**Caso 1 – documentación**

Se explicará a continuación un breve resumen de la implementación de la solución, más explicación de las interacciones entre cada par de actores que interactúen entre sí.

La aplicación consta de 4 clases que representan a cada actor del enunciado: Buffer, Cliente, Mensaje y Servidor. La ejecución principal (*main*) se realiza sobre la clase Buffer, dentro, se leen los parámetros que entran por el archivo de texto. Finalmente, crea los clientes y los servidores con base a los parámetros y, por cada cliente, inicializa su ejecución. El diseño a gran escala es un problema de productor-consumidor con un canal de recursos. En este caso, los clientes son los productores (producen las consultas), los servidores son los consumidores y el buffer es el canal.

**Interacciones entre actores:**

Cliente --> Mensaje: El cliente es el encargado de inicializar los mensajes. El número de mensajes lo recibe como parámetro en el constructor y al momento de su ejecución, inicializa las consultas con un nuevo entero aleatorio. Por ahora, no se requiere ningún esfuerzo de sincronización porque no hay concurrencia.

Cliente --> Buffer: Después de inicializar los mensajes, el Cliente los envía al buffer. Entonces, se sincroniza el objeto buffer puesto que se va a acceder a una variable compartida, la cola / lista de mensajes. Si el buffer se encuentra en su capacidad máxima, significa que no puede consultar y debe entrar en espera activa. En este caso, como sugiere el enunciado, por cada intento se cede el thread que lo está ejecutando en el momento con el método *yield*(). Cuando pueda, finalmente lo agrega y hace *wait()* sobre el mensaje. Esto se hace con el fin de que cuando a ese objeto mensaje se le llame *notify(),* salga del método consultar y pueda activar un mecanismo de que el cliente ya acabó con toda su carga.

Servidor --> Buffer: El servidor, apenas es creado, crea sus threads y los corre inmediatamente. Cada thread se encarga de responder al buffer siempre que pueda, esto lo logra mediante sincronizar el buffer y revisar si está vacío o no. Si sí, el servidor queda en espera pasiva hasta que se le notifique que se agregó algo. Si no, saca el mensaje de la cola / lista del buffer y resuelve la petición asignándole la respuesta al mensaje.

Servidor --> Mensaje: Por último, cuando ya se le asigno una respuesta al mensaje, se despierta por medio de un *notify()* haciéndole saber que ya acabó la consulta.

Cuando todos los clientes ya han acabado su ejecución, se “apaga” el servidor y no queda ningún thread en ejecución / dormido.