README

> Si vas justo de tiempo, los textos que tengan este formato los puedes ignorar

En el código también hay comentarios pero muy simplificados.

Puedes simplemente consultar este documento en caso de no entender cierto método o función del código. Si tienes tiempo o curiosidad, puedes leerlo en su totalidad.

Primero se inicia el MainTeatro.java y luego el Main.java.

Main

El ejercicio te pide que necesitas un ordenador y un teclado para poder pedir tickets. Así que he creado dos arrays de Locks que sean diferentes para que llame aleatoriamente a uno del array de los teclados y luego a otro del array de los ordenadores o viceversa.

No hay mucho más que explicar de esta clase, creas los dos arrays de Locks, los rellenas, creas el array de Filósofos, los rellenas y pasas por parámetro los dos arrays y luego inicias Filósofos

MainTeatro

Solo sirve para crear una instancia de Teatro e iniciarla.

java.util.Random

Esta clase de Java me permite instanciar un objeto y pedirle tanto booleanos como números enteros o de punto flotante.

Filósofos

Filósofos tiene muchos atributos pero todos tienen nombres descriptivos para su entendimiento

Filósofos no necesita tener ningún en este caso synchronized por dos motivos:

1. No hay nada que necesite ser protegido de forma atómica
2. Un filósofo solo puede pedir entradas cuando ha podido lockear un ordenador y un teclado.
3. Daría completamente igual que sincronicemos algo ya que solo se sincronizaría con la propia instancia de la clase filósofos así que todos los filósofos seguirían haciendo lo mismo

Primer while

Según el enunciado, los filósofos pueden comprar hasta un máximo de 20 entradas. Así que este while se encargará de que este filósofo esté intentando comprar entradas hasta que Teatro se quede sin entradas o haya comprado el máximo de entradas posibles. Esta clase tiene un constructor adicional que permite modificar el número límite de entradas a comprar, pero por defecto es 20.

Primer if

El primer **if** sirve para elegir si empieza cogiendo un teclado o un ordenador. Este if pide un booleano aleatorio al objeto Random

Segundo y tercer if

Dependiendo de la dirección a la que nos haya enviado el primer if, leckearemos primero teclado y después ordenador o viceversa

1. En caso que no poder coger el primero, el filósofo se retira.
2. En caso de coger el primero pero no el segundo, soltamos el primero y el filósofo se retira
3. En caso de de coger el primero y el segundo ejecutamos el método pedirEntrada()

Método quedanEntradas()

Para esto tengo que explicar un poco Teatro. El ServerSocket de Teatro se mantendrá abierto siempre y cuando queden entradas por vender. En caso de que se hayan vendido todas las entradas se cerraría el servidor.

Este método prueba una conexión con el servidor para comprobar si sigue abierto teatro. Abre una conexión, escribe algo para que los ObjectInputStream del servidor no ejecuten una Exception y cierra tanto el flujo de salida como la conexión.

Si ha creado una conexión sin que salte ninguna excepción, significa que el Teatro sigue abierto y devuelve true. En caso de que Teatro esté cerrado, saldría por la excepción y devolvería falso

En la parte de Teatro, el mensaje que haya enviado este método lo leerá y lo ignorará

Método pedirEntrada()

Este método existe para no repetir código ya que da igual que dirección elija el primer if, vas a terminar ejecutando el mismo código.

La idea es que este método solo se puede ejecutar en caso de que tengamos lockeados un teclado y un ordenador

Primer for

Este for se ejecuta el mismo número de veces que entradas consecutivas quiero pedir. Por defecto, ese número es 10, como especifica el enunciado, aunque existe un constructor adicional que permite poner el número que se quiera.

Dentro hay una conexión que se abre y que al final del for se cierra. Es decir, que si repito el bucle for 10 veces abriría y cerraría 10 conexiones en total. Esto es así por un motivo muy lógico, un filósofo que vaya a comprar 10 entradas no acapara todo el servidor para él solo hasta que termine de coger las 10 entradas. Es decir, otro filósofo con otro ordenador y otro teclado podría pedir tickets simultáneamente junto con el anterior filósofo (Pero jamás podrían comprar más de 50 entradas porque la parte crítica del Teatro está sincronizado, esto lo explicaremos más adelante). Al hacer esto, la excepción creada a partir de quedarnos sin entradas en la repetición 3 se resolvería más fácil y no necesitaríamos ejecutar más veces ese

for. Por eso pienso que la naturaleza de Socket en este ámbito es el de abrir una petición rápida al servidor y pedir un ticket cada vez, y no la de mantener una conexión abierta todo el rato

Después de abrir la conexión, guardamos en variables el canal de salida y el canal de entrada de la conexión para usarlos más adelante. Estos también lo cerraremos al final del bucle for. Enviamos el mensaje "Una entrada, por favor." (En la parte del Teatro se recoge este mensaje y se compara con un mensaje de verificación para comprobar las intenciones del cliente. Lo explicaremos más adelante en Teatro). Leemos el mensaje que nos envía Teatro y lo guardamos en una variable (Este mensaje contiene el número de ticket y algo de texto en caso de que queden entradas. En caso negativo, recibiremos un "No").

Primer if

Este if verifica que la longitud de la cadena String que hemos recibido sea mayor que 2 ya que en caso de que no queden entradas, Teatro nos enviaría "No". Si hemos recibido más de 2 caracteres significa que hay más entradas y ejecutamos el if.

El if imprime por pantalla el nombre del filósofo y el número de ticket que posee. Después envía al servidor el nombre del Filósofo que ha comprado la entrada para que Teatro pueda imprimir quién es el Filósofo que la ha comprado y aumento el contador de entradas que tiene mi filósofo en uno.

close();

Al final de cada bucle for cierro tanto los flujos de entrada y salida como la conexión.

