andres ramirez - Pedro Ortiz

SEMINARIO DE ACTUALIZACION

envío de objetos con socket

Tabla de contenido

[SOCKETS EN JAVA (CLIENTE Y SERVIDOR) 2](#_Toc477388640)

[Como funciona? 2](#_Toc477388641)

[BITACORA DE RED: 2](#_Toc477388642)

[1. API: 3](#_Toc477388643)

[2. SocketCliente: 5](#_Toc477388644)

[2.1 Interfaz gráfica: 8](#_Toc477388645)

[3. SocketServer: 9](#_Toc477388646)

[3.2 Interfaz grafica 12](#_Toc477388647)

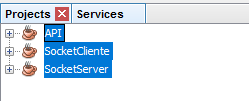
# **SOCKETS EN JAVA (CLIENTE Y SERVIDOR)**

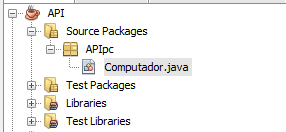
Los sockets son básicamente formas en las que podemos interconectar 2 (o más) programas mediante el uso de la internet. En java se utilizan para poder crear conexiones utilizando una IP/hostname y un puerto para establecer la conexión

## Como funciona?

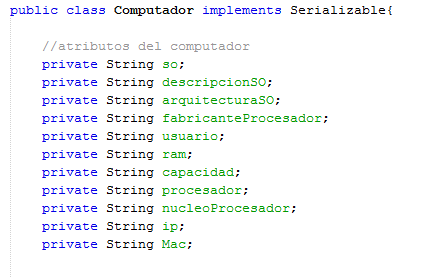
El modelo más básico de los sockets consta de 2 simples programas, un **servidor** y un **cliente.**Básicamente el programa servidor comienza a “escuchar” en un puerto determinado (nosotros lo especificamos), y posteriormente el programa que la hace de “cliente” debe conocer la ip o nombre de dominio/hostname del servidor y el puerto que está escuchando, al saber esto simplemente solicita establecer una conexión con el servidor. Es aquí cuando el servidor acepta esa conexión y se puede decir que estos programas están “conectados”, de este modo pueden intercambiar información.

## BITACORA DE RED:

La idea del proyecto es poder compartir la información técnica de múltiples computadores(clientes) a un servidor. Esto para ayudar a controlar los cambios que puedan efectuarse en cada uno de ellos. Como es de suponerse, se realizaron tres proyectos por separado:

1. ******API:** Va a contener la clase “Computador” con sus respectivos atributos. Se realiza e manera independiente para utilizarlo como un API tanto en el servidor como en en el cliente. Es necesario tenerlo en todos los computadores que quieran compartir sus especificaciones.

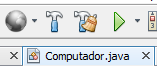
Después de crear la clase “Computador.java” creamos atributos que vamos a compartir por medio de socket. Además, se debe implementar la interfaz serializable que va a permitir enviar por medio de la red, objetos de este tipo. Nota: insertar get y set.



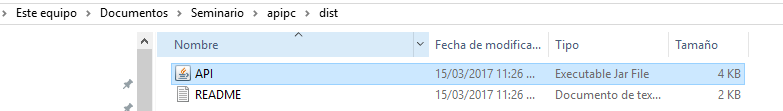
Para que los computadores que van a usar el aplicativo reconozcan el tipo de objeto que se va a enviar por la red, es necesario agregar este API creado anteriormente a los proyectos que se va a desarrollar.

Primero se genera el .jar para poder insertarlo en SocketCliente y SocketServidor. Asi:

* Para generar el JAR simplemente se hace clic sobre el botón:



