Laboratorio de Redes de Comunicaciones
Programación con Sockets en Java

Sockets orientados a conexión

Sockets no orientados a conexión

UNIVERSIDADE DA CORUÑA

Java: Distribuido

- Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como *http* y *ftp*. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

- Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

Java: Distribuido

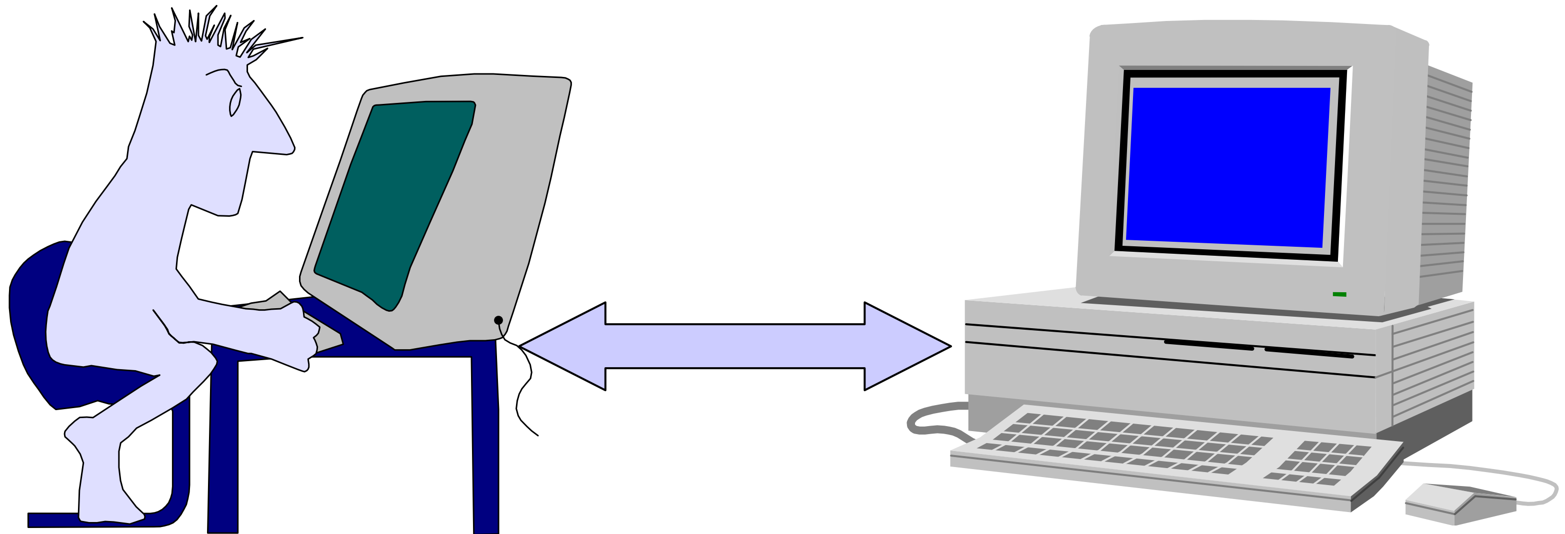
- Pensado para su aplicación en redes.
- Soporta varios niveles de conectividad en red:
  - Permite abrir un URL
  - RMI
  - Permite trabajar con "**sockets**"
- Permite crear de forma sencilla, tanto aplicaciones cliente como servidor.

Clientes y servidores

- Aplicaciones Cliente/Servidor
- Cliente:
  - Programa que ejecuta el usuario
  - Solicita un servicio a una máquina
- Servidor:
  - Ofrece un servicio a múltiples clientes
- Ejemplos: telnet, ftp, web, echo

Ejemplo: Servicio de telnet

Cliente **Servidor**



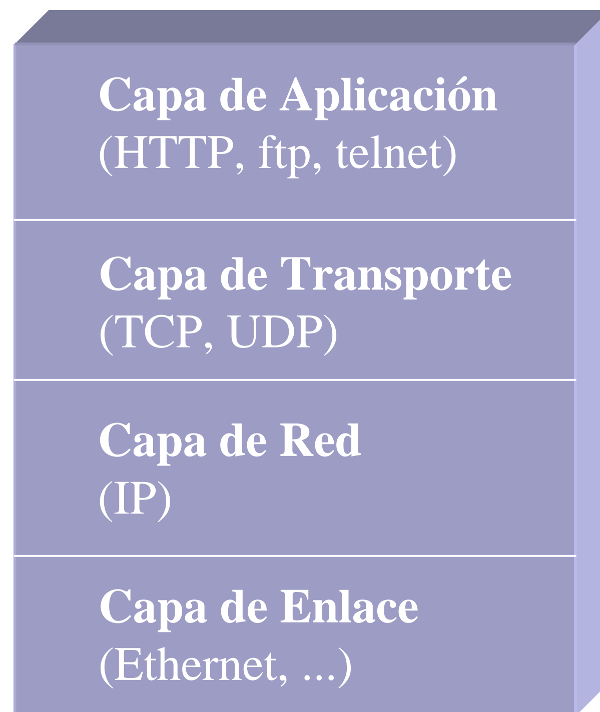
Ejecuta un telnet

Servidor de telnet: telnetd

UNIVERSIDADE DA CORUÑA

Conceptos básicos de redes

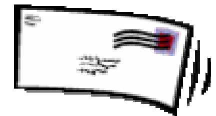
- En Internet se utiliza la pila de protocolos TCP/IP para el establecimiento y realización de conexiones, basado en un conjunto de protocolos organizados en diferentes niveles: Enlace / Red / Transporte / Aplicación.
- Normalmente, cuando se escriben aplicaciones Java en Red se programa a nivel de aplicación.
- Es posible realizar programas a más bajo nivel utilizando la API java.net => nivel de transporte => TCP /UDP.



UNIVERSIDADE DA CORUÑA

Capa de Transporte: TCP vs UDP

- TCP: Transmission Control Protocol
  - Protocolo orientado a conexión
  - Provee un flujo de bytes fiable entre dos ordenadores.
    - Llegada en orden, correcta, sin perder nada.
  - Protocolos del nivel de aplicación que usan TCP: telnet, http, ftp.



