

# **Programación Interactiva**

## **Aplicaciones Cliente-Servidor**

Escuela de Ingeniería de Sistemas y Computación  
Facultad de Ingeniería  
Universidad del Valle



# Motivación

- ✦ El entorno Java es altamente considerado en parte por su capacidad para escribir programas que utilizan e interactúan con los recursos de Internet y la World Wide Web.
- ✦ De hecho, los navegadores que soportan Java utilizan esta capacidad del entorno Java hasta el extremo de transportar y ejecutar applets a través de la red.

# Capas

- ⊕ Los ordenadores que se ejecutan en Internet comunican unos con otros utilizando los protocolos TCP y UDP, que son protocolos de 4 capas.
  - ⊠ Aplicación
  - ⊠ Red
  - ⊠ Transporte
  - ⊠ Enlace

# Capas

Application <i>(HTTP, ftp, telnet, ...)</i>
Transport <i>(TCP/IP, UDP, ...)</i>
Network <i>(IP, ...)</i>
Link <i>(device driver, ...)</i>

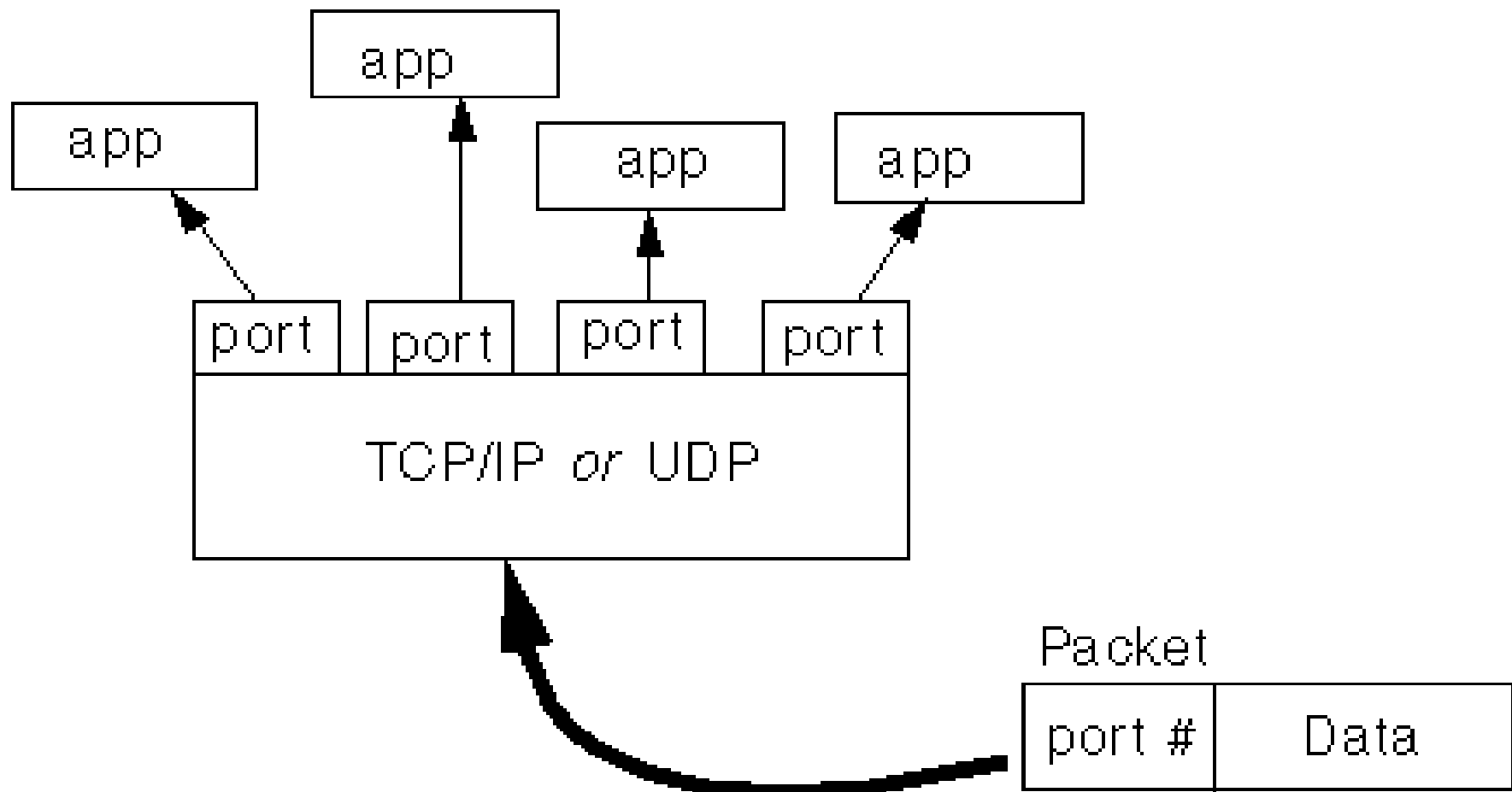
# TCP

- ✚ TCP (Transfer Control Protocol)  
es un protocolo basado en conexión  
que proporciona un flujo fiable de  
datos entre dos ordenadores.

# UDP

- ✚ UDP (User Datagram Protocol)  