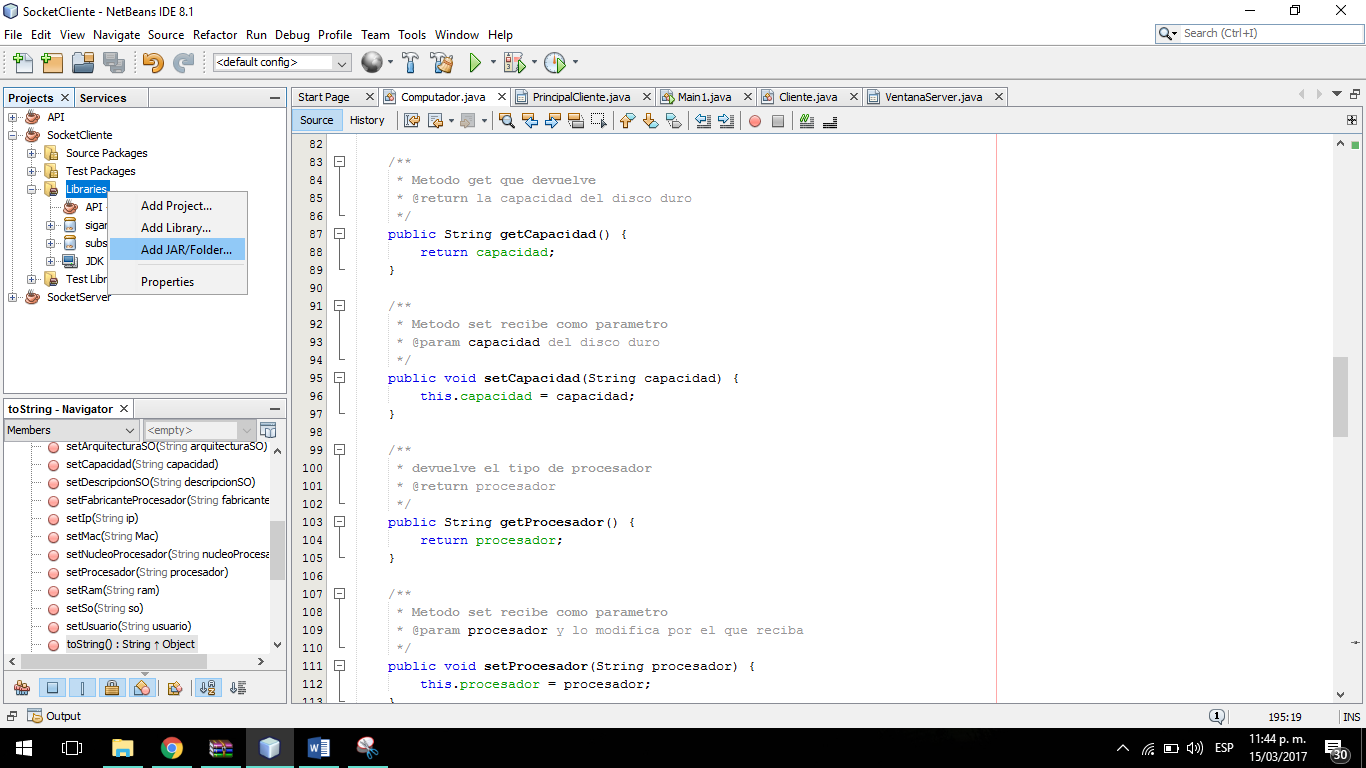
* El archivo generado va a estar en la ruta “…\nombre\_proyecto \dist”



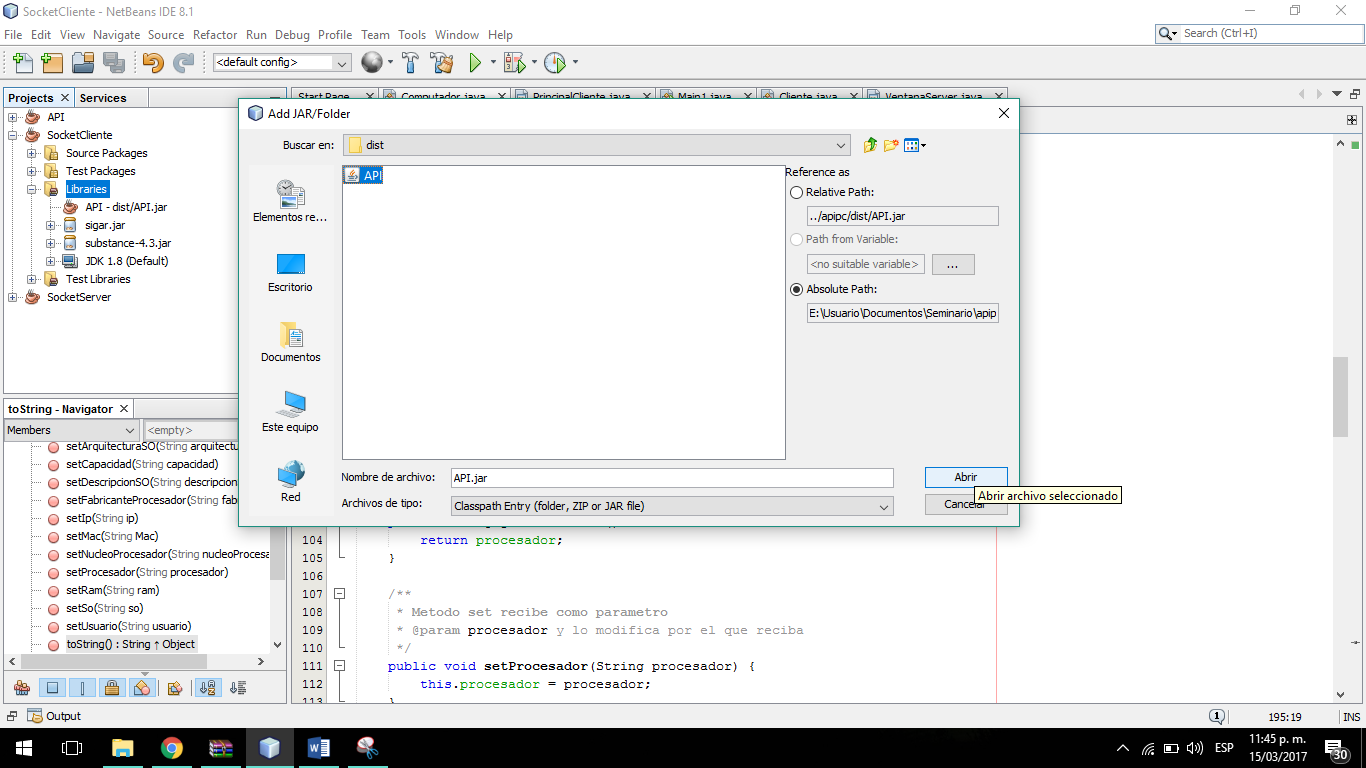
1. **SocketCliente:** Implementación por medio de socket para compartir la información hacia el servidor.