- UDP: User Datagram Protocol
  - No orientado a conexión.
  - Envía paquetes de datos (datagramas) independientes sin garantías.
  - Protocolos del nivel de aplicación que usan UDP: tftp, ping.
  - Permite broadcast.

UNIVERSIDADE DA CORUÑA

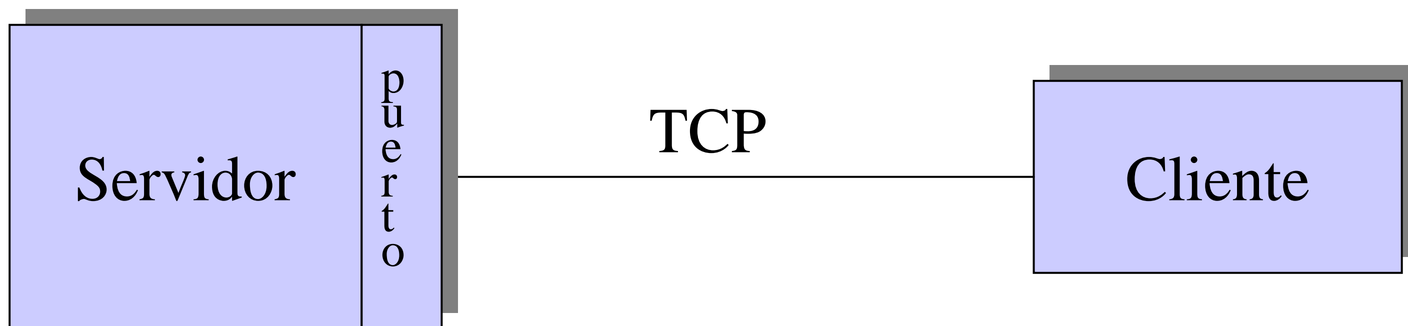
¿Qué es un puerto?

- Un ordenador tiene una o varias conexiones físicas a la red.
- A través de esas conexiones recibe los datos dirigidos a la máquina.
- ¿Cómo determinar a que aplicación enviar los datos? ➔ Puertos.
  - TCP y UDP utilizan los puertos para dirigir los datos a la aplicación correcta de entre todas las que se estén ejecutando en la máquina.
- Los datos transmitidos a través de Internet contienen información de direccionamiento que identifica a la **máquina** y **puerto** a los que van dirigidos.
  - La máquina se identifica a través de una dirección IP de 32 bits.
  - Los puertos se identifican por un número de 16 bits.
- Puertos:
  - Independientes para TCP y UDP.
  - 16 bits ➔ Rango: 0 a 65535.
  - Reservados: 0 a 1023 (“*well known ports*”)
    - Para servicios conocidos: HTTP,FTP, …
    - No deberían ser utilizados por aplicaciones de usuario.

UNIVERSIDADE DA CORUÑA

TCP

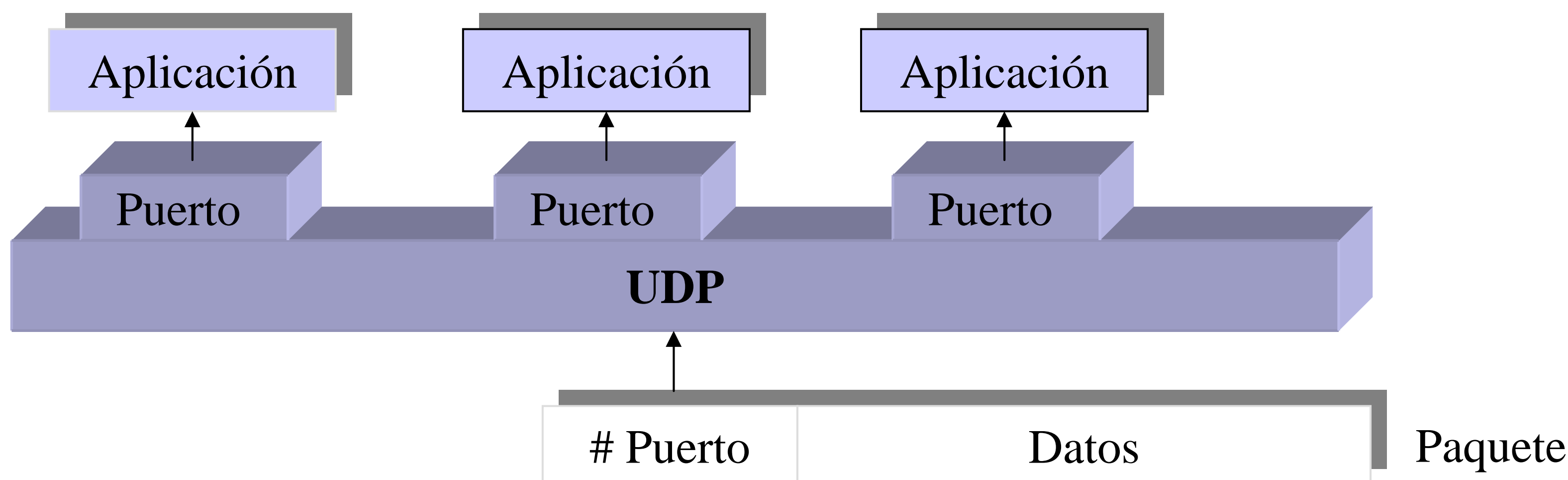
- Una aplicación servidora se registra en un puerto concreto.
  - El servidor se registra en el sistema para recibir los datos dirigidos a ese puerto.
- El cliente se conecta con el servidor usando ese número de puerto.
  - Sólo en el **establecimiento** de la conexión se precisa la **IP+puerto**.
  - El resto de paquetes TCP sólo llevan un **identificador de la conexión**.