es un protocolo que envía paquetes de datos independientes, llamados **datagramas** desde un ordenador a otro sin garantías sobre su llegada. UDP no está basado en la conexión como TCP.

# UDP - TCP



# Puertos

- ✿ Los protocolos TCP y UDP utilizan puertos para dirigir los datos de entrada a los procesos particulares que se están ejecutando en un ordenador.



# Puertos

- ⊕ Los números de puertos tienen un rango de 0 a 65535 (porque los puertos están representados por un número de 16 bits).
- ⊕ Los puertos entre los números 0 - 1023 están restringidos -- están reservados para servicios bien conocidos como HTTP, FTP y otros servicios del sistema.
- ⊕ Los puertos que están reservados para los servicios bien conocidos como HTTP y FTP se llaman **puertos bien conocidos**.

# URL

- ✚ URL es un acrónimo que viene de **Uniform Resource Locator** y es una referencia (una dirección) a un recurso de Internet.

```
URL google = new URL("http://www.google.com/");
```

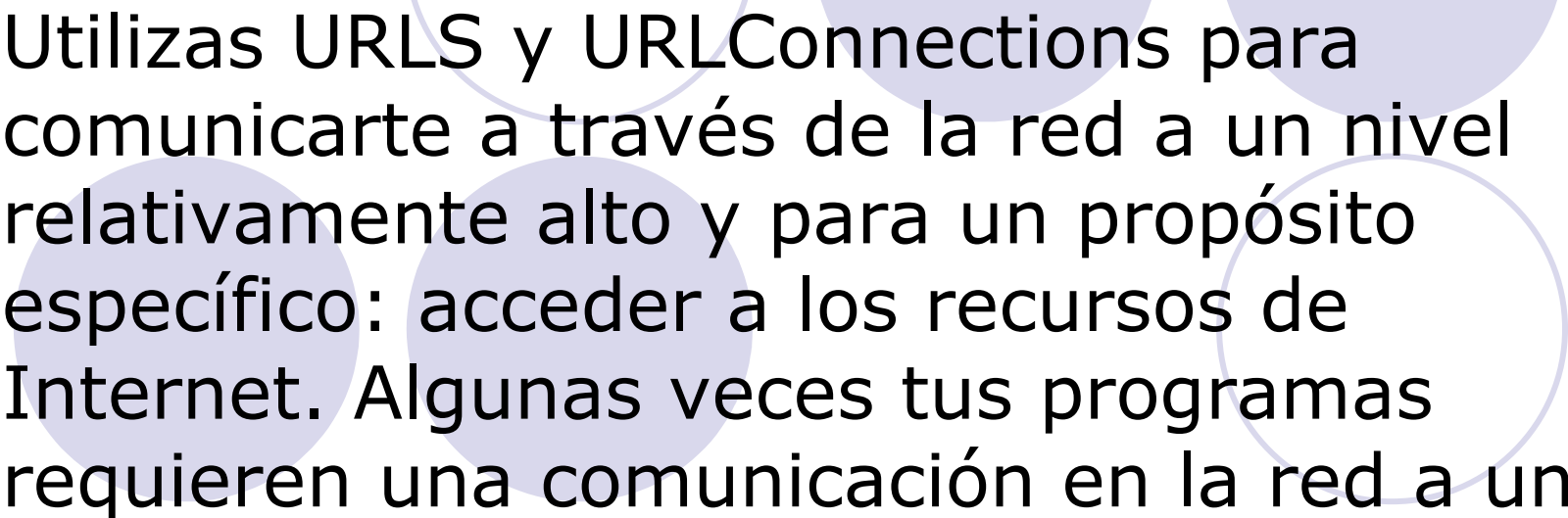
```
URL googleNetwork = new URL(google,  
    "google.network.html");
```

```
URL google = new URL("http", "www. google .com",  
    "/google.network.html");
```

```
URL google = new URL("http", "www. google .com", 80,  
    "/google.network.html");
```

