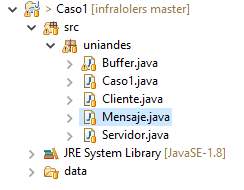
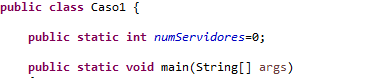
**Diseño del programa**

**Contexto de las clases:**

Para el desarrollo del caso 1 se pensó en usar 5 clases para modelar el problema propuesto. La primera clase es Caso1, la cual se encarga de ejecutar un método *main* de acuerdo a un conjunto de propiedades descritas en el archivo data/propiedades.txt. Luego, están la clase Servidor y la clase Cliente las cuales están pensadas para simular a los actores; ambas extienden de la clase *thread* y ejecutarán los procesos del problema. En último lugar, están las clases Mensaje, cuyas instancias serán creadas por los clientes, y serán respondidos por los servidores en el buffer, y finalmente, la clase Buffer, que será la que va a ser el intermediario entre el cliente y el servidor. En el Buffer, los clientes ingresan mensajes (método colocar() ) y los servidores los responden (método responder() ).



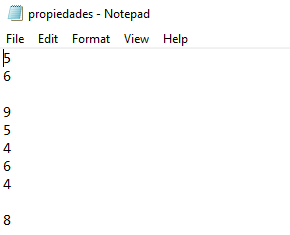
Estructura de la clase Caso1:



**Funcionamiento:**

**Caso1, clase que ejecuta el programa:**

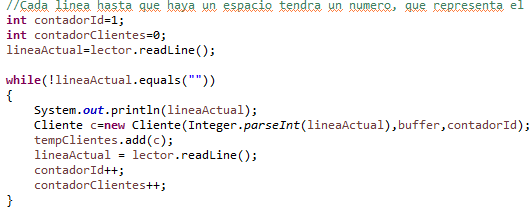
El programa comienza cuando se ejecuta el método *main* de la clase Caso 1, en el cual se hace la lectura de un archivo de texto plano donde se encuentran números que representan las propiedades.



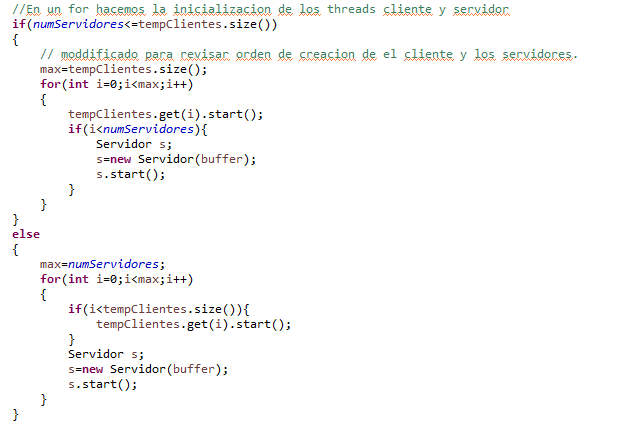
La primera línea indica el número de clientes esperados, La segunda línea indica la capacidad del buffer y después del espacio los números representan la cantidad de consultas que tendrá cada cliente. Por ejemplo, en la imagen el primer cliente hará 9 consultas, el segundo 5, y así sucesivamente. El último número, después del espacio, representa los 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 un ArrayList para ejecutar el método start() después.





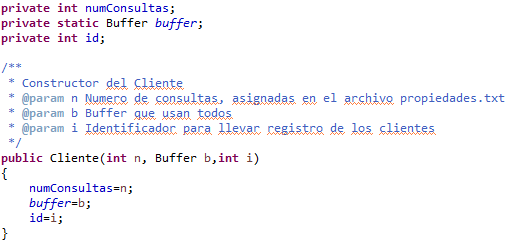
Finalmente, para optimizar el arranque de los clientes y servidores al probar el programa tuvimos en cuenta el que fuera mayor de los dos para crearlos y correrlos dentro de un solo ciclo.



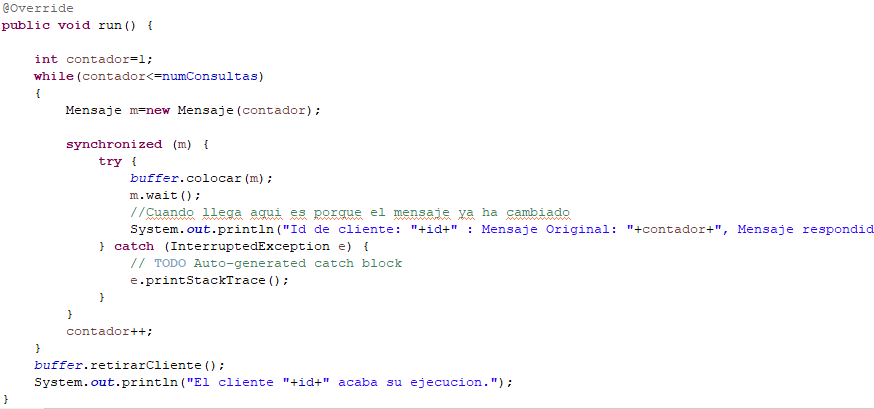
**Cliente y Servidor:**

Tanto el cliente como el servidor extienden de la clase thread para que funcionen como hilos de ejecución.

