**­­UDP**

**1º** Crearemos un programa servidor de caramelos que escuche en un puerto determinado (ver documento de rango de puertos para elegir uno adecuado) este programa entregará una cantidad solicitada de caramelos al proceso que lo solicite, siempre y cuando le queden caramelos. En otro proyecto crearemos hilos (n) que cada x tiempo le solicitan una cantidad determinada de caramelos al servidor. Cuando el servidor se quede sin caramelos indicará a los hilos conocidos que ya no tiene nada que servir y se acabará. Los hilos que reciban el código de fin de caramelos por parte del servidor, también terminarán.

**2º** Vamos a simular nuestro propio servidor DHCP. Nuestro servidor sólo proporciona 10 direcciones IP (un poco escasillo) y el formato de las direcciones es simplemente un número del 1 al 20.

Cuando un cliente (hilo nuevo cliente) se conecta a la red hace un broadcast solicitando una IP (el servidor se la otorga indicándole además cuál es su propia IP, la del servidor). El cliente simula el funcionamiento de un ordenador durante un tiempo determinado y después le indica al servidor que se va y que la IP queda libre (esta comunicación ya no será un broadcast ya que conocemos la IP del servidor). El servidor podré, por lo tanto, proporcionar esa IP a otro hilo cliente.

Si no hay más IP’s que proporcionar (agotamiento de IP) se informa al cliente que muere inmediatamente.

Simular el sistema durante 1 minuto.

**3º** Tendremos un servidor central en el que está alojado una cuenta bancaria. Los clientes enviarán sus peticiones al servidor. En estas peticiones se deberá indicar lo que se pretende hacer: ingresar, retirar o consultar. El servidor responderá adecuadamente y el cliente le enviará los datos necesarios. El tipo de comunicación será no confiable (UDP) y se debe garantizar que ninguna petición deja de ser atendida de la forma más eficiente y que se respeta la exclusión mutua de la cuenta en el servidor.

**4º** Vamos a crear el juego de los camellos en red. El programa servidor preguntará al usuario cuantos jugadores serán; cuando todos ellos se hayan conectado comienza el juego.

Cuando el juego haya terminado el servidor mostrará los resultados e informará a cada jugador de lo que ha hecho. Los clientes enviarán avanzas (generados aleatoriamente) que usarán sus camellos en el servidor.

En un segundo apartado del juego, los clientes reciben información del servidor de cómo va la competición y van viendo los progresos de su camello y el de los otros jugadores.

**5º** Realiza un juego en el que el servidor admite un número indefinido de jugadores con los que jugará partidas independientes. El juego elegido será las siete y media. Los jugadores irán enviando órdenes del tipo: quiero carta o me planto. El servidor informará del resultado de la partida y ambos (cliente e hilo del servidor) terminarán.

**6º** Diseña el juego de la mosca en red. En nuestro tablero de NxN se esconden M moscas. Se conectarán 4 jugadores (sólo 4). Después los clientes irán enviando posiciones del tablero, si hay mosca en ellos se les anota en su marcador. El servidor accederá al tablero con las peticiones de los clientes por orden de llegada. Cuando todas las moscas estén matadas se informará a cada cliente de cuantas ha cazado y de su posición final en el *pódium*.

**TCP**

**7º** Crear un programa que explore los puertos abiertos de tu equipo. Generaliza el ejercicio para descubrir los puertos abiertos de cualquier equipo de tu red.

**8º** Cread un servidor que abra un fichero solicitado por un cliente y se lo envíe a través de la red. La parte del cliente debe ser gráfica. El servidor guardará un fichero de bitácora indicando qué cliente ha accedido y cuando.

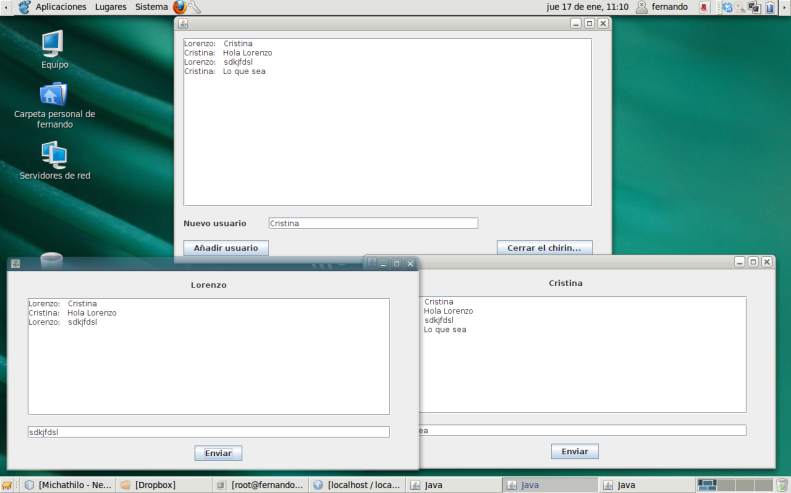
Generalizar este ejercicio para que:

* El servidor pueda atender a varios clientes.
* Los clientes puedan modificar el contenido del archivo.

