

PCD2018 - 19

PRIMITIVE DI NETWORKING

Le astrazioni di base che abbiamo a disposizione per la comunicazione fra più JVM corrispondono direttamente alle caratteristiche del protocollo TCP/IP:

- Sockets
- Datagrams

SOCKETS

Un Socket è una astrazione per la comunicazione bidirezionale punto-punto fra due sistemi.

```
package java.net;
/**
 * This class implements client sockets (also
 * called just "sockets"). A socket is an
 * endpoint for communication between two machines.
 */
public class Socket implements Closeable;
```

```
/**
 * Creates a socket and connects it to the specified
 * remote address on the specified remote port.
 * The Socket will also bind() to the local address
 * and port supplied.
 *
 * @param address the remote address
 * @param port the remote port
 * @param localAddr the local address or null for anyLocal
 * @param localPort the local port, 0 zero for arbitrary
 */
public Socket(InetAddress address, int port,
              InetAddress localAddr, int localPort)
    throws IOException
```

Un *client* Socket è un socket per iniziare il collegamento verso un'altra macchina.

Un *server* Socket è un socket per attendere che un'altra macchina ci chiami.


```
package java.net;  
/**  
 * A server socket waits for requests to come  
 * in over the network.  
 **/  
public class ServerSocket implements Closeable;
```

```
/**
 * Create a server with the specified port, listen backlog,
 * and local IP address to bind to.
 *
 * @param port the local port, 0 zero for arbitrary
 * @param backlog maximum length of the queue of incoming
 *       connections
 * @param bindAddr the local InetAddress the server
 *       will bind to
 */
public ServerSocket(int port, int backlog,
    InetAddress bindAddr)
    throws IOException
```

Un `Socket` rappresenta un collegamento attivo.

Lato client, lo diventa appena il collegamento (l'handshake TCP/IP) è completato.

Lato server, viene ritornato quando un collegamento viene ricevuto e completato.

```
/**  
 * Listens for a connection to be made to this socket and  
 * accepts it. The method blocks until a connection is made.  
 *  
 **/  
public Socket accept() throws IOException
```

Un Socket (sia *client* che *server* collegato) ci fornisce un `InputStream` ed un `OutputStream` per ricevere e trasmettere dati nel collegamento.

```
/**  
 * Returns an input stream for this socket.  
 *  
 **/  
public InputStream getInputStream()  
    throws IOException
```

```
/**  
 * Returns an output stream for this socket.  
 *  
 **/  
public OutputStream getOutputStream()  
    throws IOException
```

Questi stream sono sottoposti a diverse regole:

- sono thread-safe, ma un solo thread può scrivere o leggere per volta, pena eccezioni
- i buffer sono limitati, ed in alcuni casi i dati in eccesso possono essere silenziosamente scartati

- lettura e scrittura possono bloccare il thread
- alcune connessioni possono avere caratteristiche particolari (per es. urgent data)

Una volta terminato l'uso, il Socket va chiuso esplicitamente.

```
/**  
 * Closes this socket. Any thread currently blocked in  
 * accept() will throw a SocketException.  
 **/  
public void close() throws IOException
```

Avendo come astrazione della comunicazione gli Stream, la comunicazione via socket ha il difetto di richiedere la definizione esplicita di un confine fra richieste e risposte.

Dallo Stream non possiamo sapere se la richiesta è terminata, e non possiamo segnalare di aver inviato tutta la risposta.

pcd2018.sockets.HelloServer

```
try (  
    ServerSocket serverSocket = new ServerSocket(portNumber);  
    Socket socket = serverSocket.accept();  
    PrintWriter out = new PrintWriter(  
        socket.getOutputStream(), true);  
    BufferedReader in = new BufferedReader(  
        new InputStreamReader(socket.getInputStream()));  
)
```

```
{
String inputLine;
System.out.println("Received data.");
while ((inputLine = in.readLine()) != null) {
    System.out.println("Received: " + inputLine);
    out.println("Hello " + inputLine);
}
System.out.println("Server closing.");
}
```

Il protocollo fra client e server è "linea di testo terminata da \n".

Appena il server riceve il carattere di a capo (e non prima), `BufferedReader::readline` ritorna ed il server risponde.

pcd2018.sockets.HelloClient

```
try (  
    Socket socket = new Socket("127.0.0.1", portNumber);  
    PrintWriter out = new PrintWriter(  
        socket.getOutputStream(), true);  
    BufferedReader in = new BufferedReader(  
        new InputStreamReader(socket.getInputStream()));  
    ) {  
        System.out.println("Connected, sending " + args[0]);  
        out.println(args[0]);  
        System.out.println("Got back: " + in.readLine());  
    }
```


DATAGRAMS

Un Datagram è un'astrazione per l'invio di un pacchetto di informazioni singolo verso una destinazione o verso più destinazioni.

Il concetto di connessione è diverso rispetto al socket, e non c'è garanzia di ricezione o ordinamento in arrivo.

```
package java.net;  
/**  
 * Datagram packets are used to implement a connectionless  
 * packet delivery service.  
 **/  
public final class DatagramPacket
```

```
/**  
 * Constructs a DatagramPacket for receiving packets.  
 *  
 * @param buf buffer for holding the incoming datagram  
 * @param length the number of bytes to read.  
 **/  
public DatagramPacket(byte[] buf, int length)
```

```
/**
 * Constructs a DatagramPacket for receiving packets.
 *
 * @param buf the packet data
 * @param length the packet length
 * @param address the destination address
 * @param the destination port number
 */
public DatagramPacket(byte[] buf, int length,
    InetAddress address, int port)
```

Un DatagramPacket è pur sempre un pacchetto UDP, quindi soggiace alle stesse limitazioni:
In particolare, la dimensione massima è di 64Kb.

