**Proyecto 1**

Nicolás Munar - 201814383

Felipe Gustavo Guzman - 201813791

Mohamed Aris Bohora - 202016142

El proyecto uno de la clase Infraestructura computacional se resume en un cliente, un servidor y una clase intermedia, Buffer, para manejar la interacción entre estos dos elementos. El cliente genera mensajes que envía al buffer, este buffer espera a que un servidor reciba el mensaje lo procese y posteriormente lo devuelva al buffer para darle alguna respuesta al cliente que envió el mensaje.

Sin más esta es la explicación del código implementado en cada clase y su razón de estar ahí.

* **Clase Cliente**

La clase **Cliente** es un Thread por lo que tiene los métodos asociados a uno y se modela como tal.

**public** Cliente(String numCli, String mensaje, Buffer br1)

Empezando por el constructor se tienen tres parámetros, dos corresponden a el número de clientes que se quieren crear y al número de mensajes que se quieren enviar por cliente, el tercer parámetro enviado es el buffer el cual se maneja como variable compartida en el sentido en que desde la clase principal se envía como parámetro para cumplir con esto.

Continuando, cabe aclarar que se tienen dos variables estáticas que se inicializan con el constructor, estas corresponden a el numero de clientes y al buffer.

El cliente también tiene dos métodos adicionales muy importantes explicados a continuación;

**public** **void** run()

Este método crea un nuevo mensaje y llama al buffer por cada mensaje que desee enviar. La manera en que este hace el llamado al buffer es mediante el uso del método P(this) que tiene el buffer que implementa la espera activa es decir **yield** si el buffer se quedó sin capacidad.

**public** **void** sendMessage()

Este método se encuentra dentro del run() y está encargado de llamar al método sendmessage() del buffer para añadirlo a la cola de mensajes y así realizar la consulta de si puede o no enviar el mensaje o en su defecto poner a esperar el recurso

Es importante mencionar que en caso de que el buffer esté lleno este entra en espera activa con **yield()** lo que causa que el thread cede el procesador para que otros puedan hacer operaciones sobre él. En otro caso, si el mensaje es depositado este usa el método **wait()** para esperar pasivamente a que el servidor responda.

* **Clase Servidor**

La clase **Servidor** es un Thread por lo que tiene los métodos asociados a uno y se modela como tal.

**public** Servidor(String contador, Buffer pBr)

Empezando por el constructor de la clase tenemos un parámetro que corresponde a el numero del servidor y el buffer que se maneja como variable compartida en el sentido en que desde la clase principal se envía como parámetro para cumplir con esto.

**public** **void** run()

Este método imprime un mensaje notificando que el servidor empezó a funcinoar. Posteriormente hace un llamado al buffer y reserva el recurso mediante el uso del método P(this) que tiene el buffer que implementa la espera activa es decir **yield()** si el buffer se quedó sin capacidad.

Posteriormente se notifica mediante un mensaje que el recurso fue apartado y procede a obtener el mensaje de la lista de mensajes. Una vez realizado esto incrementa una variable que tiene la clase mensaje para indicar la cantidad de mensajes exitosos que fueron respondidos y llama al buffer para retornar el mensaje, es decir liberar el recurso **notify()**

* **Clase Mensaje**

Esta clase usa únicamente un constructor que asigna un valor aleatorio a un entero que será el mensaje que se enviará por el cliente

Una vez el cliente lo envíe y el servidor lo reciba este mensaje tendrá respuesta de uno más. Es decir si el cliente envía el mensaje 1 el servidor responderá 2.

* **Clase Principal**

Esta clase está encargada de recibir el archivo con el número de clientes, el numero de mensajes que cada uno va a enviar, el número de servidores que recibirán esa información y el tamaño del buffer.

Para esto cuenta con dos métodos. El método lectorArchivo(), startServers() y startClientes(), con los cuales se inicia el Buffer se inicializan los servidores y se inicializan los clientes.

* **Clase Buffer**

Esta clase maneja todas las operaciones del proyecto, pues es la que pone a esperar tanto activa como pasivamente a los threads y sirve de intermediario entre los dos.

La clase cuenta con tres atributos;

**private** **int** buffer\_count;

**private** **final** **int** totalMessage;

**private** ArrayList<Mensaje> listaMensajes;

Estos atributos sirven para manejar el flujo de mensajes que cada thread quiere usar. De esta manera se modela el buffer a modo de semáforo. Pues cuenta con una lista de mensajes que actúa como la cola de espera que tiene un semáforo.

Los métodos usados son los siguientes;

**public** **void** P(Cliente i)

**public** **synchronized** **void** V(Cliente i)

**public** **void** sendMessage(Mensaje m)

**public** **void** endCliente()

**public** Mensaje receiveMessage()

**public** **void** returnMessage(Mensaje m)

Los metodos P y V tanto del cliente como del servidor sirven para manejar la espera activa y tener en cuenta la capacidad del buffer. En este caso los métodos V cuentan con un **.yield()** que le da el carácter de esperar activamente y el método V cuenta con un contador para aumentar la capacidad del buffer.

Adicionalmente están los métodos send, recieve y return message los cuales hacen el papel de espera pasiva y/o notificar

**public** Mensaje receiveMessage()

Este metodo verifica que todos los clientes se hayan atendido. Si no se han atendido verifica que si hay mensajes en la lista de mensajes, si los hay remueve el primero y verifica si es nulo o no. En caso de ser nulo el Servidor lo retiene activamente y continua verificando los demás elementos, si no es nulo lo envía y opera internamente

**public** **void** returnMessage(Mensaje m)

Notifica que al thread que estaba esperando que su recurso está disponible

**public** **void** sendMessage(Mensaje m)

Este método envía el mensaje para que sea procesado en el Servidor, mientras que el buffer añade el mensaje a la lista de mensajes y pone al cliente a esperar dentro del mensaje de forma pasiva. Así una vez el Servidor responda el recurso será despertado.

Finalmente tenemos el método **public** **void** endCliente().Este método se llama cuando el cliente no tiene más mensajes por enviar, pues al suceder esto se dice que el cliente está satisfecho y se cierra el thread.