**9º** Crear un chat simple (muy simple), que usa Sockets para crear conexiones de red. Consta de un servidor (*Servidor.java*) y un cliente (*Cliente.java*), mediante los cuales se puede establecer una conversación entre varios clientes. Si se escribe “TERMINAR” (sin comillas), la conexión se cierra para ese cliente.

**10º** Repite el ejercicio anterior con ventanas swing de modo que una ventana será el servidor y otra aplicación serán clientes.

El salón de chat se puede realizar con un área de texto.



**11º** Vamos a realizar un sistema de gestión para DAM2. En este sistema tendremos dos tipos de usuario (profesores y alumnos). Ambos acceden al sistema con un usuario y una clave pero según su rol verán unas pantallas u otras.

Los profesores: podrán poner sus notas (solo una nota final para simplificar la base de datos) en sus asignaturas (y modificarlas si lo ven preciso). También podrán enviar mensajes a todos sus alumnos o sólo a aquellos que quiera.

Los alumnos: verán sólo sus notas y podrán acceder a la zona de mensajes para ver si ha recibido alguno. El sistema le permitirá responder y redactar mensajes a los profesores.

**[Opcional]** Permitir que el sistema de mensajería sea generalizado para todos los usuarios.

**12º** Realiza un programa que reparta cartas a sus clientes que lo soliciten. Se debe pasar un objeto carta que el cliente mostrará por pantalla. El servidor cerrará su conexión cuando se quede sin cartas.

Reutiliza lo programado anteriormente para realizar un juego sencillo. Nuestro servidor permitirá conectarse a dos jugadores que jugarán a las siete y media. Cada jugador, por turno, puede pedir cartas al servidor, con la condición de que una sola de las que tiene permanezca tapada. Si alguno se pasa ganará el otro; si ambos se plantan el servidor determinará quien ha ganado. Cuando la partida haya terminado se indicará la puntuación de ambos a los dos jugadores y quien ha ganado o si se ha quedado empate. Después el servidor esperará a otros dos jugadores.

**[Opcional]** Una vez se haya realizado el ejercicio, generaliza la solución para un número indeterminado de partidas. Cuando llega un jugador: si hay partida iniciada se suma a ella, en caso contrario se inicia una partida para él y se quedará esperando contrincante.

**13º** Vamos a diseñar un troyano muy sencillo con fines didácticos y sin ánimo de caer en el lado oscuro. Utilizaremos una de las prácticas que realizamos en el tema de hilos; el proyecto que era un editor de textos que tenía hilos para contar las letras y palabras que estábamos escribiendo.

Cuando nuestro cándido cliente ejecute el programa se conectará con nuestro equipo (de forma silenciosa sin que el usuario de nuestro procesador se entere de nada). Nuestro equipo piratón registrará a ese usuario (IP, nombre de usuario, carpeta de trabajo, nombre de equipo… tó lo que pillemos) y mantendrá el contacto con ese usuario (por debajo). Si algún otro utiliza nuestro procesador en otra máquina, repetimos la operación.

En un momento dado podemos “atacar” a todas las máquinas “infectadas”. Nuestro ataque consiste en volcar en el área de texto de nuestro usuario toda la información que hemos sacado de su sistema con la frase final de Yoda: “*Grande!?!?! Gran guerrero, dices…. mmmm. La guerra no hace a uno grande*”.

Nota – métodos para obtener información del sistema:

String sSistemaOperativo = System.getProperty("os.name");

System.out.println(sSistemaOperativo);

InetAddress localHost = InetAddress.getLocalHost();

System.out.println(localHost.getHostName());

System.out.println(localHost.getHostAddress());

String usuario = System.getProperty("user.name");

System.out.println(usuario);

String arquitecturaSO = System.getProperty("os.arch");

System.out.println(arquitecturaSO);

String versionSO = System.getProperty("os.version");

System.out.println(versionSO);

String currentDIR = System.getProperty("user.dir");

System.out.println(currentDIR);

String userHome = System.getProperty("user.home");

System.out.println(userHome);