Teatro

Teatro tiene un atributo ServerSocket que se inicializa cuando creamos una instancia de Teatro. Es decir, si hacemos un new Teatro estaríamos también creando un new ServerSocket en uno de sus atributos. Como apunte, este ServerSocket se mantendrá abierto mientras tenga entradas para vender, en caso contrario se cerraría el ServerSocket.

> Otro atributo es un array de int llamado *numeroEntrada* en el que si se observa más detenidamente el código no sirve para absolutamente casi nada, solo sirve para tener una longitud y determinar el número de tickets a vender. De hecho, el array ni se rellena de números. Esto es así porque la idea original era crear también un objeto Ticket que tenga atributos como su nombre, el nombre del concierto o evento, el número

> de butaca, fecha del evento, etc. Por falta de tiempo y para no complicar más el código, se ha quedado como está.

Un atributo que sí le vamos a dar importancia es *textoConf* ya que este es el String al que vamos a comparar el mensaje que recibiremos para comprobar si quieren comprar una entrada o no. Esto es así porque recordemos que para comprobar si nuestro servidor sigue abierto hacíamos una conexión, enviábamos algo fuera de lugar y cerrábamos directamente la conexión en la parte del Filósofo (mirar método `quedanEntradas()`).

> Podríamos haber hecho una función en Teatro que nos devolviese un booleano indicando si hay más entradas o no, pero para acceder a ese método desde Filósofo solo causaría problemas ya que para acceder a ese método tendrías que pasar por parámetro a Filósofo la instancia de Teatro. Como no queremos eso porque no tendría sentido, tendríamos que haber vuelto a la función estática pero al volverla estática no podríamos acceder a los atributos de la clase que no sean estáticas así que tendríamos que hacer al atributo ServerSocket estático pero entonces todas las instancias de Teatro compartirían el servidor y si una de esas instancias cerrase el servidor, todas las demás instancias de Teatro se quedarían sin servidor. En conclusión, solo daría problemas.

Primer while

Este while se repetirá mientras que el índice que recorre el array (que es un atributo de la clase que empieza a 0) sea menor que la cantidad de entradas que tenemos para vender.

> Hay que tener en cuenta que los arrays empiezan desde cero, así que terminará en 49, así que significa de cuando llegue a 50 ya no se ejecutará.

Hacemos un `servidor.accept()` y recogemos los que nos devuelva en una variable Socket.

Abrimos solamente un flujo de entrada

Esta variable Socket y ObjectInputStream la cerramos después del Primer if, da igual que el if se ejecute o no.

Primer if

ESTE IF ES MUY IMPORTANTE, porque determinará si lo que estamos recibiendo es una petición de ticket de un Filósofo o la comprobación de que el servidor sigue abierto.

> Sin este if, venderíamos tickets a nadie, porque no distinguiríamos si estamos recibiendo una petición de ticket de un Filósofo o la comprobación de que el servidor sigue abierto. Además, dentro de este if hay más lecturas y escrituras entre Teatro y Filósofo y recordemos que en el método `quedanEntradas()` enviamos un mensaje y nada más, por lo que en Teatro simplemente tendríamos errores y saltarían excepciones dentro de este if porque iríamos a leer cosas que nadie nunca nos mandará.

Entraremos en este if si el mensaje que nos ha enviado el Filósofo es el mismo que se usa para la petición de un ticket (Recordemos que el Filósofo envía el servidor "Una entrada, por favor" en caso de querer una entrada). Esto lo logramos comparando el mensaje que nos han enviado con el atributo de clase Teatro *textoConf*.

Si la comparación nos devuelve false, simplemente cerramos el flujo de entrada de datos y cerramos el Socket de cliente que nos hay devuelto el `servidor.accept()`

Si la comparación nos devuelve true, entraríamos en el if:

LO QUE ESTÁ DENTRO DE ESTE IF ESTÁ SINCRONIZADO. Lo sincronizamos con esta misma instancia (Es decir, `this`). Esto es porque esta parte del código es crítica, ya que vamos a vender una entrada y en caso de no tenerlo sincronizado puede darse el caso de vender más entradas de las que tenemos ya que las podríamos estar vendiendo en este caso hasta de 3 en 3 (Porque hay 3 teclados y 3 ordenadores, así que puede haber 3 filósofos comprando compulsivamente). Al sincronizarlo, solo un filósofo podrá ejecutar este código a la vez para comprar una entrada.

Segundo if

Este if está dentro del `synchronized(this)` , así que está hecho para comprobar de nuevo que el índice del array es menor que la longitud del array ya que puede darse el caso de que tres filósofos a la vez hayan llegado hasta aquí pasando por el while en algún momento en el que solo quede una entrada. Es un caso muy hipotético, pero puede ocurrir.

Dentro de este if creamos un flujo de salida para pasarle al filósofo el número de ticket y algo de texto y nosotros recibiremos el nombre del Filósofo junto con algo de texto para poder imprimir en el Teatro.

Aumentamos el índice en uno, es decir, acabamos de vender una entrada.

Imprimimos cuantas entradas quedan por vender.

En caso de no entrar en este if porque ya nos hemos quedado sin entradas, el servidor enviaría al cliente "No" (Recordemos en método `pedirEntrada()` / primer if de Filósofos, el filósofo está preparado para en caso de recibir un "No", dejar de enviar información a Teatro y cerrar sus conexiones)

En cualquiera de las dos rutas de este if, cerraremos el flujo de salida con `salida.close()` y esperaremos 1 segundo para la siguiente petición ya que así lo pedía el enunciado.

close()

Una vez se haya salido del while, significa que nos hemos quedado sin más entradas que vender. Así que cerramos en servidor con `servidor.close()` .