no tenemos en cuenta la concurrencia, por tanto se pueden modificar los valores de forma incorrecta. Antes de difundir su mensaje, cada cliente hará un sleep de un tiempo aleatorio. Si vamos disminuyendo este tiempo se observan cada vez más valores incorrectos o desordenados.

**Ejercicio3:**

En este apartado lo único que hemos tenido que hacer ha sido añadir la palabra reservada synchronized en los métodos de difundir e inscribirse para controlar el acceso concurrente.

**Ejercicio4:**

Para cambiar el funcionamiento interno del programa utilizamos clases del paquete java.util.concurrent y quitamos los synchronized. Seguimos todas las indicaciones que aparecen, simulamos el callback con comunicación asíncrona gracias a un nuevo hilo que solo se encarga de realizar esa operación, suistituímos el contador de mensajes en el servidor por un AtomicInteger para evitar los bloqueos, cambiamos la lista de clientes por un CopyOnWriteArrayList para permitir accesos concurrentes, creamos otro hilo auxiliar que se encarga de realizar la operación de difundir, y empleamos dos ConcurrentLinkedQueue para gestionar los mensajes que llegan al cliente, almacenándolos en una cola de pendientes y otra de listos. También usamos dos variables para asegurar que las introducimos en orden en la cola de listos. Hemos separado también la función de escribir (por pantalla) en otro hilo auxiliar para que no haya interferencias. En este caso los clientes se crean a mano. Utilizamos un sleep simplemente para dar más tiempo a ejecutar los clientes al principio.

**Ejercicio5:**

Apartado opcional.

Aquí se nos pedía guardar todos los mensajes enviados entre los distintos clientes en un log. Nosotros hemos creado un log que es esencialmente igual que un cliente, pero que escribe los mensajes en un fichero en vez de por consola. Al igual que el cliente utiliza un hilo auxiliar para mostrar sus mensajes, el log emplea uno que simplemente escribe los mensajes en el fichero. El log se crea automáticamente en el servidor antes que todos los demás clientes, para así poder tener constancia de la “conversación” desde el principio. Los clientes se crearán manualmente.