Lo primero que debemos realizar es agregar el API de computadores al proyecto de cliente, de la siguiente forma:

* Add JAR

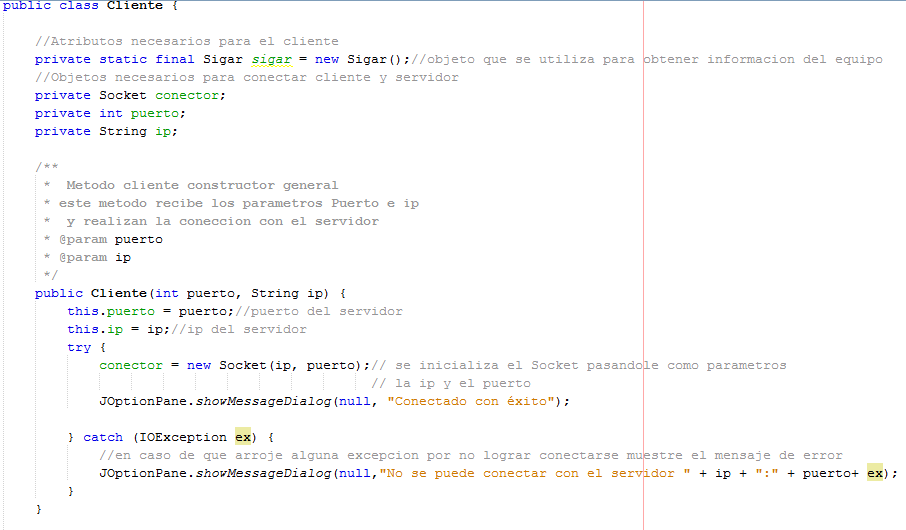


* Buscamos la carpeta dist del proyecto del API y agregamos el JAR generado anteriormente

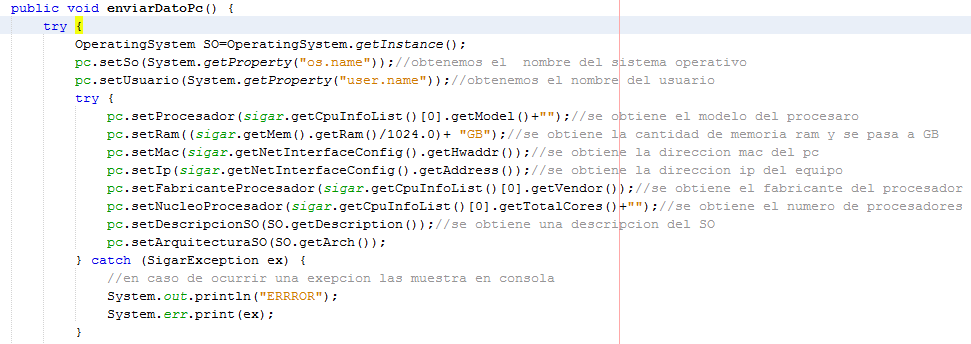


Nota: Ademas de el API se agregaron dos .JAR, uno que mejora la interfaz grafica de los proyector (subtance 4.3) y otro que va a ayudar a extraer información física y/o lógica del computador (sigar).

