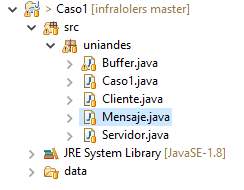
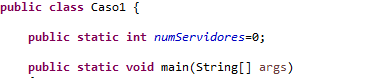
**Diseño del programa**

**Contexto de las clases:**

Para el desarrollo del caso 1 se pensó principalmente en el uso de 4 clases para modelar el problema propuesto. La primera clase es una a la que llamamos caso1, la cual se encarga de ejecutar un método *main* y crear unos objetos estáticos que usara para simular al buffer y al numero de servidores que se va a usar. Luego, están la clase Servidor y la clase Cliente las cuales están pensadas para simular a estos actores; ambas extienden de la clase *thread* y será usados para procesar los procesos del problema. En último lugar, están las clases Mensaje que su función será la de contener el mensaje y guardar el contenido de este, y la clase buffer que será la que va a coordinar la comunicación de mensajes concurrentemente entre el cliente y el servidor. En la clase Buffer estarán los métodos de colocar y responder, de modo que sean usados por los clientes y los servidores, para manejar la transmisión de mensajes.



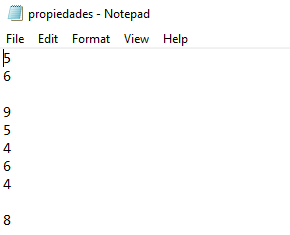
Estructura de la clase Caso1:



**Funcionamiento:**

**Clase que ejecuta el programa y crea las instancias Caso1:**

El programa comienza cuando se ejecuta el método *main* de la clase Caso 1, en el cual se hace la lectura de un archivo de texto plano donde se encuentran números que representan los mensajes.

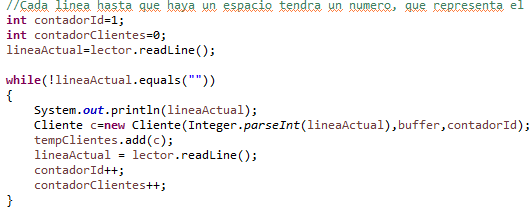


La primera línea indica el número de clientes esperados, La segunda línea indica la capacidad del buffer y después del espacio los números representan la cantidad de consultas que van a tener los clientes. El ultimo numero es el numero de servidores que se van a crear.

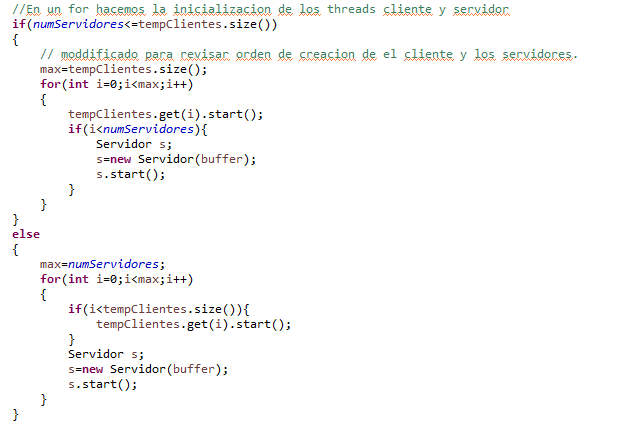
Inicialmente se inicializan todas las variables del programa con los datos sacados del archivo plano. Primero se inicializa el buffer con la cantidad de espacios que va a manejar, seguidamente creamos los clientes con el numero de consultas que quiere hacer y los guardamos en una arraylist para inicializarlos después.



Siguiente



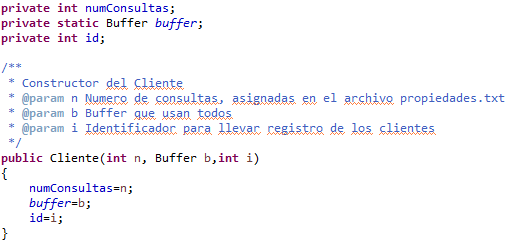
Finalmente, para optimizar el arranque de los clientes y servidores al probar el programa tuvimos en cuenta el que fuera mayor de los dos para crearlos y correrlos dentro de un solo ciclo. Siempre tuvimos en cuenta el crear primero a los clientes y luego a un servidor, en este orden para un servidor tuviera por lo menos el trabajo de un cliente cuando arrancara.



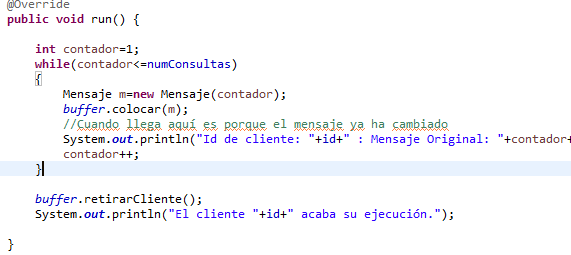
**Funciones del Cliente y Servidor:**

**Nota:** tanto el cliente como el servidor extienden de la clase thread para que funcionen como hilos de ejecución intentando usar el procesador concurrentemente.

