Curso 2º D.A.M.
Mercedes Hernández Losada

CLASES JAVA PARA COMUNICACIONES EN RED.

Los equipos conectados a internet se comunican utilizando **el protocolo de transporte TCP o UDP.**
Cuando se escriben programas Java que se comunican a través de la red, **se está programando en la capa de aplicación**, normalmente no es necesario preocuparse por las capas de transporte TCP o UDP, en su lugar **se utilizan las clases del paquete java.net.**
Existen diferencias entre unas clases y otras, tenemos que decidir qué clases usar en los programas; Como sabemos:

- TCP: Protocolo basado en la conexión. Garantiza que los datos envíados desde un extremo de la conexión llegan al otro extremo y además los datos llegan en el mismo orden en que fueron enviados. Se establece un canal de comunicación particular entre las dos aplicaciones.
- UDP : No está basado en la conexión, envía de una aplicación a otra **paquetes de datos independientes denominados datagramas,** el orden de entrega no es importante y no se garantiza la entrega de los paquetes enviados.

EL PAQUETE java.net

Contiene clases e interfaces para la implementación de aplicaciones que se comunican en red.

- **La clase URL**, (Uniform Resource Locator) Localizador Uniforme de Recursos. Representa un puntero a un recurso en la web.
- **La clase URLConnection**, que admite operaciones más complejas en la URL
- **Las clases ServerSocket y Socket**, **para dar soporte a sockets TCP. Serversocket** utilizada por el programa servidor para crear un socket en un puerto por el que está escuchando las peticiones de conexión de los procesos clientes. **Socket** utilizada tanto por el proceso cliente como por el proceso servidor para comunicarse entre sí, escribiendo y leyendo datos utilizando Streams.
- **Las clases DatagramSocket, MulticastSocket y DatagramPacket, para dar soporte a la comunicación** entre aplicaciones **vía datagramas UDP.**
- **La clase InetAddres**, que representa las direcciones de internet.

LOS PUERTOS

En general un ordenador tiene una única conexión física a la red. Los datos destinados a ese ordenador llegan a través de esa conexión, sin embargo los datos pueden estar destinados a diferentes aplicaciones que se están ejecutando en el ordenador.
Entonces, mediante **el uso de puertos se diferencian los datos destinados a cada aplicación en el ordenador.**

Los datos transmitidos a través de internet van acompañados de información de direccionamiento que identifica la máquina y el puerto para el que está destinado. La máquina se identifica por su dirección **IP de 32 bits,** por tanto para enviar datos a una máquina concreta necesitamos conocer su

IP. Los **puertos** se identifican mediante un número de **16 bits**, que TCP y UDP utilizan para entregar los datos a la aplicación correcta dentro de la máquina.
En la comunicación TCP una aplicación servidor vincula un socket servidor a un puerto especifíco. Una aplicación cliente puede comunicarse con el servidor envíandole peticiones a través de ese puerto.
Hay puertos TCP de 0 a 65535. los puertos en el rango de 0 a 1023 están reservados para servicios privilegiados. Otros puertos de 1024 a 49151 están reservados para aplicaciones concretas, (por ejemplo el 3306 lo usa MySQL, el 1521 Oracle)
Por último de 49152 a 65535 no están reservados para ninguna aplicación concreta.
En la comunicación UDP basada en datagramas, el paquete de datagramas contiene el número de puerto de su destino.

LA CLASE InetAddress

La clase InetAddress es la abstracción que representa una dirección IP.
Tiene dos subclases Inet4Address para direcciones IPv4 e Inet6Address para direcciones IPv6; en la mayoría de los casos InetAddress proporciona la funcionalidad necesaria y no es necesario recurrir a estas subclases.

