Clase "Servidor":

• Contiene **8 validaciones (bool)**. Cada una de ellas valida algo diferente (que no haya caracteres inválidos, que no haya dos operadores juntos, etc.). Retornan *true* si todo está OK.

• El método **realizarOperación (int)** solo se ejecuta si todas las validaciones fueron exitosas. Se tienen (2) char[] que guardan los operandos recibidos (si se recibió un solo operando o la operación a hacer es "factorial" el segundo char[] queda en desuso). Tambien contiene un char que guarda la operación recibida. Una vez que separa el/los número/s y conoce la operación, por medio de la estructura *switch case* y las funciones *atoi* realiza la operación, guarda el resultado y lo retorna.

• El método **Recibir** recibe una información o pedido de Cliente. Hay 4 posibles casos: (1) Si el largo del mensaje >= 20 caracteres Cliente envió una operación, por lo que realizará todas las validaciones una por una. (1.1) Si *cualquier* validación falla coloca el mensaje de error en el buffer para enviárselo a Cliente (1.2) Si todas las validaciones están OK, se llama al método **realizarOperacion** y se guarda como array de char en el buffer para mostrarle el resultado a Cliente. (2) Si se recibieron (30) unos Cliente quiere ver el log: Se abre el archivo, se guarda el contenido del mismo en una variable *string* y se guarda desde el primer renglón luego de que resten 1023 caracteres en el buffer para enviárselo al cliente (3) Si el mensaje recibido es (30) dos Cliente superó el time out: Servidor desconecta al cliente, se queda a la espera de un nuevo cliente y Cliente vuelve a la pantalla de conexión. (4) Si el mensaje recibido es (30) tres Cliente cerró sesión: Sucede lo mismo que en el caso 2 con la diferencia de que Cliente deberá cerrar la app y volver a ingresar.

Clase "Cliente":

• Cuando se ejecuta el constructor, consulta el puerto de conexión. La función main contiene el **menú**: Siempre que Cliente ingrese una opción errónea, se le solicitará el ingreso nuevamente. Las opciones están trabajadar con char[] para que el programa no se rompa si el cliente ingresa algo erróneo (por ejemplo, una cadena). Tiene (3) opciones (cálculo/log/cerrar sesión). (1) El cliente ingresa la operación: (1.1) Si supera los 20 caracteres se le solicita nuevo ingreso (1.2) Si ingresa "volver" el método **Enviar** retorna 1 y, en ese caso, no hará nada y volverá al while (es decir, a mostrar las 3 opciones) (1.3) Si ingresa una cadena < 20 caracteres y distinta a "volver", la cadena es enviada a Servidor (2) El cliente pide el log: Se llama al método **pedirLog** que envía (30) unos a Servidor. (3) El cliente cierra sesión: Llama al método **solicitarDesconexion** que manda (30) tres al servidor.

Server.log:

Se crea de forma *manual* en el directorio. Siempre se abre en modo *escritura al final del archivo.* Siempre que se produce un evento se llama al método **agregarTextoLog**: Recibe como argumento la cadena que queremos agregar en el .txt, guarda en un char[] la fecha y hora actual del sistema, le concatena el texto recibido como argumento y lo agrega al archivo.

La obtención de la fecha & hora se logra gracias al método **getCurrentDateAndHour**, que guarda en la cadena de caracteres recibida como argumento la fecha&hora actual.

Dato interesante sobre el algoritmo para mostrar a Cliente el log: La capacidad máxima del buffer[] es de **1023** caracteres. Si la cantidad de caracteres del log es inferior a 1023 se lo mando completo, pero ¿Qué hago si *server.log* tiene más de 1023 caracteres?: En vez de enviarlo completo, tomo el *length()* del string obtenido del archivo, le resto 1023 y ahí obtengo sus *últimos 1023 caracteres.* Para no enviar el primer renglón entrecortado, mientras que no esté en el comienzo del renglón & no detecte un salto de línea, muevo el índice hasta que llegue al próximo renglón. El cliente verá por consola las últimas transacciones realizadas (tal vez no todas). Pero toda la información está dentro del archivo.

Time out:

Fue resuelto con la librería **time.h**: Permite manipular la fecha y hora del sistema. La clase **Cliente** tiene registrada Fecha & Hora del último acceso y todas las validaciones para desconectarlo en caso de haber cumplido el time out.

Una vez que el Cliente se conecta, se registra el momento del acceso. Cada vez que Cliente ingresa algo por teclado, se valida que no hayan pasado 2’ del último acceso (tomando como referencia la Fecha & Hora actual). En ese caso se llama al método **solicitarDesconexionPorInactividad**; si no superó el time out se *actualiza* el último acceso con el momento actual.