Programación de Servicios y Procesos

UNIDAD 6:
Sockets

Índice

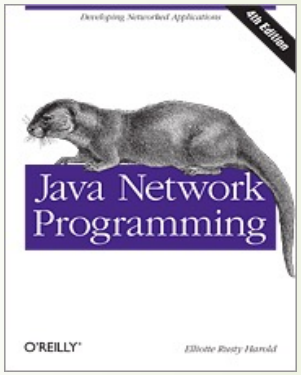
1. OBJETIVOS

- Estudiar los tipos de sockets
- *Conocer las caracterósticas del paquete java.net.*
- Implementar programas que se comunican utilizando sockets de java.
- Crear alplicaciones simples de cliente servidor.

2. Bibliografía

- ✓ Java Network Programming 3rd Edition. Descarga:

  http://it-ebooks.info/book/909/



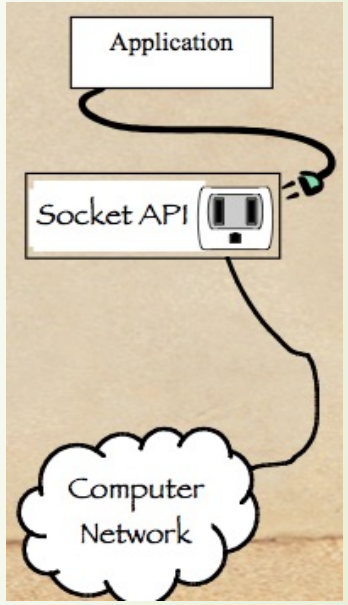
- ✓ Tutoriales de Oracle sobre Sockets:

  http://docs.oracle.com/javase/tutorial/networking/**sockets**/

3. SOCKETS
Introducción

Socket:

- Objeto abstracto.
- Representa los puntos finales de un canal de comunicación.
- Los hosts envían/reciben datos a través de ese canal.
- Independiente del lenguaje de programación.



Origen: Año 1981, introducido por primera vez en el v.4.2 Unix BSD. Se convirtió en la interfaz estándar para conectarse a Internet