La clase InetAddress (java.net.InetAddress), métodos más importantes:

MÉTODO	TIPO DE RETORNO	FUNCIÓN
getLocalHost()	InetAddress	Devuelve un objeto InetAddress que representa la dirección IP de la máquina donde se está ejecutando el programa. Puede lanzar la excepción UnknownHostException
getByName(String host)	InetAddress	Devuelve un objeto InetAddress que representa la dirección IP de la máquina que se especifica como parámetro (host). Este parámetro puede ser el nombre de la máquina, un nombre de dominio o una dirección IP. Puede lanzar la excepción UnknownHostException
getAllByName(String host)	InetAddress[]	Devuelve un array de objetos de tipo InetAddress. Este método es útil para averiguar todas las direcciones IP que tenga asignada una máquina en particular. Puede lanzar la excepción UnknownHostException
getHostAddress()	String	Devuelve la dirección IP de un objeto InetAddress en forma de cadena.
getHostName()	String	Devuelve el nombre del host de un objeto

		InetAddress.
getCanonicalHostName()	String	Obtiene el nombre canónico completo de un objeto InetAddress.(suele ser la dirección real del host)

La forma más típica de crear instancias InetAddress es invocando al método estático getByName(String host) pasándole el nombre DNS del host como parámetro.
Este objeto recibido representará la dirección IP de ese host y se podrá utilizar para construir sockets.

LOS SOCKETS

Los protocolos TCP y UDP utilizan el concepto de socket para proporcionar a las aplicaciones **los puntos extremos de la comunicación entre aplicaciones o procesos.**
La comunicación entre aplicaciones va a consistir en la transmisión de un mensaje entre un socket de un proceso y otro socket de otro proceso.
Cada socket tiene una IP y un puerto asociado. El proceso cliente debe conocer el puerto y la IP del proceso servidor. El servidor está escuchando por un determinado puerto acordado. El proceso cliente podrá enviar el mensaje por cualquier puerto.

Los procesos pueden utilizar el mismo socket para enviar y para recibir mensajes. Cada proceso escribe en su socket para enviar mensajes y lee desde su socket para recibir mensajes.

FUCIONAMIENTO

Un puerto es un punto de destino que identifica la aplicación o proceso dentro de la máquina a la que se dirigen los datos. En una aplicación cliente_servidor, el programa servidor se ejecuta en una máquina específica y tiene un socket servidor que está asociado a un número de puerto específico. El proceso servidor queda a la escucha para recibir peticiones de conexión de los procesos clientes por ese puerto. (método accept)

El proceso cliente tiene que conocer el nombre de la máquina en la que se ejecuta el servidor y ese número de puerto concreto por el que está escuchando. El proceso cliente también tiene que identificarse ante el servidor por lo que la conexión se realizará utilizando el puerto local del cliente asignado por el sistema.

Una vez aceptada la conexión, en el proceso servidor se **obtiene un nuevo socket (Socket) sobre un puerto diferente, este nuevo socket es el que se utilizará para mantener la comunicación de forma permanente con el cliente, por otro lado el proceso servidor sigue atendiendo las**

peticiones de conexión de los posibles procesos clientes mediante el socket original (ServerSocket).

El cliente y el servidor pueden ahora comunicarse, transmitiendo mensajes por la red, escribiendo y leyendo en sus respectivos sockets.

Curso 2º D.A.M.
Mercedes Hernández Losada

TIPOS DE SOCKETS

SOCKETS ORIENTADOS A CONEXIÓN: SOCKETS STREAM.

La comunicación se realiza utilizando el protocolo de transporte TCP, por tanto es una conexión fiable, se garantiza la entrega de los paquetes y el orden en que fueron enviados.
TCP utiliza un sistema de acuse de recibo de los mensajes, de forma que si el emisor no recibe dicho acuse dentro de un determinado tiempo vuelve a transmitir el mensaje.
Los procesos que se comunican deben establecer una conexión mediante un stream. Un stream es una secuencia ordenada de unidades de información (bytes, caracteres, etc.) Están diseñados para acceder a los datos de forma secuencial.
Una vez establecida la conexión, las aplicaciones se van a comunicar leyendo y escribiendo en los stream sin preocuparse de las direcciones de internet ni de los números de puerto.
Los socktes TCP o socktes stream son los utilizados en la gran mayoría de las aplicaciones .
En Java hay dos tipos de sockets stream asociados a las **clases Socket y ServerSocket** para implementar el servidor.

