Trabajo Practico Nº01

Conceptos básicos para la construcción de Sistemas Distribuidos

Fecha de entrega: 03/04/2020

Ledesma Damián – 155825 – damiledesma67@gmail.com
Salinas Leonardo – 104478 – Sal_chueco@hotmail.com
Pittavino Patricio – 121476 - pitta1881@gmail.com

INTRODUCCIÓN:

Todos los ejercicios fueron desarrollados bajo el IDE IntelliJ IDEA. Para compilar y ejecutar cualquiera de los proyectos, éstos deben ser importados como un Proyecto Maven, esperar a que se instalen las dependencias necesarias y luego como medida general tendremos en cada paquete un "Servidor.java" y un "Cliente.java" los cuales deben ser ejecutados en dicho orden, y a partir del Ejercicio 2 se podrán ejecutar varios Clientes simultáneamente.

Adicionalmente, todos los proyectos cuentan con un Logger del lado del Servidor el cual capturará acciones tales como "Servidor Iniciado", "Mensaje Recibido", "Mensaje Enviado", etc., identificando para la comunicación por Sockets, la IP y el Puerto, mientras que para aquella comunicación por RMI, solo se identifica la IP. También se capturará mensajes de error.

1. *Escriba un servidor que, usando sockets, reciba un mensaje de texto y lo repita a su cliente. Programe también el cliente para verificar y probar el comportamiento del servidor. Realice la implementación en TCP y comente los resultados.*

La resolución se encuentra en: **TP1-Ej1\src\main\java\Ej1**

ServerSide	ClientSide
Servidor.java	Cliente.java
Log.java	
logVolatil.java	

El Servidor abre un Socket en un Puerto especifico en el cual estará esperando a UN solo cliente. Cuando el Cliente se conecta a dicho Puerto, le pide al usuario un Input por consola, que al presionar Enter se envía al Servidor. Éste lo recibe y lo envía de vuelta al Cliente tal cual. En el caso que el usuario escriba la palabra reservada "Exit" y la envíe, el Servidor terminará la conexión sin retornar nada.

2. *Modifique el programa anterior para que pueda atender varios clientes a la vez.*

La resolución se encuentra en: **TP1-Ej2\src\main\java\Ej2**

ServerSide	ClientSide
Servidor.java	Cliente.java
ClientHandler.java	
Log.java	
logVolatil.java	

Similar al programa anterior, pero en esta ocasión, luego de abrir el Socket del Servidor, se crea un ciclo infinito en el que en cada paso se acepta a un nuevo Cliente, se crea un objeto de la clase ClientHandler quien recibe por parámetro el Socket de dicho Cliente y el cual será administrado por un nuevo Hilo (Thread).

3. *Escriba un servidor de mensajes en colas, que permita a los clientes dejar un mensaje (identificando de alguna forma a quién se lo dejan), y bajar los mensajes que le están dirigidos. La comunicación entre cliente y servidor debe ser mediante sockets, y el servidor debe poder atender varios clientes a la vez.*

La resolución se encuentra en: **TP1-Ej3\src\main\java\Ej3**

ServerSide	ClientSide
Servidor.java	Cliente.java
ClientHandler.java	Mensaje.java
Mensaje.java	
Log.java	
logVolatil.java	

La conexión entre Cliente y Servidor se realiza de la misma forma. Ahora el Servidor cuenta con una cola de mensajes volátil de tipo ArrayList<Mensaje> el cual se pasa por parámetro a cada hilo, quienes tendrán los métodos de "Leer Mensajes" y "Añadir mensajes a la Cola". Por su parte el Usuario en primer lugar se identifica con un "Nick" y luego se le mostrará un Menú con las opciones "Ver Mensajes", "Escribir Mensaje", "Actualizar Bandeja" y "Salir". El programa continuará hasta tanto no escoja la opción Salir. Los mensajes son intercambiados previa transformación a String en formato Json implementando la Dependencia Gson.