3. SOCKETS
Tipos

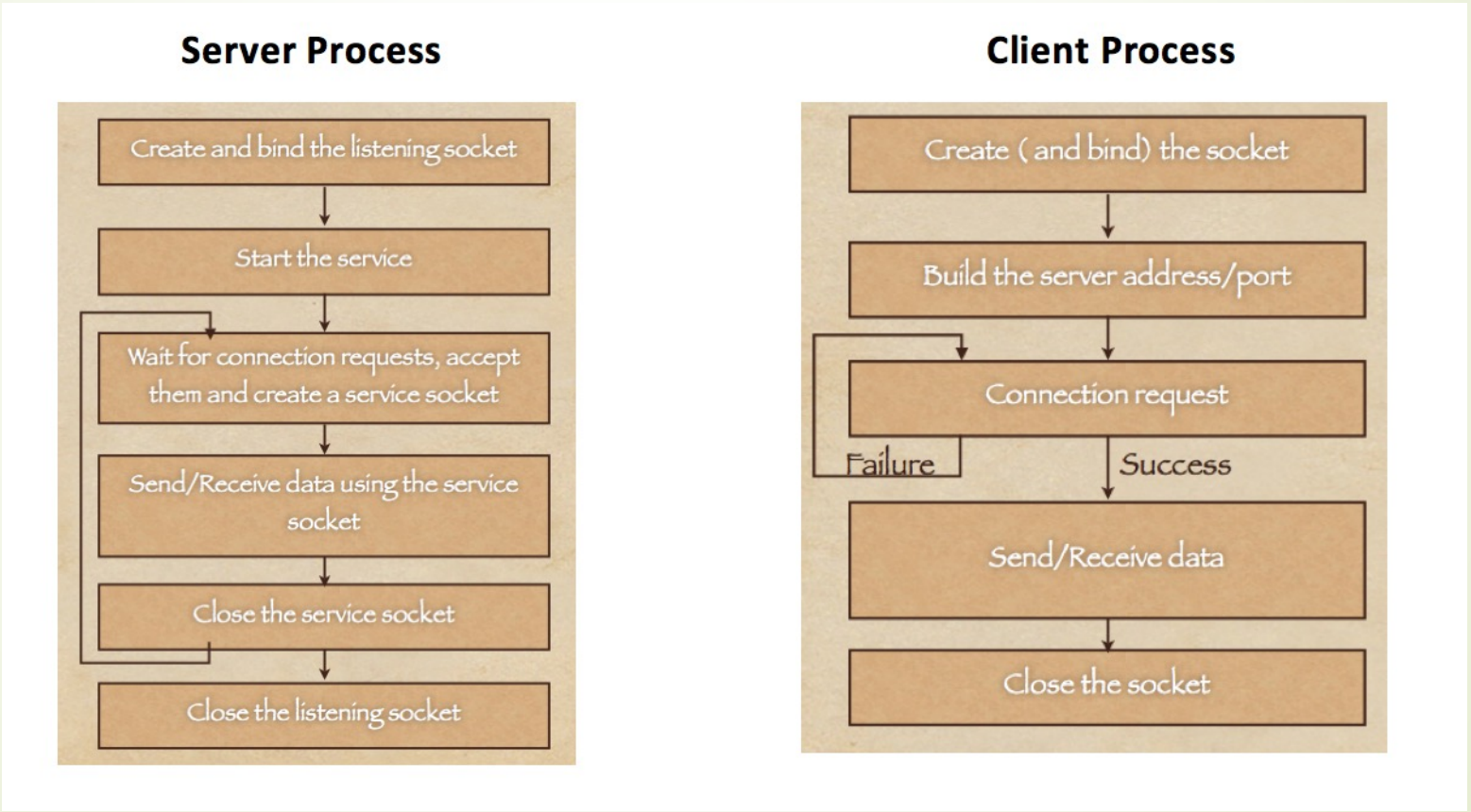
- Hay **dos tipos** de sockets dependiendo del protocolo de transporte que utilicen.:
  - **Stream Sockets** (TCP)
  - **Datagram Sockets** (UDP)

- Los sockets TCP y UDP, desempeñan el mismo papel, pero funcionan de manera diferente..
- Ambos reciben paquetes de protocolo de de los protocolos de Aplicación y pasan su contenido a la capa de Internet.

3. SOCKETS
Stream sockets – Flujo de mens.

- **Orientado a conexión**
- Utilice el protocolo de transporte **TCP**. **TCP** es **fiable**:
  - TCP divide los mensajes en paquetes y los vuelve a ensamblar en la **secuencia** correcta en el extremo receptor.
  - También se encarga de la solicitud de retransmisión de paquetes **perdidos**.
- La comunicación mediante **sockets de flujo** implica: **conexión**, **diálogo** y **desconexión**.
  - Un proceso de servidor crea un socket y espera una solicitud de conexión de proceso de cliente.
  - Cuando un proceso cliente quiere iniciar una comunicación, crea su socket conectado al servidor y se establece el canal de comunicación.
  - La comunicación termina cuando el cliente o servidor cierra su socket.

3. SOCKETS
Stream sockets - TCP



3. SOCKETS
Datagram sockets - UDP

- **Sin conexión**.
- Usar el protocolo **UDP**.
- **No es fiable**: UDP no realiza un seguimiento de los paquetes enviados/recibidos:
  - Las capas superiores deben asegurarse de que el mensaje está completo y ensamblado en la secuencia **correcta**.
- Los sockets de datagramas no distinguen entre el proceso del servidor y el del cliente.
- UDP es **más rápido** y **eficiente** que TCP (sin comprobación de errores de paquetes)
  - Perfecto si la cantidad de datos intercambiados a la vez es poca.
  - No es necesario volver a ensamblar datagramas para completar el mensaje.