# Método openConnection

```
try { URL yahoo = new
    URL("http://www.yahoo.com/");
    yahoo.openConnection();
} catch (MalformedURLException e) {
    // nueva URL() fallada . . .
} catch (IOException e) {
    // openConnection() fallada . . .
}
//ver connectionTest.java
```



Utilizas URLs y URLConnections para comunicarte a través de la red a un nivel relativamente alto y para un propósito específico: acceder a los recursos de Internet. Algunas veces tus programas requieren una comunicación en la red a un nivel más bajo, por ejemplo, cuando quieras escribir una aplicación cliente-servidor.

# Sockets

- ✚ Los *sockets* (zócalos, referido a los enchufes de conexión de cables) son mecanismos de comunicación entre programas a través de una red TCP/IP.
- ✚ De hecho, al establecer una conexión vía Internet estamos utilizando sockets: los sockets realizan la interfase entre la aplicación y el protocolo TCP/IP.

# Sockets

- ✿ Un socket es un punto final en un enlace de comunicación de dos vías entre dos programas que se ejecutan en la red.
- ✿ Las clases Socket son utilizadas para representar conexiones entre un programa cliente y otro programa servidor.

# Sockets

- ✚ El paquete `java.net` proporciona dos clases -- `Socket` y `ServerSocket` -- que implementan los lados del cliente y del servidor de una conexión, respectivamente.

# Sockets

- ✚ Los sockets tienen asociado un *port* (puerto). En general, las conexiones via internet pueden establecer un puerto particular (por ejemplo, en `http://www.rockar.com.ar:80/index.html` el puerto es el 80).
- ✚ Esto casi nunca se especifica porque ya hay definidos puertos por defecto para distintos protocolos: 20 para ftp-data, 21 para ftp, 79 para finger, etc.



# Sockets

- ✿ Para establecer una conexión a través de un socket, tenemos que programar por un lado el servidor y por otro los clientes.
- ✿ En el servidor, creamos un objeto de la clase **ServerSocket** y luego esperamos algún cliente (de clase **Socket**) mediante el método **accept()**:

# Sockets

- ✚ `ServerSocket conexion = new ServerSocket(5000);`  
//5000 es el puerto en este caso
- ✚ `Socket cliente = conexion.accept();`  
// espero al cliente
- ✚ Desde el punto de vista del cliente, necesitamos un Socket al que le indiquemos la dirección del servidor y el número de puerto a usar:
- ✚ `Socket conexion = new Socket ( direccion, 5000 );`

# Sockets

- ✚ Una vez establecida la conexión, podemos intercambiar datos usando streams.
- ✚ Como la clase `URLConnection`, la clase `Socket` dispone de métodos `getInputStream` y `getOutputStream` que nos dan respectivamente un `InputStream` y un `OutputStream` a través de los cuales transferir los datos.

# URL - Sockets

- ✦ Los clientes y servidores que se comunican mediante un canal fiable (como una URL o un socket) tienen un canal punto a punto dedicado entre ellos (o al menos la ilusión de uno).
- ✦ Para comunicarse, establecen una conexión, transmiten los datos y luego cierran la conexión.

# URL - Sockets

- ✦ Todos los datos enviados a través del canal se reciben en el mismo orden en el que fueron enviados.
- ✦ Esto está garantizado por el canal.

# Datagramas

- ✦ Las aplicaciones que se comunican mediante datagramas envían y reciben paquetes de información completamente independientes.
- ✦ Estos clientes y servidores no tienen y no necesitan un canal punto a punto dedicado.
- ✦ El envío de los datos a su destino no está garantizado, ni su orden de llegada.

# Datagramas

- ✦ En general, un datagrama es un mensaje autocontenido independiente enviado a través de la red, cuya llegada, momento de llegada y contenido no está garantizado.

# Datagramas

- ✦ El paquete `java.net` contiene dos clases para ayudarte a escribir programas Java que utilicen datagramas para enviar y recibir paquetes a través de la red: `DatagramSocket` y `DatagramPacket`. Su aplicación envía y recibe `DatagramPackets` a través de un `DatagramSocket`.

`java.net.DatagramPacket`

`java.net.DatagramSocket`