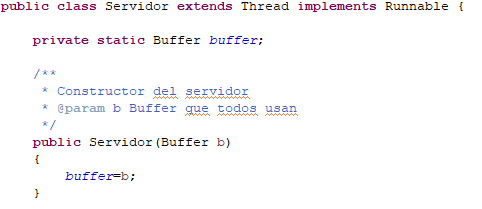
El cliente tiene un atributo estático que representa el buffer que comparte con los otros clientes y los servidores, un atributo que representa el número de consultas que va a realizar y un atributo para representar el id del hilo, el cual facilita la revisión del funcionamiento del programa mientras se imprimen los mensajes en consola.



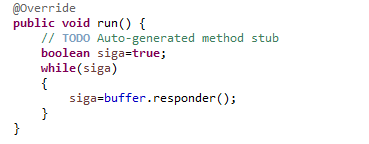
En el método run se encuentra el uso del buffer para poder enviar el mensaje y esperar la repuesta, se inicia con un contador en 1, el cual será el contenido del mensaje y se crea el objeto mensaje para agregarlo usando el método del buffer. Se ejecuta el método wait() sobre el mensaje para que cuando un servidor ejecute un notify() sobre dicho mensaje, el cliente pueda continuar. Una vez que la espera ha terminado, se imprime en consola el mensaje original (representado por el valor del contador) y el valor actual del mensaje (que ha sido respondido por algún servidor y se espera que su valor haya sido incrementado). El contador de consultas se incrementa en 1 para continuar con sus tareas. Cuando el contador es igual al número de consultas, se dejan de realizar y se ejecuta otro método del buffer, retirarCliente(), para avisarle que el cliente se retira.



El servidor tiene un solo atributo, buffer, el cual será el intermediario para poder procesar los mensajes que estén en espera.

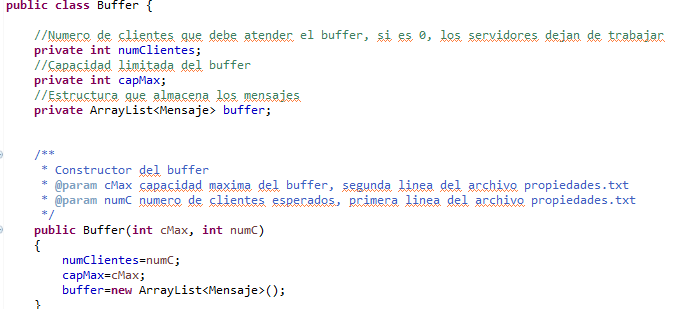


En el método run se maneja con un ciclo que se ejecuta hasta que el método de sacar los mensajes del buffer indique que ya no hay clientes. Más información de este método se encuentra en la sección Buffer.

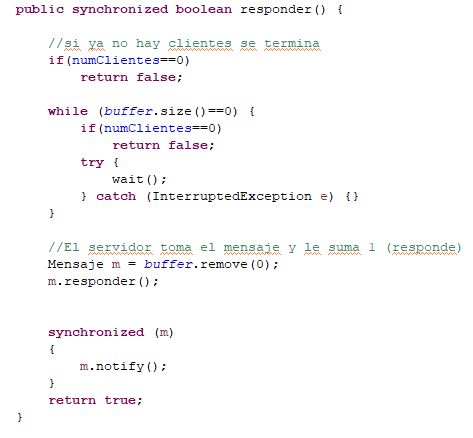


**Buffer:**

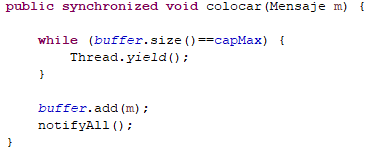
Los atributos de esta clase son el número de clientes, la capacidad máxima de mensajes que puede almacenar el buffer y un ArrayList para los mensajes del buffer.



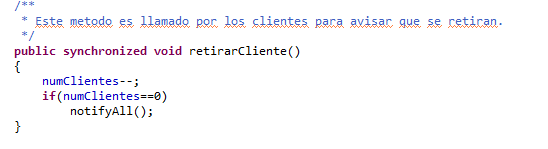
Primero se tiene el método que van a usar los servidores para procesar los mensajes. Es un método sincronizado, puesto que varios servidores intentarán utilizarlo de manera simultánea. Se valida que haya clientes esperando y luego mientras no haya mensajes se hace una espera con el método wait. Cuando hay mensajes se quita el mensaje del buffer y se “responde” con el método responder() de la clase Mensaje. Finalmente, después de alterar el mensaje, se hace un notify sobre el objeto mensaje para que el Cliente que estaba esperando sobre éste pueda continuar su ejecución.



Para el método colocar, el cual será utilizado por los Clientes, primero se valida que se pueda ingresar el mensaje que se quiere encolar, revisando la capacidad actual con la capacidad máxima, y si no es posible se hace la espera con el método yield. Se agrega el mensaje al buffer cuando la capacidad es menor a la capacidad máxima y se ejecuta un notifyAll para que los servidores puedan continuar sus tareas.



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 puedan finalizar su ejecución.



Todos los métodos del buffer llamados en conjunto respecto a la ejecución de los threads hacen concurrente el sistema y los mensajes en consola sirven para evidenciar el funcionamiento. Se muestra un ejemplo de los mensajes impresos con la configuración mostrada en la página 2 del documento.