SOCKETS NO ORIENTADOS A CONEXIÓN: SOCKETS DATAGRAM.

La comunicación entre aplicaciones o procesos se realiza mediante el protocolo de transporte UDP. Esta conexión no es fiable y no se garantiza que la información llegue a su destino, tampoco se garantiza el orden de llegada de los paquetes. **Los datagramas se transmiten desde un proceso emisor a otro receptor sin haber establecido previamente una conexión**, un canal de comunicación, sin acuse de recibo ni reintentos.
Cualquier proceso que necesite enviar o recibir mensajes debe crear primero un socket asociado a una dirección IP y a un puerto local. El proceso servidor enlanzará su socket a un puerto local conocido por todos los clientes, sin embargo, el proceso cliente enlazará su socket a cualquier puerto local libre. **Cuando un proceso recibe un mensaje, podrá obtener además del mensaje la dirección IP y el puerto del emisor, permitiéndole entonces responder al emisor.**

Los sockets UDP se usan cuando una entrega rápida es más importante que una entrega garantizada. O en aquellos casos cuando la información que se transmite es tan pequeña que cabe en un datagrama. Se usan para aplicaciones de transmisión de audio o vídeo en tiempo real donde no es posible el reenvio de paquetes retrasados.

Para implementar en Java este tipo de sockets se utilizan las clases **DatagramSocket y DatagramPacket**

Curso 2º D.A.M.
Mercedes Hernández Losada

ENVÍO DE OBJETOS A TRAVÉS DE SOCKETS

Intercambio de objetos entre el programa emisor y receptor o entre programas cliente y servidor usando sockets.

OBJETOS EN SOCKETS TCP

Las clases **ObjectInputStream** y **ObjectOutputStream** nos van a permitir enviar y recibir objetos a través de los sockets TCP.
Utilizaremos los métodos **readObject()** para leer el objeto del stream y **writeObject()** para escribir el objeto al stream.

El constructor de cada una de las clases admite un InputStream y un OutputStream respectivamente.

Para obtener flujo de entrada de donde leer/recibir objetos:

```
ObjectInputStream inObjeto=
    new ObjectInputStream( socket.getInputStream());
```

Para obtener flujo de salida en donde escribir/enviar objetos:

```
ObjectOutputStream outObjeto=
    new ObjectOutputStream( socket.getOutputStream());
```

Las clases a las que pertenecen estos objetos a transmitir tienen que implementar la interface Serializable.

Curso 2º D.A.M.
Mercedes Hernández Losada

GESTIÓN DE SOCKETS UDP.

Los roles clientes-servidor están difusos. Podemos considerar al servidor como el proceso que comienza esperando un mensaje y responde luego. Y al cliente como al proceso que inicia la comunicación. Tanto un proceso como otro necesitan saber en qué ordenador y puerto está escuchando el otro proceso.

Ambos procesos necesitan crear un socket DatagramSocket.

. El servidor crea un socket DatagramSocket asociado a un puerto local para escuchar peticiones de clientes. Permanece a la espera.
. El cliente crea un socket DatagramSocket. Para poder enviar datagramas necesita conocer la IP y el puerto por el que escucha el otro proceso "servidor". Utilizará el **método send()** del socket para enviar el datagrama, en el datagrama de envío va la IP y el puerto de destino.
. El servidor recibe las peticiones mediante el **método receive()** del socket. De el datagrama datagrama se extrae además del mensaje, el puerto y la IP del proceso emisor del datagrama, lo que permite al proceso servidor conocer la dirección del proceso que ha enviado el datagrama. Ahora, utilizando el método send() del socket podrá contestar al cliente emisor.