UNIVERSIDADE DA CORUÑA

UDP

- Una aplicación servidora se registra en un puerto concreto.
  - El servidor se registra en el sistema para recibir los datos dirigidos a ese puerto.
- El cliente envía datagramas que contienen el número de puerto del destino asociado a la aplicación servidora.
  - UDP enruta hacia la aplicación adecuada.
  - En cada paquete UDP va toda la información necesaria para que enrute: **IP+puerto**



Clases para Redes en JDK

- Paquete java.net
  - Clases Java para crear programas que utilicen TCP o UDP para comunicarse sobre Internet.
  - Clases URL, URLConnection, Socket, ServerSocket
    - Utilizan TCP para comunicaciones de red.
  - Clases DatagramPacket, DatagramSocket, MulticastSocket
    - Utilizan UDP para comunicaciones de red.

- Niveles de comunicación de red
  - Bajo nivel: Aplicaciones cliente/servidor basadas en protocolos.
  - Alto nivel: Acceso a recursos de red

¿Qué es un socket?

- Es un extremo de un enlace de comunicación bidireccional entre dos programas que se comunican por la red.
  - Un socket se asocia a un número de puerto.

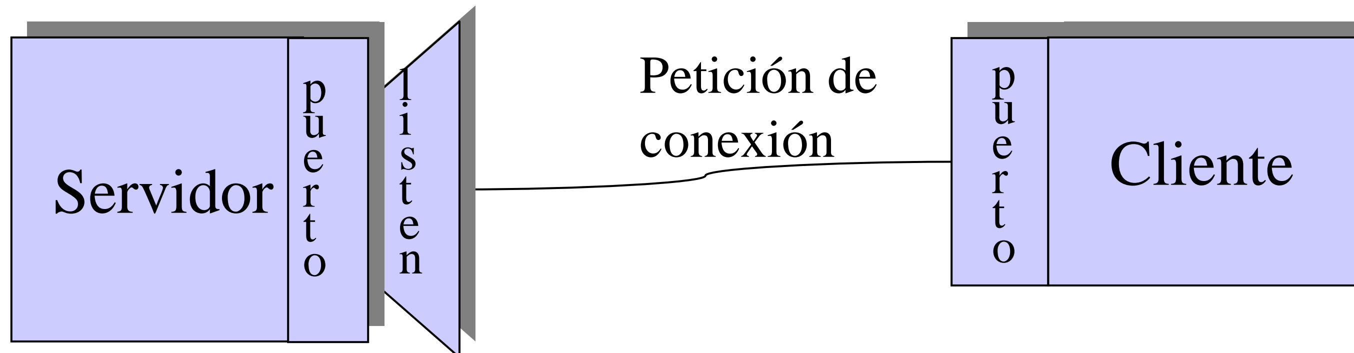
- Se identifica por dirección IP de la máquina + número de puerto.

- Existen en TCP y UDP.

UNIVERSIDADE DA CORUÑA

Sockets TCP (1)

- Sockets orientados a conexión.
- El servidor se ejecuta en una máquina y crea un socket orientado a conexión ligado a un número de puerto específico.
  - El servidor espera, escuchando por ese socket, a que los clientes hagan peticiones de conexión.
- El cliente conoce:
  - La máquina donde se está ejecutando el servidor
  - El puerto donde el servidor está escuchando.

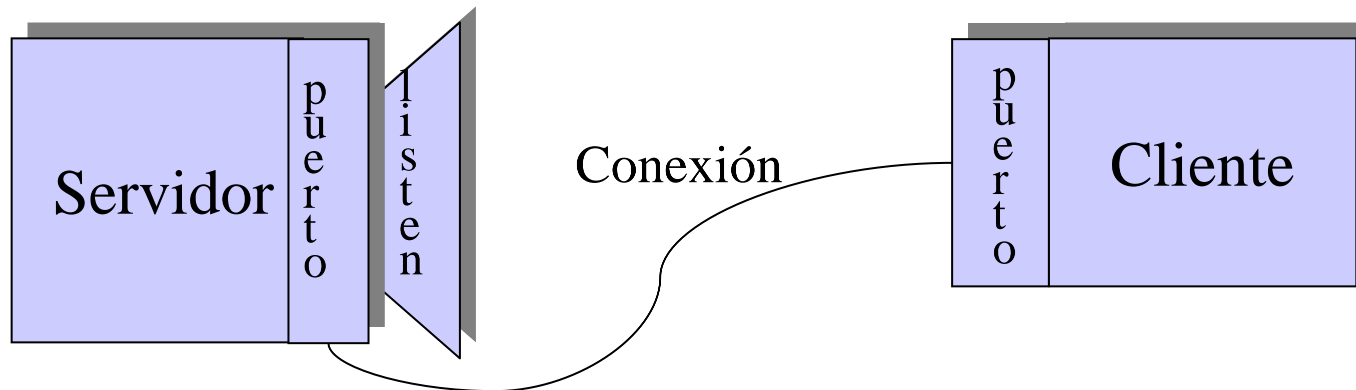


Sockets TCP (2)

- El cliente envía una petición de conexión al servidor, usando esa máquina y número puerto.
- Además, el cliente tiene que identificarse ante el servidor, por tanto al cliente se le asigna un puerto en su máquina, que será utilizado a lo largo de la conexión.
  - Normalmente esta asignación la realiza el sistema.
- Si todo va bien el servidor acepta la conexión.
  - El servidor obtiene un nuevo socket asociado al mismo puerto local que el original, y que tiene como otro extremo de la conexión la dirección y puerto del cliente.
  - Es necesario crear un nuevo socket para atender al cliente, para poder seguir recibiendo peticiones de conexión a través del socket original.

Sockets TCP (3)

- En el lado cliente, si la conexión es aceptada, se obtiene un socket conectado con el servidor.
- A partir de ahí, cliente y servidor se comunican escribiendo y leyendo por sus respectivos sockets.



- Clases:
  - java.net.Socket: Implementa un extremo de una conexión bidireccional.
  - java.net.ServerSocket: Implementa un socket que los servidores pueden utilizar para escuchar y aceptar peticiones de clientes.