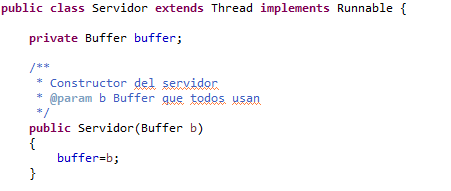
El cliente por su lado tiene un atributo estático que representa el buffer que comparte con los otros clientes y los otros servidores, un atributo que representa el numero de consultas que va a realizar y un atributo para representar el id del hilo.



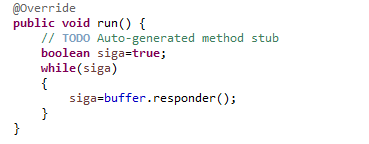
En el método run se encuentra el uso del buffer para poder enviar el mensaje y esperar la repuesta, se inicia con un contador en 1 que va ser el contenido del mensaje y se crea el mensaje para agregarlo usando el método del buffer, seguido a esto se muestra en consola como quedo el mensaje después de que ya lo pudo colocar. Cuando ya termina el método de colocar el mensaje es porque ya el servidor lo ha modificado entonces se muestra esto y se vuelve a intentar con el numero siguiente del contador enviando un nuevo mensaje. Cuando el contador es igual al numero de consultas se acaba el run y se le avisa con otro método al buffer para sacar al cliente y que no lo considere más.



Para el servidor, se tiene un solo atributo que seria el buffer el cual será el comunicador para poder procesar los mensajes que estén en espera.

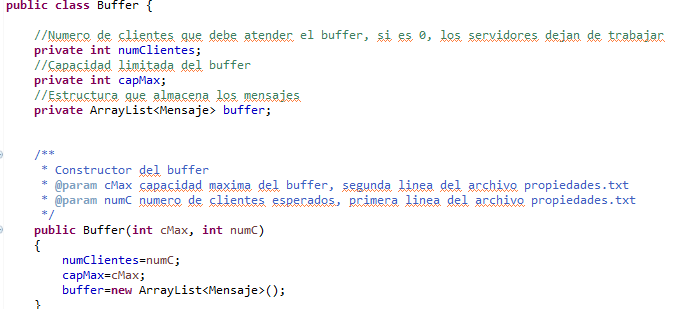


En el método run se maneja con un ciclo que se ejecuta hasta que el método de sacar los mensajes del buffer y este se mantienen hasta que el método de sacar le avisa al servidor que ya no hay mas peticiones. Finalmente, se termina el método usando el avisar para que el buffer ya no considere más a este servidor.

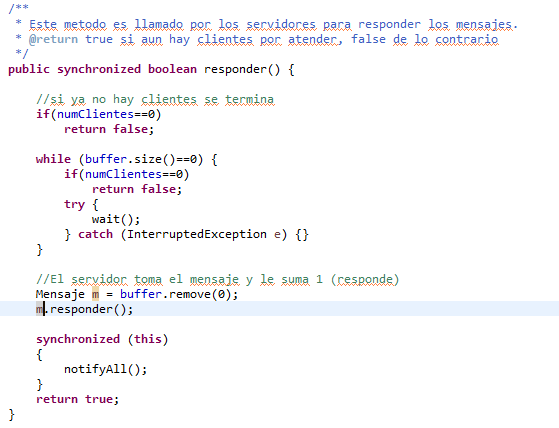


**El Buffer la clase que permite toda la concurrencia de los threads:**

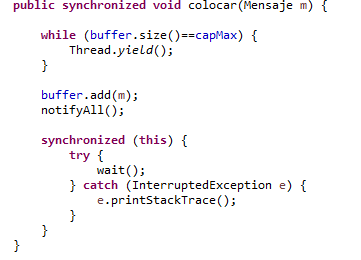
Los atributos de esta clase son principalmente el numero de clientes, la capacidad máxima de mensajes que puede almacenar el buffer y un ArrayList para los mensajes del buffer.



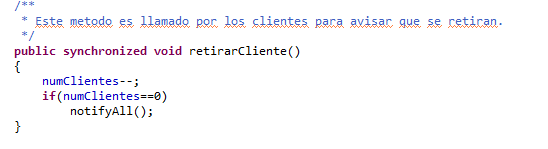
Primero se tiene el método que va a usar los servidores para poder procesar los mensajes es un método sincronizado en caso de que exista varios servidores que vayan a utilizarlo. Se valida que haya clientes esperando y luego mientras no haya mensajes se hace una espera con el método wait si no han llegad mensajes. Cuando hay mensajes se quita el mensaje del buffer y se cambia con un método interno de la clase de este. Finalmente, después de alterar el mensaje, se les notifica sincronizadamente a los clientes que revisen si su mensaje ya esta modificado con el notifyall.



Para el método colocar mensaje en el buffer de los clientes, primero se valida que se pueda ingresar el mensaje que se quiere encolar, si no es posible se hace la espera con el método yield. Luego, se agrega el mensaje al buffer cuando el yield o permite y se notifica a todos los que están esperando para que sigan con sus tareas una vez que se alteró el buffer. Luego se pone al método en espera sincronizadamente.



Finalmente está el método de retirar los clientes el cual hace esta tarea disminuyendo en 1 el numero de clientes en el buffer y notificándole a los servidores si ya no hay clientes para que paren.



Todos los métodos del buffer llamados en conjunto respecto a la ejecución de los threads hacen concurrente el sistema y los sysouts sirven para evidenciar el funcionamiento.