3. SOCKETS
Uso de los Sockets

- TCP es perfecto para aplicaciones que **requieren alta fiabilidad**, y el tiempo de transmisión no es tan crítico.
  - HTTP, HTTPs, FTP, SMTP, Telnet,…
- UDP es adecuado para aplicaciones que necesitan una **transmisión rápida** y **eficiente**, como **juegos** en línea multijugador y **transmisión de video**.
  - DNS, DHCP, TFTP, VOIP, …
  - **Ejemplo**: Un servidor que recibe actualizaciones meteorológicas de muchas estaciones una vez por segundo.
    - Eso es una gran cantidad de datos, pero lo más importante es que cada nuevo mensaje de una sola estación meteorológica hace que el mensaje anterior sea obsoleto.
    - En este caso, si **perdemos algunos mensajes**, no pasaría nada.

4. SOCKETS IN JAVA

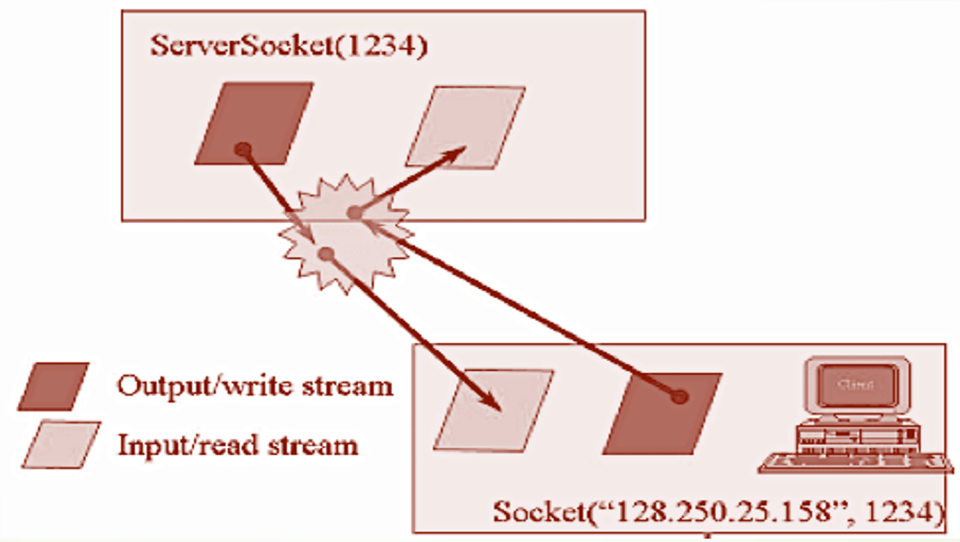
Classes		Interfaces	Exceptions
ContentHandler	***ServerSocket***	ContentHandlerFactory	BindException
DatagramPacket	***Socket***	FileNameMap	ConnectException
DatagramSocket	SocketImpl	SocketImplFactory	MalformedURLException
DatagramSocketImJpl	URL	URLStreamHandlerFactory	NoRouteToHostException
HttpURLConnection	URLConnection		ProtocolException
InetSocketAddress	URLEncoder		SocketException
MulticastSocket	URLStreamHandler		UnknownHostException
			UnknownServiceException

http://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html

4. SOCKETS IN JAVA

TCP sockets

- Clases Clave en Java: **ServerSocket** y **Socket**



- **Programa Servidor:** Crea un *ServerSocket*, que escuchas peticiones del cliente.
- Después de aceptar una petición del cliente, crea un nuevo Socket y se intercambia la información mediante **input/output streams**.

4. SOCKETS IN JAVA
TCP sockets

Server socket

Constructors

Constructor and Description
`ServerSocket()` Creates an unbound server socket.
`ServerSocket(int port)` Creates a server socket, bound to the specified port.
`ServerSocket(int port, int backlog)` Creates a server socket and binds it to the specified local port number, with the specified backlog.
`ServerSocket(int port, int backlog, InetAddress bindAddr)` Create a server with the specified port, listen backlog, and local IP address to bind to.

https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html#ServerSocket(int)

- Por defecto, el número máximo de peticiones de conexión (conocido como **backlog**) es por defecto **50**

4. SOCKETS IN JAVA
TCP sockets

Methods

Modifier and Type	Method and Description
`Socket`	**`accept()`** Listens for a connection to be made to this socket and accepts it.
`void`	**`bind(SocketAddress endpoint)`** Binds the `ServerSocket` to a specific address (IP address and port number).
`void`	**`bind(SocketAddress endpoint, int backlog)`** Binds the `ServerSocket` to a specific address (IP address and port number).
`void`	**`close()`** Closes this socket.