A continuación se muestra una captura de la implementación de la Clase “Cliente” y que realiza cada sentencia:

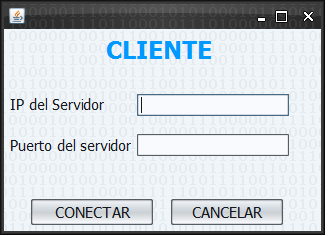


Continuando con la clase Cliente, debemos implementar el método que obtenga la información del computador y proceda a enviarlo al servidor. Con la librería SIGAR obtenemos lo datos necesarios del computador. Es importante recalcar que los métodos get y set deben ser incluidos en esta clase. La explicación está detallada en la imagen.

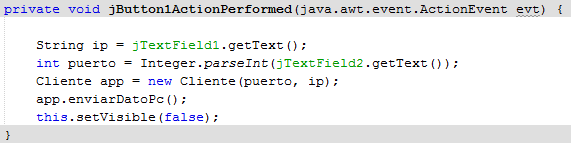




* 1. Interfaz gráfica: Esta interfaz es muy sencilla, con dos campos de textos donde se introducen la IP y el puerto del servidor. Este debe ser proporcionada por el mismo servidor. Además, dos botones, uno anula la operación y otro que inicia la comunicación.

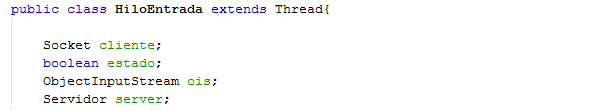


Al accionar el botón conectar se implementa las siguientes líneas de código. S obtienen la ip y puerto de los campos de textos. Se crea el objeto de tipo cliente con sus datos y se envía por el método “enviar dato” implementado anteriormente.

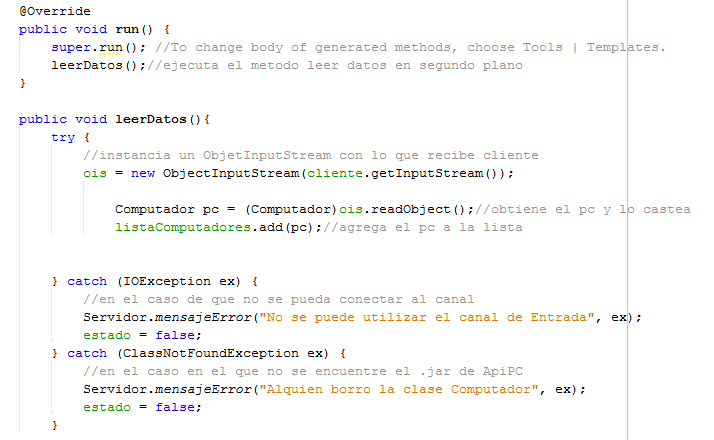


1. SocketServer: Realiza la comunicación y obtiene información de computadores conectados.

* **Clase HiloEntrada:** Esta es la clase encargada de recibir el objeto que es enviado desde el servidor, es necesario que se hilo para poder ejecutar de manera simultánea. Por eso hereda de “Thread”.