UNIVERSIDADE DA CORUÑA

Sockets UDP (1)

- Sockets NO orientados a conexión.
  - Se envían y reciben paquetes independientes de información
  - Clientes y servidores NO se conectan
  - NO se garantiza la recepción del paquete ni el orden.
- Datagrama:
  - Mensaje independiente, enviado a través de una red cuya llegada, tiempo de llegada y contenido no está garantizado.

- El servidor se ejecuta en una máquina y crea un socket no orientado a conexión que está ligado a un número de puerto específico.
  - A partir de ahí el servidor recibe todos los datos enviados a ese puerto UDP, leyendo a través de ese socket.
- El cliente conoce:
  - La máquina donde se está ejecutando el servidor
  - El puerto asociado al servidor.

Sockets UDP (2)

- El cliente crea un socket no orientado a conexión ligado a un número de puerto específico y lo utiliza para enviar Datagramas al servidor.
  - Normalmente la asignación del número de puerto en el cliente la hace el sistema.
  - Los Datagramas enviados deben contener la dirección y número de puerto del servidor al que van dirigidos.
- Cuando un servidor recibe un Datagrama:
  - Obtiene la dirección de la máquina del cliente y su número de puerto.
  - Envía la respuesta al cliente creando un Datagrama dirigido a esa máquina y puerto.
- Clases:
  - java.net.DatagramSocket
  - java.net.DatagramPacket

UNIVERSIDADE DA CORUÑA

Direcciones IP

- Clase java.net.InetAddress:
  - Métodos estáticos:
    - InetAddress getByName(String host)
      - Obtiene la dirección IP de la máquina
      - Recibe el nombre de la máquina o su dirección IP como cadena.
    - InetAddres[] getAllByName(String host)
      - Otiene todas las direcciones IP de una máquina.
      - Recibe el nombre de la máquina o su dirección IP como cadena.
    - InetAddress getLocalHost():
      - Obtiene la dirección IP de la máquina en la que se está ejecutando.
  - Método:
    - public String getHostName()
      - Devuelve el nombre de la máquina correspondiente a esta IP.

TCP - Cliente (1)

- Operación:
  - Crear un socket
  - Establecer la conexión con el servidor
  - Intercambio de datos (según protocolo):
    - Abrir un flujo de entrada y otro de salida
    - Escribir y leer de los flujos
  - Cerrar los flujos
  - Cerrar el socket
- Clase: java.net.Socket

UNIVERSIDADE DA CORUÑA

TCP - Cliente (2)

- Creación y conexión del socket (java.net.Socket).
  - Al llamar al constructor se **crea** el socket y si se indica la dirección y puerto del servidor ya se **conecta** con la máquina y puerto indicados.
  - Constructores:
    - Socket()
    - Socket(InetAddress dir, int puerto)
    - Socket(InetAddress dir, int puerto, InetAddress dirLocal, int puertoLocal)
    - Socket(String nombre, int puerto)
    - Socket(String nombre, int puerto, InetAddress dirLocal, int puertoLocal)
  - Métodos
    - void bind(SocketAddress bindpoint)
    - void connect(SocketAddress endpoint)
    - void connect(SocketAddress endpoint, int timeout)

UNIVERSIDADE DA CORUÑA

TCP - Cliente (3)

- Abrir flujos de datos del socket:
  - Métodos:
    - InputStream getInputStream()
    - OutputStream getOutputStream()
  - Recomendación (si se transmite TEXTO):
    - Para la entrada de datos: BufferedReader – InputStreamReader
      - Método readLine().
    - Para la salida de datos: PrintWriter
      - Método println().

UNIVERSIDADE DA CORUÑA

TCP - Cliente (4)

- Cierre de los flujos de lectura y escritura:
  - Método:
    - void close()
- Cierre del socket:
  - Método:
    - void close()

UNIVERSIDADE DA CORUÑA

TCP - Servidor (1)

- Operación:
  - Crear un socket servidor
  - Mientras (dure la ejecución del servidor)
    - Esperar conexión de un cliente.
      - Crear un socket conectado al cliente.
      - Intercambio de datos (según protocolo). En otro hilo de ejecución?
        - » Abrir un flujo de entrada y otro de salida
        - » Leer y escribir de los flujos
      - Cerrar los flujos
      - Cerrar el socket
  - Cerrar el socket servidor

- Clases:
  - java.net.ServerSocket
  - java.net.Socket

TCP - Servidor (2)

- Creación del socket servidor (java.net.ServerSocket)
  - Al llamar al constructor se **crea** el socket servidor y si se indica el puerto ya se asocia a ese puerto.
  - Constructores:
    - ServerSocket()
    - ServerSocket(int puerto)
    - ServerSocket(int puerto, int backlog)
      - backlog: Número de máximo de conexiones pendientes que aceptará el socket.
    - ServerSocket(int puerto, int backlog, InetAddress dirIP)
      - dirIp: Dirección por la que va a aceptar conexiones (en caso de que la máquina del servidor tenga más de una).
  - Métodos:
    - void bind(SocketAddress endpoint)
    - void bind(SocketAddress endpoint, int backlog)

TCP - Servidor (3)

- Esperar y aceptar conexiones:
  - Método:
    - Socket accept()
      - Devuelve un Socket conectado al cliente que realizó la conexión.
- Abrir un flujo de entrada y otro de salida
  - Ídem cliente.
- Leer y escribir en los flujos
  - Ídem cliente.