Cada vez que el proceso emisor envíe datagramas debe indicar explícitamente la dirección IP y el puerto de destino para cada paquete y el proceso receptor debe extraer la dirección IP y el puerto del emisor del paquete.

El paquete del datagrama está formado por los siguientes campos:

CADENA DE BYTES CON EL MENSAJE	LONGITUD DEL MENSAJE	DIRECCIÓN IP DESTINO	Nº PUERTO DESTINO

Curso 2º D.A.M.
Mercedes Hernández Losada

LA CLASE DatagramPacket

Para crear instancias de los datagramas que se van a recibir y de los datagramas que se van a enviar.

CONSTRUCTOR	FUNCIÓN
DatagramPacket(byte[] buf, int length)	**Constructor para datagramas recibidos,** se especifica la cadena bytes para alojar el mensaje y la longitua de la misma.
DatagramPacket(byte[] buf, int offset, int length)	**Constructor para datagramas recibidos,** se especifica la cadena bytes para alojar el mensaje, la longitua de la misma y el offset dentro de la cadena.
DatagramPacket(byte[] buf, int length, InetAddress addrss, int port)	**Constructor para el envío de datagramas,** se especifica la cadena de bytes a enviar, la longitud, el número de puerto de destino y el host especificado en la dirección addrss.
DatagramPacket(byte[] buf, int length, int offset, InetAddress addrss, int port)	**Constructor para el envío de datagrama,** se especifica la cadena de bytes a enviar, un offset dentro de la cadena, la longitud, el número de puerto de destino y el host especificado en la dirección addrss.

MÉTODOS	FUNCIÓN
InetAddress getAddress()	Devuelve la dirección IP del host del que el datagrama se recibió.
byte[] getData()	Devuelve el mensaje contenido en el datagrama tanto recibido como enviado.
int getLength()	Devuelve la longitud de los datos a enviar o recibir.
int getPort()	Devuelve el número de puerto de la máquina remota del que se recibió el datagrama.
setAddress (InetAddress addr)	Establece la dirección IP de la máquina a la que se le envía el datagrama referenciado.
setData(byte[] buf)	Establece el buffer de datos para este paquete referenciado.
setLength(int length)	Establece la longitud
setPort(int Port)	Establece el número de puerto remoto del host al que este datagrama referenciado se envía .

Curso 2º D.A.M.
Mercedes Hernández Losada

La Clase DatagramaSocket
Da soporte para la recepción y envío de datagramas UDP.

CONSTRUCTOR	FUNCIÓN
DatagramSocket()	**Construye un socket para datagrama**s, el sistema elige un puerto de los que están sin utilizar.
DatagramSocket(int port)	**Construye un socket para datagramas** y lo conecta al puerto local especificado
DatagramSocket(int port, InetAddress ip)	**Construye un socket para datagramas,** además se especifica la dirección local a la que se va a asociar el socket

MÉTODO	FUNCIÓN
receive (DatagramPacket paquete)	Recibe un DatagramPacket del socket y llena paquete con los datos que recibe. Mensaje, longitud y origen. Puede lanzar la excepción *IOException.* *Este método se bloquea hasta que se recibe un datagrama, a menos que se establezca un tiempo límite (timeout) sobre el socket.*
send(DatagramPacket paquete)	Envía un DatagramPacket a través del socket. El argumento paquete contiene el mensaje y su destino. Puede lanzar la excepción *IOException*
close()	Cierra el socket
int getLocalPort()	Devuelve el número de puerto en el host local al que está enlazado el socket. -1 si el socket está cerrado.0 si no está enlazado a ningún puerto.
int getPort()	Devuelve el número de puerto al que está conectado el socket. -1 sino está conectado.
connect(InetAddress address, int port)	Conecta el socket a un puerto y a una dirección IP, el socket podrá enviar y recibir mensajes desde esa dirección.
setSoTimeout(int timeout)	Permite establecer un tiempo de espera límite. El método receive() se bloquea durante el tiempo fijado. Si no se reciben datos en el tiempo fijado se lanza la excepción *InterruptedIOException*