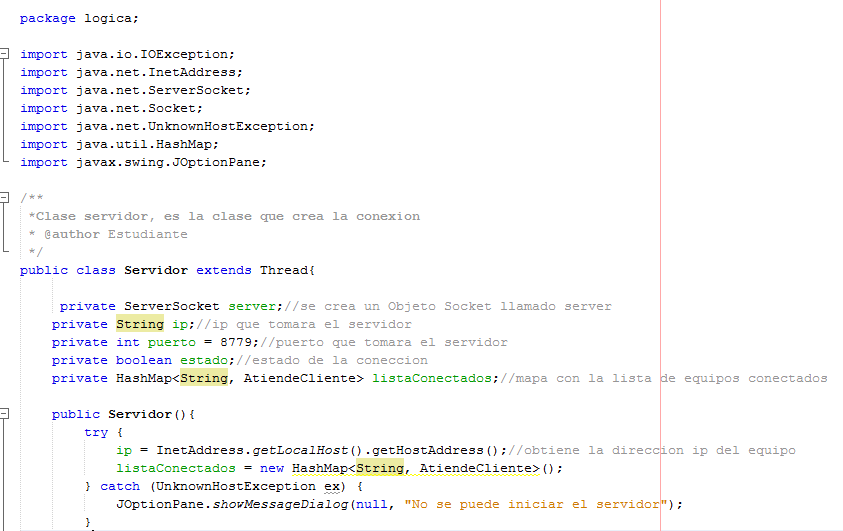


En el método run, que ejecuta el hilo, llamamos a un método leer es datos. Este método recibe lo que ha enviado el cliente por medio de una variable ObjectInputStream. Se realiza el respectivo casteo con la Clase Computador que recordemos está incluida en el api creado anteriormente. Este PC recibido lo agregamos a la lista de computadores.

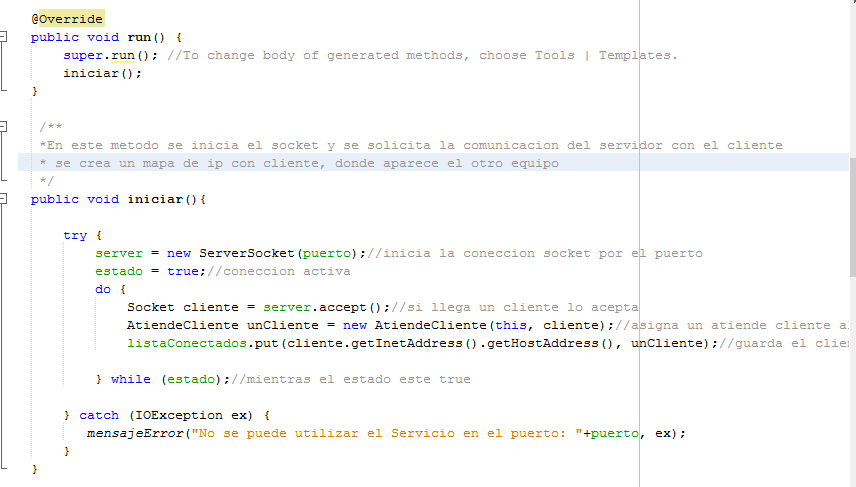


* **Clase Servidor**

Al abrir esta clase tenemos los atributos básicos IP del servidor, Puerto que está utilizando, HashMap<String, AtiendeCliente> listaConectados es un colección de referencias hacia los objetos exportados por los clientes y que permite realizar las conexiones.

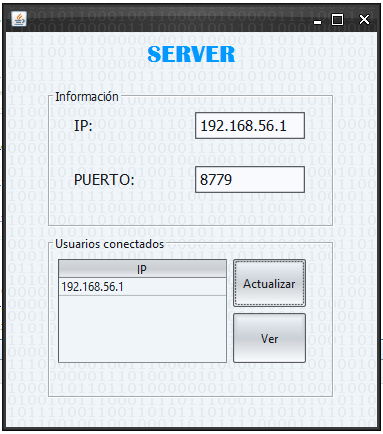


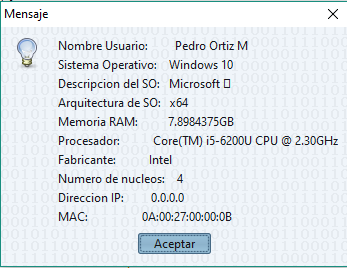
Aquí tenemos uno de los métodos que es iniciar que le permite al cliente establecer una conexión de doble canal con el servidor (Ósea que servidor lo escucha y el escucha al servidor), como vemos este se ejecuta en segundo plano



En éste método lo que hace es activar la conexión pasándole por parámetro el puerto al cual queremos que conecte, si un cliente se quiere conectar se envía a atiende cliente, y se agrega en el mapa lista contactos, siempre y cuando el estado sea true, si llega a ocurrir un error al momento de crear la conexión se notifica enviando un mensaje de advertencia.

### 3.2 Interfaz grafica





|  |  |
| --- | --- |
| SOCKET | RMI |
| Permite intercambiar flujos de datos | Permite intercambiar flujos de datos |
| Una invocación remota a un método, y este es pasado por copia en vez de referencia | Los clientes de objetos remotos pueden interactuar con las interfaces remotas |
| Hay menos posibilidades de que haya fallas | Se debe manejar una gran cantidad de excepciones |
| Poca seguridad | Un poco más seguro que socket |
| Fácil implementación | Fácil implementación |
| Utiliza cliente servidor | Utiliza cliente servidor |
|  |  |