UNIVERSIDADE DA CORUÑA

TCP - Servidor (4)

- Cerrar los flujos
  - Ídem cliente.
- Cerrar el socket
  - Ídem cliente.
- Cerrar el socket servidor.
  - Método:
    - void close()

UNIVERSIDADE DA CORUÑA

UDP - Cliente (1)

- Operación:
  - Crear un socket no orientado a conexión.
  - Intercambio de datos (según protocolo):
    - Enviar y recibir Datagramas
  - Cerrar el socket

- Clases:
  - java.net.DatagramSocket
  - java.net.DatagramPacket

UDP - Cliente (2)

- Creación del socket (java.net.DatagramSocket)
  - Constructores:
    - DatagramSocket()
      - Normalmente se utiliza este en los clientes
    - DatagramSocket(int puerto)
    - DatagramSocket(int puerto, InetAddress dirIP)
- Intercambio de datos
  - Enviar:
    - void send(DatagramPacket)
  - Recibir:
    - void receive(DatagramPacket)
  - Establecer timeout para recibir.
    - setSoTimeout(int milisegundos)
- Cierre del socket:
  - Método:
    - void close()

UDP – Cliente (3)

- Datagramas (java.net.DatagramPacket):
  - Constructores (para enviar):
    - DatagramPacket(byte[] buf, int length, InetAddress addr, int port)
    - DatagramPacket(byte[] buf, int offset, int length, InetAddress addr, int port)
      - buf : array con los datos que se van a enviar
      - offset: desplazamiento dentro del array de datos a enviar
      - length: número de bytes a enviar
      - addr: dirección IP del destino
      - port: número de puerto del destino
  - Constructores (para recibir):
    - DatagramPacket(byte[] buf, int length)
    - DatagramPacket(byte[] buf, int offset, int length)
      - buf : array para almacenar los datos que se van a recibir
      - offset: desplazamiento dentro del array de datos a recibir
      - length: número máximo de bytes que se van a recibir

UDP - Servidor (1)

- Operación:
  - Crear un socket no orientado a conexión.
  - Mientras (dure la ejecución del servidor)
    - Intercambio de datos (según protocolo):
      - Recibir y enviar Datagramas
  - Cerrar el socket

- Clases:
  - java.net.DatagramSocket
  - java.net.DatagramPacket

UNIVERSIDADE DA CORUÑA

UDP - Servidor (2)

- Creación del socket (java.net.DatagramSocket)
  - Constructores:
    - DatagramSocket()
    - DatagramSocket(int puerto)
    - DatagramSocket(int puerto, InetAddress dirIP)
  - Métodos:
    - void bind(SocketAddress addr)
- Intercambio de datos
  - Ídem cliente.
- Cierre del socket
  - Ídem cliente.

Laboratorio de Redes de Comunicaciones
Flujos de datos en Java

Flujos de datos

Ficheros

UNIVERSIDADE DA CORUÑA

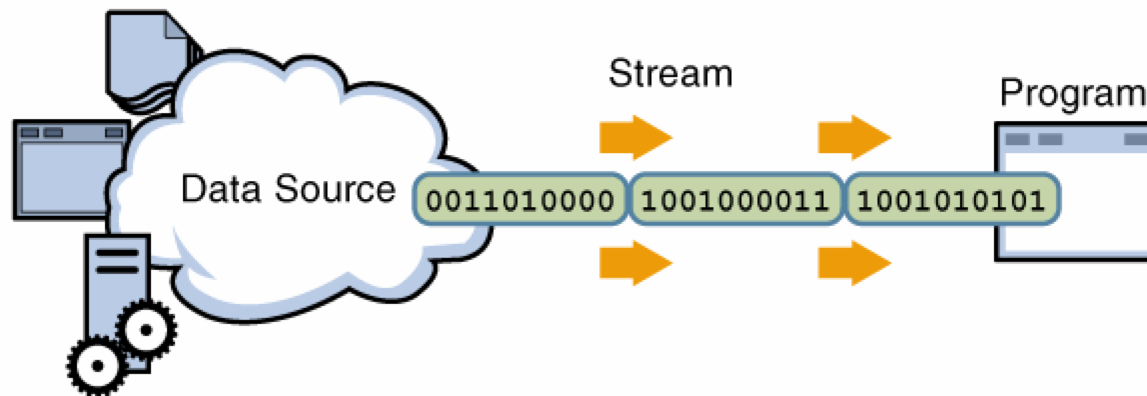
Flujos de datos: Introducción

- Los programas necesitan:
  - Recoger información de una fuente externa.
  - Enviar información a un destino externo.
- Esta información puede estar en: ficheros en disco, **la red**, memoria, programas, etc.
- Y puede ser de cualquier tipo: objetos, caracteres, imágenes, sonidos, etc.

UNIVERSIDADE DA CORUÑA

Flujos de datos: Lectura

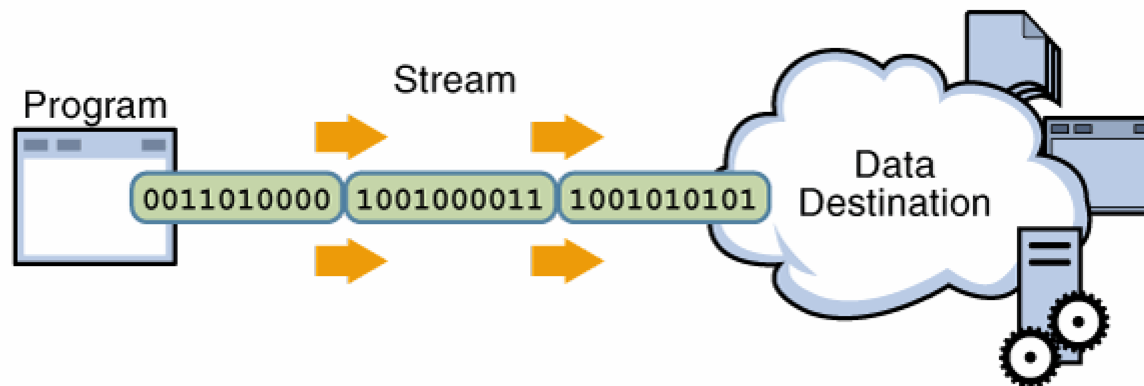
- El programa abre el flujo (*stream*) de datos en la fuente.
- El programa lee la información.



UNIVERSIDADE DA CORUÑA

Flujos de datos: Escritura

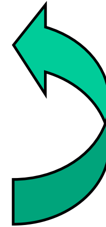
- El programa abre un flujo (*stream*) de datos con el destino.
- Escribe la información.