4. *Modifique el programa anterior para que el mensaje de la cola sea borrado por el servidor una vez que el cliente confirma, mediante un mensaje de tipo ACK, que efectivamente recibió el mensaje que estaba en la cola.*

La resolución se encuentra en: **TP1-Ej4\src\main\java\Ej4**

ServerSide	ClientSide
Servidor.java	Cliente.java
ClientHandler.java	Mensaje.java
Mensaje.java	
Log.java	
logVolatil.java	

Ampliando el ejercicio anterior, se añadió al Menú la opción "Borrar Mensajes", la cual lista uno por uno los mensajes y pregunta al Usuario si desea eliminarlo. Por otro lado, se agregaron las propiedades "ID" y "Borrado" (boolean) a la clase Mensaje, con el primero se identifica unívocamente cada mensaje, mientras que el segundo indica si el mensaje fue borrado o no. Cabe aclarar que cuando el Servidor recibe la confirmación para eliminar el mensaje, éste hace un Borrado Lógico, es decir, no lo elimina del ArrayList, sino que cambia el valor de la propiedad Borrado de dicho mensaje, luego, cuando debe listar los mensajes de un Usuario, solo muestra aquellos cuya propiedad Borrado es falsa. Para futuras implementaciones, se debería crear un planificador que cada cierto tiempo elimine efectivamente del ArrayList los mensajes "borrados", así como también una nueva opción en el Menú que liste los mensajes de la "Papelera de Reciclaje".

5. *Escribir un servicio que devuelva información de clima del lugar donde reside el servidor. Esta información podrá generarse de forma aleatoria. Deberá ser realizado con RMI. Para ello considere la interface remota, la clase (lado servidor) que implementa esa interface, el servidor, y un cliente que permita probar este funcionamiento.*

La resolución se encuentra en: **TP1-Ej5\src\main\java\Ej5**

ServerSide	ClientSide
Servidor.java	Cliente.java
ServidorImplementer.java	(Iface)RemoteInterface.java
(Iface)RemoteInterface.java	
Log.java	
logVolatil.java	

El Servidor RMI inicia en un Puerto especifico y publica su servicio “Clima”. El cliente se conecta al Servidor y utiliza el servicio Clima. Tanto Cliente como Servidor implementan la Interfaz RemoteInterface por lo que saben que métodos usa cada servicio. El clima se obtiene utilizando la API openweathermap.

6. *Escribir un servidor utilizando RMI, que ofrezca la posibilidad de sumar y restar vectores de enteros. Introduzca un error en su código que modifique los vectores recibidos por parámetro. Qué impacto se genera? Qué conclusión saca sobre la forma de pasaje de parámetros en RMI? Mostrar claramente los valores de los vectores del lado cliente, antes y después de la ejecución de la suma o resta.*

La resolución se encuentra en: **TP1-Ej6\src\main\java\Ej6**

ServerSide	ClientSide
ServerMain.java	Cliente.java
ServerImplementer.java	(Iface)RemoteInt.java
(Iface)RemoteInt.java	
Log.java	
logVolatil.java	

Los métodos “sumaVectores” y “restaVectores” se encuentran en la clase ServerImplementer, en el primero de ellos, luego de hacer los cálculos correspondientes, se añade un ‘Error’, modificando los valores de ambos vectores que llegan por parámetro, pero éstos al ser pasado por VALOR, no sufren ningún cambio del lado del Cliente. Nunca podrían ser por REFERENCIA ya que las partes no comparten el mismo espacio de memoria.

7. *Implemente un servidor RMI que resuelva tareas genéricas. Para ello tener en cuenta la interface Tarea, que tendrá un método ejecutar(). El objetivo es que desde el cliente se puedan escribir objetos (que implementen la interface Tarea) que hagan un cálculo concreto (calcular un número aleatorio, un primo, el valor de Pi con cierta precisión, etc.), y que esa tarea se pase al servidor RMI, quien hará el cálculo y devolverá el valor al cliente.*

La resolución se encuentra en: **TP1-Ej7\src\main\java\Ej7**

ServerSide	ClientSide
Servidor.java	Cliente.java
ServidorImplementer.java	(Iface)ITareaServer.java
(Iface)Tarea.java	(Iface)Tarea.java
Log.java	NumPiPrecision.java
logVolatil.java	NumAleatorio.java
	NumAleatorioPrimo.java

Manteniendo la estructura de trabajo con RMI para los ejercicios anteriores, se agrega una Interface nueva llamada *'Tarea'*. Dicha interfaz será implementada por las tareas específicas (*NumAleatorio, NumAleatorioPrimo y NumPiPrecision*) implementado el método correspondiente según su tarea a realizar.

La interfaz *RemoteInterface* contiene el método *ejecutar* al igual que ejercicios previos con el agregado de un argumento de tipo Tarea.

Desde el cliente se realizan las correspondientes peticiones según opción seleccionada por usuario. El *ServidorImplementer* recibe la orden de ejecutar con una tarea como argumento (ya que implementa la interface RemoteInterface) y llama a ejecutar el método de la tarea recibida.