**Caso 1: Infraestructura computacional**

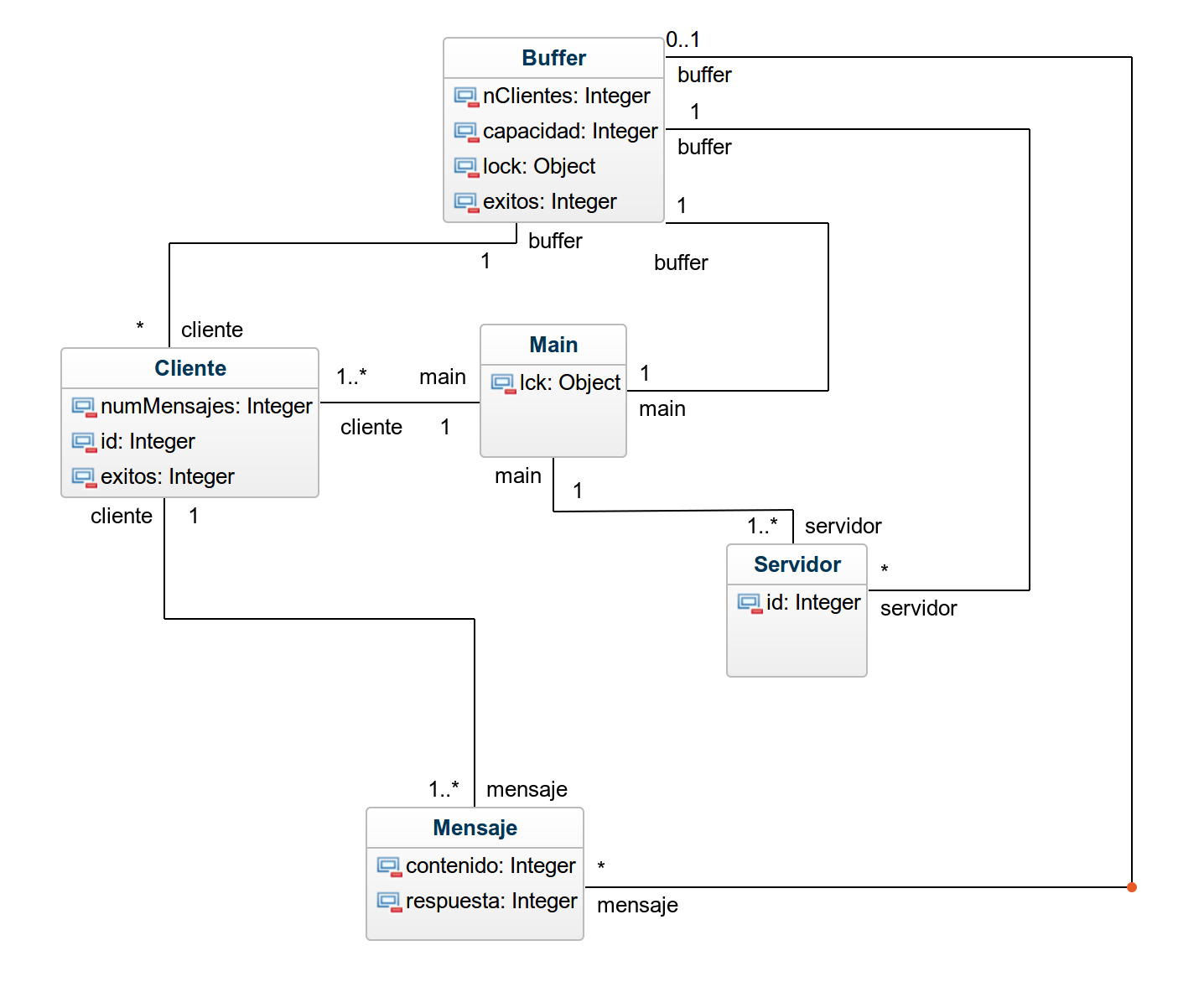
Mauricio Neira Giraldo - 201424001

Daniel Agudelo - 201533411

Diseño del programa:

Como el enunciado sugiere, se tienen 4 clases, que son las principales: Cliente, Servidor, Mensaje, Buffer. Los Clientes tienen querys por resolver (Mensajes por ser respondidos), y están activamente mandándolas al Buffer. Por su parte, los Servidores están pasivamente intentando resolver los querys existentes (respondiendo los mensajes) en el Buffer. Éste último debe funcionar como un Monitor, es decir, el acceso a él no es concurrente, ya que al momento que un Cliente o un Servidor accede a él, no puede ser interrumpido, porque se pueden generar inconsistencias.

Las clases mencionadas siguen el siguiente diagrama UML:



Funcionamiento:

Al inicio del programa se obtiene la información de cuántos Servidores y Clientes hay, además de cuántos mensajes enviará cada Cliente, y de la capacidad del Buffer. Acto seguido, se crea el Buffer, los threads Servidores, threads Clientes y los mensajes correspondientes a cada cliente.

Los Clientes se crean y en consecuencia se crea una lista de Mensajes que quieren llevar al Buffer para obtener respuesta. Estos mensajes no son Threads por lo que simplemente se crean con un contenido aleatorio y un campo de respuesta vacío. Cuando inicia el Thread Cliente, intenta enviar los Mensajes que tiene el Buffer activamente, para ello, llama un método sincronizado de la clase Buffer llamado agregarMensaje, que recibe el mensaje y devuelve el booleano que indica si fue posible agregarlo al Buffer o no. Cuando recibe *true*, el Thread Cliente se duerme sobre el objeto del mensaje (mensaje.wait()) y se pone en espera de que un Servidor lo despierte cuando el mensaje sea respondido. Adicionalmente, dado que en este caso la inserción es exitosa, invoca un notifyAll() sobre el Buffer para avisar a los Servidores que están en espera de mensajes por atender, que ya pueden intentarlo. Cuando recibe *false*, llama el método yield(), que hace que todos los demás Threads Cliente que estaban esperando para intentar, lo hagan antes de que éste lo vuelva a hacer.

Al momento de recibir una respuesta (que un Servidor ejecute un notify() sobre el mensaje), verifica que la respuesta sea correcta, es decir, que sea el contenido aumentado por uno, y continúa intentando enviar los mensajes que le queden sin responder, repitiendo lo anteriormente descrito..

Por otro lado, los Threads Servidor se crean y empiezan a intentar sacar mensajes de la estructura que los almacena en el buffer por medio de un método sincronizado de esa clase llamado removerMensaje. Si obtienen *null* , es porque no hay mensajes por atender en el Buffer y entonces se dormirán en el Buffer esperando que un Cliente logre depositar exitosamente un mensaje en el mismo y que los despierte en consecuencia. Si obtienen un Mensaje distinto de null, lo procesan (asignan en el campo respuesta el valor del contenido aumentado) y ejecutan sobre él un notify() que despertará al Cliente que estaba esperando la respuesta.

Los servidores, continúan intentando responder mensajes siguiendo el proceso anteriormente descrito hasta que el Buffer acabe, es decir, que todos los clientes hayan sido atendidos.

Cuando el buffer acaba, notifica al Thread principal (en el main) que puede continuar. Esto, debido a que cuando éste crea los Threads Clientes y Servidores, se pone a esperar sobre un Object llamado lock (lock.wait()) hasta que todos terminen, para continuar la ejecución y terminarla.

----------------------------