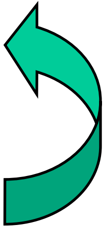
UNIVERSIDADE DA CORUÑA

Flujos de datos

- Lectura
  - *Abrir el flujo de datos*
  - *Mientras hay información*
    - *Leer información*
  - *Cerrar el flujo de datos*



- Escritura
  - *Abrir el flujo de datos*
  - *Mientras hay información*
    - *Escribir información*
  - *Cerrar el flujo de datos*



UNIVERSIDADE DA CORUÑA

Flujos de datos: Clases

- Paquete java.io
- Existen 4 clases abstractas padre del resto de clases:
  - Flujos de bytes
    - java.io.OutputStream, java.io.InputStream
  - Flujos de caracteres
    - java.io.Reader, java.io.Writer
    - Normalmente son envoltorios sobre flujos de bytes.
      - El flujo de bytes se utiliza para realizar la E/S física, y el flujo de caracteres realiza la conversión entre caracteres y bytes.
    - La plataforma Java almacena los caracteres utilizando convenciones Unicode.
      - Los flujos de caracteres traducen entre este formato interno y el juego de caracteres local.

UNIVERSIDADE DA CORUÑA

Flujos de bytes. Lectura.

- Clase java.io.InputStream:
  - Lee BYTES uno a uno o en un array.
  - Métodos:
    - int read()
      - Lee un byte
    - int read(byte buf[])
      - Lee un conjunto de bytes y los almacena en un array
      - Devuelve el número de bytes leídos
    - int read(byte buf[], int offset, int longitud)
      - Lee un conjunto de bytes y los almacena en una porción de un array
      - Devuelve el número de bytes leídos

      Devuelven –1 en caso de llegar al final del flujo de datos.
    - void close()
      - Cierra el flujo

UNIVERSIDADE DA CORUÑA

Flujos de bytes. Escritura.

- Clase java.io.OutputStream:
  - Escribe BYTES uno a uno o a través un array.
  - Métodos:
    - void write (int c)
      - Escribe un byte
    - void write(byte buf[])
      - Escribe un array de bytes
    - write(byte buf[], int offset, int longitud)
      - Escribe una porción de un array de bytes
    - void close()
      - Cierra el flujo

Flujos de caracteres. Lectura.

- Clase java.io.Reader:
  - Lee CARACTERES uno a uno o en un array.
  - Métodos:
    - int read()
      - Lee un carácter
    - int read(char buf[])
      - Lee un conjunto de caracteres y los almacena en un array
      - Devuelve el número de caracteres leídos
    - int read(char buf[], int offset, int len)
      - Lee un conjunto de caracteres y los almacena en una porción de un array
      - Devuelve el número de caracteres leídos

    Devuelven –1 en caso de llegar al final del flujo de datos.
    - void close()
      - Cierra el flujo

UNIVERSIDADE DA CORUÑA

Flujos de caracteres. Escritura.

- Clase java.io.Writer:
  - Escribe CARACTERES uno a uno o a través un array.
  - Métodos:
    - void write (int c)
      - Escribe un carácter
    - void write(char buf[])
      - Escribe un array de caracteres
    - void write(char buf[], int offset, int longitud)
      - Escribe una porción de un array de caracteres
    - void close()
      - Cierra el flujo

Flujos de datos “orientados a líneas”:Lectura

- Clase java.io.BufferedReader:
  - Clase útil para la lectura de líneas de texto.
  - Constructor:
    - BufferedReader(Reader reader)
  - Métodos:
    - String readLine()
      - Lee una línea de texto.
      - Devuelve la cadena leída o *null* si se alcanza el final del flujo.
    - void close()
      - Cierra el flujo

UNIVERSIDADE DA CORUÑA

Flujos de datos “orientados a líneas”:Escritura

- Clase java.io.PrintWriter:
  - Clase útil para escribir líneas de caracteres.
  - Constructores:
    - PrintWriter(OutputStream out)
    - PrintWriter(OutputStream out, boolean autoFlush)
    - PrintWriter(Writer out)
    - PrintWriter(Writer out, boolean autoFlush)
      - *autoFlush* indica si ciertos métodos (incluido println) provocan un *flush* del buffer de escritura (por defecto *false)*
  - Métodos:
    - void println(String x)
      - Escribe la cadena especificada más un fin de línea
    - void flush()
      - Escribe lo que haya en el buffer de escritura
    - void close()
      - Cierra el flujo

Conversión entre flujos de datos

- Clase java.io.InputStreamReader:
  - Clase útil para la conversión de objetos InputStream a Reader.
  - Hereda de Reader.
  - Constructor:
    - InputStreamReader(InputStream in)

- Clase java.io.OutputStreamWriter:
  - Clase útil para la conversión de objetos OutputStream a Writer.
  - Hereda de Writer.
  - Constructor:
    - OutputStreamWriter(OutputStream out)

Ficheros

- Unidad de almacenamiento contenida en: discos, cintas, etc.
- Clase para manejo de ficheros y directorios: java.io.File
- Constructores:
  - File(File parent,String child)
  - File(String pathname)
  - File(String parent,String child)

UNIVERSIDADE DA CORUÑA

Clase java.io.File (1)

- Métodos
  - String getName()
    - Devuelve el nombre del fichero o directorio
  - String getParent()
    - Devuelve el path del directorio padre o null si no tiene.
  - String getPath()
    - Devuelve el path
  - String getAbsolutePath()
    - Devuelve el path absoluto
  - long lastModified()
    - Devuelve la hora de la última modificación.
  - long length()
    - Devuelve el tamaño.
  - boolean canExecute()
  - boolean canRead()
  - boolean canWrite()
    - Indican si la aplicación puede ejecutar, leer y escribir en el fichero.

UNIVERSIDADE DA CORUÑA

Clase java.io.File (2)

- Métodos
  - boolean delete()
    - Borra el fichero o directorio
  - boolean renameTo(File dest)
    - Renombra el fichero o directorio
  - boolean exists()
    - Indica si existe un fichero o directorio con ese nombre
  - boolean isDirectory()
    - Indica si se trata de un directorio.
  - boolean isFile()
    - Indica si se trata de un fichero
  - boolean createNewFile()
    - Crea un fichero vacío con este nombre si no existía ya.
  - boolean mkdir()
    - Crea un directorio con este nombre
  - boolean mkdirs()
    - Crea un directorio con este nombre creando todos los directorios padre que sea necesario
  - String[] list()
    - Obtiene una lista de los ficheros y directorios contenidos en este directorio.

