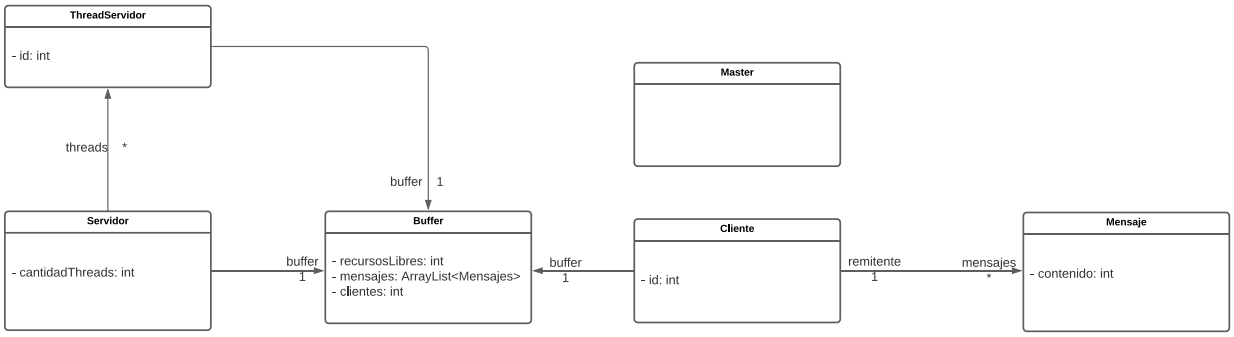
Nicolás Fajardo 201729477
Daniel Babativa 201729103

Caso 1. Infraestructura computacional, concurrencia

División de clases:

El problema se modeló de tal manera en la que se pudo dividir el mundo en 6 clases:

- La clase Master: se encarga de correr todo el programa y cargar los datos del archivo y pasarlos como parámetros a las demás subclases.
- Cliente: juega el rol de un cliente en nuestro caso de estudio, el cual se encarga de crear los mensajes e irlos mandando al buffer a medida que estos son respondidos. Esta clase se modela como un Thread de Java.
- Buffer: es el intermediario entre cliente y ThreadServidor quien almacena los mensajes mientras el cliente espera respuesta y el ThreadServidor se libera para darle respuesta al mensaje almacenado aquí.
- Servidor: es la clase que se encarga de manejar y crear los ThreadServidor, representa al servidor en el problema y hace que los ThreadServidor encargados de responder mensajes empiecen a trabajar, esta clase se modela como un Thread de Java.
- ThreadServidor: vendrían a ser quienes que se encargan de retirar los mensajes del buffer y darles respuesta. Esta clase también se modela como un Thread de Java.
- Mensaje: modela el mensaje que el Cliente crea y manda al Buffer para que un ThreadServidor del Servidor le de respuesta.



Carga de archivos:

Los archivos se cargan en la carpeta Caso 1 desde un archivo llamado datos.txt. Este archivo está compuesto por 4 líneas de texto donde cada una puede solamente tener un valor entero, y una breve explicación de estas. Estas líneas representan en orden, número de clientes, número de threads del servidor, tamaño del buffer y número de mensajes por cliente.

Funcionamiento del programa:

El programa empieza desde la clase Master, esta se encarga de leer el archivo según las especificaciones dadas en la sección de “Carga de archivos”. Estos datos son guardados y enviados a las respectivas clases en su momento de creación. Se crea el Buffer, luego los Clientes y Finalmente el Servidor. Después de crear todos estos objetos se ejecuta el método start() de la clase Servidor y luego para cada uno de los Clientes.

Nicolás Fajardo 201729477
Daniel Babativa 201729103

Por parte del Servidor al ejecutarse el método start() este pone a todos sus threads de la clase ThreadServidor a ejecutar el método start(). Estos, mientras aún haya clientes que sigan teniendo mensajes por mandar le pedirán al buffer un mensaje ejecutando sobre el objeto Buffer el método retirarMensaje(). Este, en caso de estar vacío hace que el thread ejecute el método yield() para liberar procesador mientras espera a que lleguen mensajes. En caso de que no esté vacío, el thread ejecuta el método responderMensaje() del mensaje que retorna el método retirarMensaje().