4. SOCKETS IN JAVA
TCP sockets

Client socket

Constructors

Modifier	Constructor and Description
	`Socket()` Creates an unconnected socket, with the system-default type of SocketImpl.
	`Socket(InetAddress address, int port)` Creates a stream socket and connects it to the specified port number at the specified IP address.
	`Socket(String host, int port)` Creates a stream socket and connects it to the specified port number on the named host.
	`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)` Creates a socket and connects it to the specified remote address on the specified remote port.

https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

4. SOCKETS IN JAVA
TCP sockets

Client socket

Methods

Modifier and Type	Method and Description
`void`	**`bind`**`(`**`SocketAddress`**` bindpoint)` Binds the socket to a local address.
`void`	**`close`**`()` Closes this socket.
`void`	**`connect`**`(`**`SocketAddress`**` endpoint)` Connects this socket to the server.
`void`	**`connect`**`(`**`SocketAddress`**` endpoint, int timeout)` Connects this socket to the server with a specified timeout value.
`int`	**`getLocalPort`**`()` Returns the local port number to which this socket is bound.
`int`	**`getPort`**`()` Returns the remote port number to which this socket is connected.
`InputStream`	**`getInputStream`**`()` Returns an input stream for this socket.
`OutputStream`	**`getOutputStream`**`()` Returns an output stream for this socket.

4. SOCKETS IN JAVA
TCP sockets

Actividad

- Implementa el **cliente-servidor** TCP que se encuentra en la docuentación.

4. SOCKETS IN JAVA
TCP sockets

Recomendaciones de transmisión de datos:

- ✓ Para transmitir solo datos de texto → puedes utilizar *InputStream* y *OutputStream*
- ✓ Para transmitir datos binarios → Has de utilizar los stream en *DataInputStream* y *DataOutputStream*.
- ✓ Para transmitir objetos serializados → debes utilizar *ObjectInputStream* y *ObjectOutputStream*.

4. SOCKETS IN JAVA
TCP sockets

PRÁCTICA :
Toc, Toc!

Lee la actividad 5 en la docuementación de la unidad y realiza las tareas que pide.

4. SOCKETS IN JAVA

UDP sockets

- TCP es **fiable**, pero a veces, como en las aplicaciones de transmisión, se puede tolerar la pérdida de paquetes, se recomienda el uso de un protocolo de transporte **más ligero y rápido**: protocolo UDP**.**
- El protocolo **UDP** no orientado a conexión utiliza paquetes de datagramas:
  - Contiene el mensaje y la información de destino necesaria para transferir el mensaje.

    | Msg | length | Host | serverPort |
  - Puede enrutarse de manera diferente y llegar en cualquier orden.
  - La entrega no está garantizada.

4. SOCKETS IN JAVA
UDP sockets

- Java admite la comunicación de datagramas a través de las siguientes clases:
  - **DatagramPacket**
  - **DatagramSocket**

4. SOCKETS IN JAVA
UDP sockets

Datagram Packet

https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html

Constructors

Constructor and Description
`DatagramPacket``(byte[] buf, int length)` Constructs a `DatagramPacket` for receiving packets of length `length`.
`DatagramPacket``(byte[] buf, int length,` **`InetAddress`** `address, int port)` Constructs a datagram packet for sending packets of length `length` to the specified port number on the specified host.
`DatagramPacket``(byte[] buf, int length,` **`SocketAddress`** `address)` Constructs a datagram packet for sending packets of length `length` to the specified port number on the specified host.