Ficheros. Flujos de bytes: Lectura

- Clase java.io.FileInputStream:
  - Lee BYTES de un fichero. Acceso secuencial.
- Apertura de un FileInputStream:
  - A través del constructor:
    - FileInputStream(File file)

      ```
      File miFichero = new File("/etc/ejemplo");
      FileInputStream fis = new FileInputStream(miFichero);
      ```
    - FileInputStream(String name)

      ```
      FileInputStream fis = new FileInputStream("/etc/ejemplo");
      ```
- Lectura y cierre: Mismos métodos que InputStream.

Ficheros. Flujos de bytes: Escritura

- Clase java.io.FileOutputStream:
  - Escribe BYTES en un fichero.

- Apertura de un FileOutputStream:
  - A través del constructor:
    - FileOutputStream(File file)
    - FileOutputStream(File file, boolean append)
    - FileOutputStream(String name)
    - FileOutputStream(String name, boolean append)
      - *append* indica si lo que se escribe se añade al final del fichero

- Escritura y cierre: Mismos métodos que OutputStream.

Ficheros. Flujos de caracteres: Lectura

- Clase java.io.FileReader:
  - Lee CARACTERES de un fichero. Acceso secuencial.

- Apertura de un FileReader:
  - A través del constructor:
    - FileReader(File file)
    - FileReader(String name)

- Lectura y cierre: Mismos métodos que Reader.

Ficheros. Flujos de caracteres: Escritura

- Clase java.io.FileWriter:
  - Escribe CARACTERES en un fichero.

- Apertura de un FileWriter:
  - A través del constructor:
    - FileWriter(File file)
    - FileWriter(File file, boolean append)
    - FileWriter(String name)
    - FileWriter (String name, boolean append)
      - *append* indica si lo que se escribe se añade al final del fichero

- Escritura y cierre: Mismos métodos que Writer.

Streams de E/S estándar

- Entrada estándar.
  - Para leer datos introducidos por el usuario.
  - System.in
  - De tipo *InputStream.*

- Salida estándar y salida de error.
  - Para mostrar información y mensajes de error al usuario.
  - System.out, System.err
  - De tipo *PrintStream.*
    - Desciende de java.io.OutputStream
    - Tiene métodos para escribir caracteres (*print*, *println)* que aceptan parámetros de diferentes tipos (*Object*, *String*, *char*[], *char*, *int*, *long*, *float*, *double*, y *boolean*)
    - Tiene métodos para escribir bytes: métodos *write* de OutputStream.

Laboratorio de Redes de Comunicaciones
Programación Concurrente

Threads

UNIVERSIDADE DA CORUÑA

UNIVERSIDADE DA CORUÑA

Programación Multithread

- La programación multihilo permite realizar muchas actividades simultáneas en un programa. Los hilos -a veces llamados, procesos ligeros, o hilos de ejecución- son ejecuciones concurrentes de un programa donde se comparten diversos recursos entre los distintos threads. Al estar estos hilos construidos en el mismo lenguaje, son más fáciles de usar que sus homólogos en C o C++.
- El beneficio de ser multihilo consiste en un mejor rendimiento interactivo y una mayor facilidad para modelar procesos concurrentes (como un servidor en Internet que debe atender a múltiples clientes de forma simultánea).

Threads: Introducción (I)

- Normalmente los programas están formados por un único hilo de ejecución y su comportamiento es secuencial, tienen:
  - Un punto de inicio
  - Una secuencia de ejecución
  - Un final
- Cuando en un programa se quiere realizar más de una tarea simultáneamente se utilizan técnicas de ejecución concurrente, que permiten que diversos puntos del programa se estén ejecutando a la vez.
- Existen dos aproximaciones para ejecutar tareas de forma concurrente en un programa:
  - Procesos.
  - Threads.

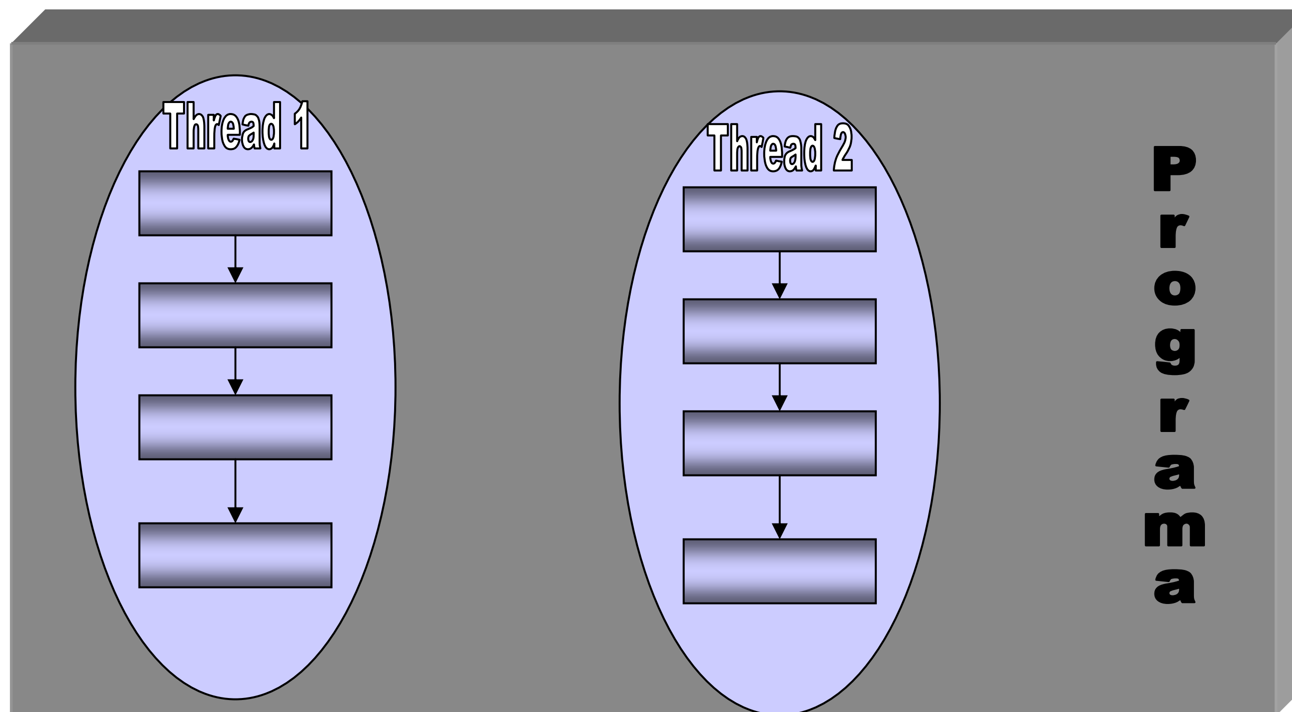
Threads: Introducción (II)