El Cliente, al ejecutarse su método start() tendrá las siguientes funciones, mandar mensajes al Buffer y esperar a que se respondan. De esta manera entonces mientras aun tenga mensajes el Cliente va a tomar el primero y se lo va a entregar al Buffer, si este no está vacío lo recibe y lo almacena. En caso de estar lleno en el método del Buffer se pone a esperar al thread del Cliente con un wait(). Si se envía el mensaje al Buffer, el cliente ejecuta el método enviarMensaje() del mensajeActual para así entrar a un wait() mientras espera a ser notificado cuando su mensaje haya sido respondido. Al terminar de enviar todos los mensajes el Cliente notifica al Buffer que se va y ha terminado.

Dentro del Buffer tenemos dos métodos (sincronizados) principales. El primero, enviarMensaje(), se encarga de recibir un mensajes el Cliente, en caso de tener espacio lo acepta y lo guarda en un array con los mensajes, en caso de que no haya espacio pone el thread del Cliente a dormir y lo notifica en el método retirarMensaje(). Este método lo que hace es que le retorna el primer mensaje en el Buffer al ThreadServidor que lo ejecute, en ese momento se revisa si los nuevos recursos disponibles son iguales a 1 y si es el caso notifica a un thread de Cliente que durmió con anterioridad y retorna el mensaje, si no es el caso, simplemente retorna el mensaje sin notificar a nadie.

La concurrencia se evidencia en varias interacciones de nuestras clases. El primer ejemplo es entre las clases Cliente-Mensaje-ThreadServidor. El Cliente se queda en un wait() cuando ejecuta el método enviarMensaje() de las clase Mensaje. Esto se hace para que entonces el cliente espere a que el Mensaje sea respondido por un ThreadServidor. Así mismo como se dice entonces, una vez el ThreadServidor ejecute el método responderMensaje() de la clase Mensaje este le notifica al Cliente que ya se respondió su mensaje y que puede continuar con su ejecución.

El segundo ejemplo o utilización de concurrencia en el desarrollo del caso es entre las clases Cliente-Buffer-ThreadServidor. La clase Buffer como se mencionó previamente tiene dos métodos sincronizados, esto se hizo así para que los métodos sean atómicos y no haya interrupciones en estos. El flujo puede iniciar nuevamente con Cliente, este manda un Mensaje al Buffer, en este momento hay dos opciones, el Buffer puede o no puede estar lleno, en caso de no estarlo el método se ejecuta con normalidad, en caso de estar lleno entonces el thread Cliente se pone a dormir con el método wait(). Este espera hasta que un ThreadServidor retire un mensaje y le notifique al thread Cliente que ya hay espacio para su mensajes, esto ocurre dentro del método retirarMensaje() de la clase Buffer y solo notifica al thread Cliente si el número de recursos libres después de retirar el mensaje es igual a 1. Por el lado del objeto ThreadCliente este ejecuta el método retirarMensajes() (de la clase Buffer) el cual nuevamente tiene dos opciones, el Buffer está vacío o tiene mensajes, en caso de tener mensajes se ejecuta el método con normalidad y si se cumple la condición mencionada previamente descrita notifica a un thread Cliente dormido, en caso de no tener ningún

mensaje para retirar el buffer hace pasar el ThreadServidor de En Ejecución a Listo para nuevamente intentar entrar a retirar un mensaje.

Información adicional del caso:

Dentro del código se dejaron varias impresiones en consola de forma intencional, para que cuando se ejecute el código quede rastro y se pueda visualizar (a pesar de que es muy rápido) el hilo de como funciona el programa. Estas impresiones en consolas tienen ciertos estándares de nombramiento que deben ser descritos para mayor comprensión. Los Clientes tienen un ID que corresponde al número de mensajes que puede enviar, es decir que si un cliente tiene un id de 5, se quiere decir que el cliente 5 envió el mensaje, pero no es el cliente 5 por su orden de creación sino porque puede enviar 5 mensajes (nos aseguramos de que dos clientes no tengan el mismo id al multiplicar un valor constante del archivo datos.txt por el número del cliente en ser creado). Los mensajes se identifican a través de su remitente y de su número de creación, es decir el mensaje "El mensaje 3.2 ha sido enviado" quiere decir que el tercer mensaje del cliente 3 ha sido enviado. Finalmente, los ThreadServidor se identifican por su orden de creación, entonces un id de 0 se refiere a que fue el primer thread en ser creado.