4. SOCKETS IN JAVA
UDP sockets

Datagram Packet

Methods

Modifier and Type	Method and Description
`InetAddress`	`getAddress()` Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
`byte[]`	`getData()` Returns the data buffer.
`int`	`getLength()` Returns the length of the data to be sent or the length of the data received.
`int`	`getPort()` Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
`SocketAddress`	`getSocketAddress()` Gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from.

4. SOCKETS IN JAVA
UDP sockets

DatagramSocket

Constructors

Modifier	Constructor and Description
	DatagramSocket() Constructs a datagram socket and binds it to any available port on the local host machine.
protected	**DatagramSocket(DatagramSocketImpl impl)** Creates an unbound datagram socket with the specified DatagramSocketImpl.
	DatagramSocket(int port) Constructs a datagram socket and binds it to the specified port on the local host machine.
	DatagramSocket(int port, InetAddress laddr) Creates a datagram socket, bound to the specified local address.
	DatagramSocket(SocketAddress bindaddr) Creates a datagram socket, bound to the specified local socket address.

4. SOCKETS IN JAVA
UDP sockets

DatagramSocket

Methods

Modifier and Type	Method and Description
void	**receive**(**DatagramPacket** p) Receives a datagram packet from this socket.
void	**send**(**DatagramPacket** p) Sends a datagram packet from this socket.

→**Receive**: El mensaje se trunca si su longitud es mayor que la longitud del paquete.

void	**setSoTimeout**(int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.

4. SOCKETS IN JAVA
UDP sockets

Actividad

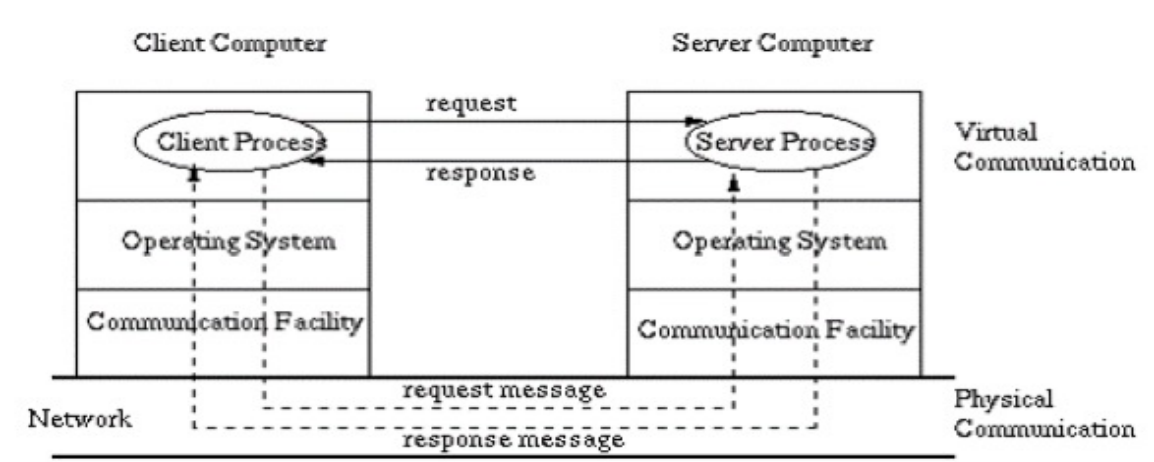
- Implemente el ejemplo de **cliente/servidor UDP** que encontrarás en la documentación de la unidad.
- Cuando haya terminado, modifique los programas después de la 6ª Actividad - Think Over.

5. COMMUNICATION MODELS

- Un **modelo de comunicacines** especifica cómo se comunican entre sí los distintos elementos de una aplicación distribuida.