- Threads vs Procesos
  - Ambos son flujos secuenciales de control dentro de un programa.
  - Todos los threads dentro de un proceso comparten recursos como la memoria. (Un objeto creado en un thread es visible desde otro).
  - Los procesos no comparten recursos entre sí. (Un objeto creado en un proceso no es visible desde otro proceso distinto).
- Al compartir recursos, los threads se crean más rápido que los procesos y los cambios de contexto entre threads son más baratos. Esta ventaja de los threads varía mucho según el SO, siendo más grande en Windows (donde la creación de procesos es muy costosa).
- Como contrapartida, el mayor aislamiento que proporcionan los procesos hace más fácil evitar problemas que surgen de la ejecución concurrente sobre los mismos recursos.
- Ambas aproximaciones se pueden combinar en un mismo programa, que puede tener varios procesos, y cada uno de ellos con varios threads simultáneos.

UNIVERSIDADE DA CORUÑA

Threads: Introducción (III)

- Un programa concurrente ejecuta múltiples threads simultáneamente, cada uno realizando diferentes tareas.



UNIVERSIDADE DA CORUÑA

Threads: Introducción (IV)

- Java es multithread
  - Permite la realización de programas de flujo múltiple:
    - Programas que ejecutan simultáneamente múltiples tareas.
    - Por ejemplo:
      - Navegador que descarga páginas Web: un thread para la página HTML, otro para cada imagen, etc.
  - Las aplicaciones Java básicas están formadas normalmente por un único proceso con un único thread, que invoca al método "main" de la clase que se ejecuta.

UNIVERSIDADE DA CORUÑA

Threads: Introducción (V)

- Razones para utilizar multithreading …
  - Para lograr concurrencia en un programa (hacer varias cosas a la vez, cuando unas tareas no dependen de otras).
  - Para eliminar los tiempos de espera a los usuarios de un programa
    - Permiten hacer otro trabajo mientras el programa está bloqueado haciendo algo.
  - Servir a varios clientes en paralelo, sin que tengan que esperar a que terminen otros para ser atendidos.
  - Para aprovechar las capacidades multiproceso de las CPUs actuales (Multithreading, Multicore) y conseguir una mejora de la eficiencia.
- Ejemplos:
  - Servidores que aceptan conexiones de múltiples clientes (Servidores Web, FTP, correo, …)
  - Almacenar un archivo en disco.
  - Mostrar una barra de progreso de una operación de una aplicación.

UNIVERSIDADE DA CORUÑA

Threads: Java (I)

- Java para la creación de threads dispone de dos mecanismos:
  - La clase java.lang.Thread
    - Se utiliza cuando se puede heredar directamente.
  - La interfaz java.lang.Runnable
    - Se utiliza en los casos de que sea necesario la herencia múltiple
      - (por ejemplo cuando una clase ya hereda de otra, como un Applet).

- NOTA:
  - Método estático Thread.sleep(int milisegs): duerme un proceso o un thread durante el tiempo especificado.

UNIVERSIDADE DA CORUÑA

Threads: Java (II)

- **Método1**: Para crear una clase que sea un thread (un hilo de ejecución independiente):
  - Heredar de la clase Thread.
  - Definir un constructor.
  - Sobrescribir el método run():
    - public void run()

```
class HijoThread extends Thread {
    HijoThread() {
        // Inicialización
    }

    public void run() {
        // Tarea a ejecutar en el thread
        . . .
    }
}
```

- El programa principal:
  - Crea una instancia de la nueva clase:
    - HijoThread t = new HijoThread();
  - Inicia la ejecución:
    - t.start();

UNIVERSIDADE DA CORUÑA

Threads: Java (III)

- **Método 2**: En caso de utilizar herencia múltiple:
  - Declarar una clase que implemente la interfaz Runnable.
  - Define un constructor.
  - Sobrescribir el métod run().
    - public void run()

```
class HijoRunnable implements Runnable{
    HijoRunnable() {
        // Inicialización
    }

    public void run() {
        // Tarea a ejecutar en el thread
        . . .
    }
}
```

- El programa principal:
  - Crea una instancia de la nueva clase:
    - HijoRunnable runnable = new HijoRunnable();
  - Crea un thread pasándole como parámetro la nueva clase:
    - Thread t = new Thread(runnable);
  - Inicia la ejecución del thread:
    - t.start();

UNIVERSIDADE DA CORUÑA

Más información

- http://java.sun.com/docs/books/tutorial/
  - http://java.sun.com/docs/books/tutorial/reallybigindex.html
- Programación con Sockets
  - http://java.sun.com/docs/books/tutorial/networking/sockets/index.html (Sockets)
  - http://java.sun.com/docs/books/tutorial/networking/datagrams/index.html (Datagramas)
- Flujos de datos
  - http://java.sun.com/docs/books/tutorial/essential/io/index.html
  - http://java.sun.com/docs/books/tutorial/essential/io/streams.html (Streams)
  - http://java.sun.com/docs/books/tutorial/essential/io/fileio.html (Files)
- Programación concurrente
  - http://java.sun.com/docs/books/tutorial/essential/concurrency/threads.html (Threads)