- Los más populares son:
  - Modelo Cliente-Servidor
  - Peer-to-peer
  - Modelos híbridos
  -

5. COMMUNICATION MODELS
Client-Server



The client-server model

El modelo cliente-servidor es **sencillo** de implementar (utilizando por ejemplo sockets TCP o UDP), **modular, extensible y flexible.**

Example: Servidor Web a la espera de solicitudes de navegadores de Internet

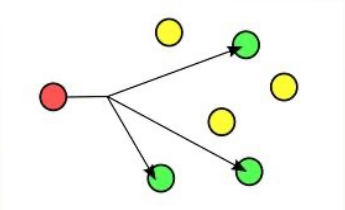
5. COMMUNICATION MODELS Client-Server

- **Ventajas**
  - Reduce el volumen de tráfico de datos en la red.
  - Proporciona respuestas más rápidas al cliente.
  - Permite utilizar clientes informáticos menos costosos: la mayor parte del trabajo lo realiza el servidor.
  - **Desventajas**
  - Los servidores son costosos.
  - Toda la red se ve afectada si hay algún problema en el equipo servidor.

5. COMMUNICATION MODELS

Group Communication

- A veces, la comunicación involucra **múltiples receptores**, no solo dos como sucede en la comunicación cliente-servidor.
- **Solución**: enviar el mensaje a cada receptor.
- La **comunicación de grupo** permite enviar un mensaje a varios receptores en una operación llamada **multicast.**
  - Para recibir un mensaje de multidifusión, los sockets de los receptores del grupo deben utilizar la misma IP de multidifusión (224.0.0.0 to 239.255.255.255).



  - La multidifusión conserva el ancho de banda al obligar a la red a realizar la replicación de paquetes solo cuando sea necesario.
  - Es perfecto para: cotizaciones de acciones en directo, videoconferencias multiparte, aplicaciones de pizarra compartida, herramientas de clonación masiva de discos, ...

5. COMMUNICATION MODELS
Group Communication

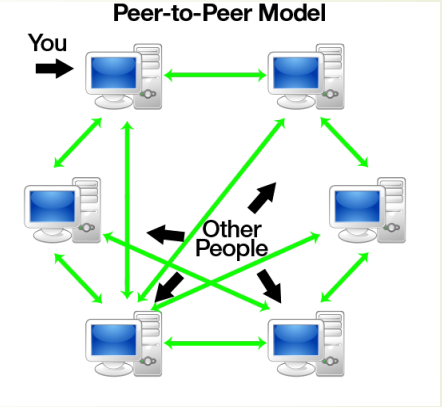
Actividad: Mensajes Multicast

- Realiza el ejercicio **número 9** de los apuntes.
- Busca la información necesaria y **contesta** a las preguntas.

5. COMMUNICATION MODELS

Peer-to-Peer

- La computación **Peer-to-peer** (**P2P**) es una arquitectura de aplicación distribuida que divide tareas o cargas de trabajo entre pares.
- Se dice que forman una **red peer-to-peer** de nodos (computadoras)
- Los pares son son igualmente privilegiados.
  - Los pares son tanto **proveedores** como **consumidores** de recursos: no hay necesidad de coordinación central por parte de los servidores.
- Ejemplo: el intercambio de archivos, que popularizó la tecnología, como los famosos **Torrents.**



5. COMMUNICATION MODELS Peer-to-Peer

- **Ventajas**
  - Es útil en oficinas pequeñas.
  - Es fácil de diseñar y mantener.
  - No requiere ningún ordenador potente.
- **Desventajas**
  - Se vuelve lento bajo un uso intensivo.
  - No hay un lugar central para almacenar datos y software.
  - No proporciona la seguridad para proteger los datos almacenados en los equipos conectados en la red.

5. COMMUNICATION MODELS

Modelos híbridos

- **Combinación** de modelos de red punto a punto y cliente-servidor.
  - Un modelo híbrido común tiene servidores centrales que ayudan a los pares a encontrarse entre sí..
  - Requiere hardware menos potente y costoso.
  - Los usuarios comparten los datos y el software.
  - Cada nodo puede almacenar sus propios archivos de datos y programas.
- Ejemplo1: **Spotify** es un ejemplo de un modelo híbrido.
- Ejemplo2: **Juegos multijugador masivos en línea (MMO)**
-