Protocollo	MTU (bytes)
IPv4 (link)	68
IPv4 (host)	576
IPv4 (Ethernet)	1500
IPv6	1280
802.11	2304

Per inviare o ricevere abbiamo una sola classe, senza distinzione di operatività.


```
package java.net;
/**
 * Constructs a datagram socket and binds it to the
 * specified port on the local host machine.
 *
 * @param port the port to use
 */
public DatagramSocket(int port) throws SocketException
```

Possiamo "connettere" una `DatagramSocket` già creata ad un indirizzo, ma il significato è diverso.

```
/**
 * Connects the socket to a remote address for this
 * socket. When a socket is connected to a remote address,
 * packets may only be sent to or received from that
 * address. By default a datagram socket is not connected.
 *
 * @param address the remote address for the socket
 * @param port the remote port for the socket.
 */
public void connect(InetAddress address, int port)
```

```
/**
 * Sends a datagram packet from this socket. The
 * DatagramPacket includes information indicating the data
 * to be sent, its length, the IP address of the remote host,
 * and the port number on the remote host.
 *
 * @param p the DatagramPacket to be sent
 */
public void send(DatagramPacket p) throws IOException
```

```
/**
 * Receives a datagram packet from this socket. When this
 * method returns, the DatagramPacket's buffer is filled
 * with the data received. The datagram packet also
 * contains the sender's IP address, and the port number
 * on the sender's machine.
 *
 * @param p the DatagramPacket to be sent
 */
public void receive(DatagramPacket p) throws IOException
```

```
/**
 * Closes this datagram socket.
 *
 * Any thread currently blocked in receive() upon this
 * socket will throw a SocketException.
 *
 * @param p the DatagramPacket to be sent
 */
public void close()
```

Con i Datagram la logica del protocollo è differente.

Abbiamo a disposizione:

- la dimensione del messaggio nota (e quindi l'informazione di ricezione completa)
- la possibilità di inviare messaggi a più indirizzi contemporaneamente (multicast)

Ma rispetto ai Socket, perdiamo:

- l'affidabilità: non c'è né garanzia né segnale di consegna
- la reciprocità: c'è una sola direzione; la risposta richiede mettersi in ascolto
- la dimensione: messaggi grossi subiscono una forte penalità di affidabilità

pcd2018.sockets.EchoServer

```
public void run() {  
    byte[] buf = new byte[256];  
    DatagramPacket packet = new DatagramPacket(buf, buf.length);  
    System.out.println("Server setup. Receiving...");  
    try {  
        socket.receive(packet);  
        String received = new String(  
            packet.getData(), 0, packet.getLength());  
        System.out.println("Received: " + received);  
    } catch (IOException e) { e.printStackTrace(); }  
    finally { socket.close(); }  
}
```

pcd2018.sockets.EchoClient

```
DatagramSocket socket = new DatagramSocket();  
byte[] buf = args[0].getBytes();  
InetAddress address = InetAddress.getByName("localhost");  
DatagramPacket packet = new DatagramPacket(  
    buf, buf.length, address, PORT);  
socket.send(packet);  
socket.close();
```

URL

Già dalla prima versione Java include nativamente una classe per interagire con risorse web.

```
package java.net;  
/**  
 * Class URL represents a Uniform Resource Locator, a  
 * pointer to a "resource" on the World Wide Web.  
 **/  
public final class URL
```

Una URL ha un formato complesso ed in grado di esprimere molte cose (protocolli, porte richieste, indirizzi remoti, file, jar, ecc.)

```
/**  
 * Creates a URL object from the String representation.  
 **/  
public URL(String spec) throws MalformedURLException
```

Possiamo ottenere da una URL direttamente lo stream
ottenuto dalla richiesta GET


```
/**  
 * Opens a connection to this URL and returns an  
 * InputStream for reading from that connection.  
 **/  
public InputStream openStream() throws IOException
```

pcd2018.sockets.UrlGet

```
BufferedReader reader = new BufferedReader(  
    new InputStreamReader(  
        new URL("https://httpbin.org/get").openStream()));  
String line;  
while ((line = reader.readLine()) != null) {  
    System.out.println(line);  
}
```

Se vogliamo invece effettuare una richiesta POST dobbiamo esplicitamente richiederlo ottenendo la `connection` dalla URL e segnalando che desideriamo usarla a scopi di output.

```
/**  
 * Returns a URLConnection instance that represents a  
 * connection to the remote object referred to by the URL.  
 **/  
public URLConnection openConnection() throws IOException
```

```
package java.net;
/**
 * The abstract class URLConnection is the superclass of
 * all classes that represent a communications link
 * between the application and a URL.
 */
public abstract class URLConnection;
```

```
/**  
 * Sets the value of the doOutput field for this  
 * URLConnection to the specified value.  
 **/  
public void setDoOutput(boolean dooutput)
```

```
/**  
 * Returns an output stream that writes to this connection.  
 **/  
public OutputStream getOutputStream() throws IOException
```

pcd2018.sockets.UrlPost

```
URL url = new URL(https://httpbin.org/post);
URLConnection connection = url.openConnection();
connection.setDoOutput(true);

PrintWriter writer = new PrintWriter(
    connection.getOutputStream());
writer.println("test=val");
writer.close();
```



```
BufferedReader reader = new BufferedReader(  
    new InputStreamReader(connection.getInputStream()));  
String line;  
while ((line = reader.readLine()) != null) {  
    System.out.println(line);  
}
```

ESEMPIO COMPLETO

Tema: realizzare un server che gestisce il gioco di TicTacToe fra due giocatori.

Realizzare quindi un client che gioca scegliendo una casella libera a caso.

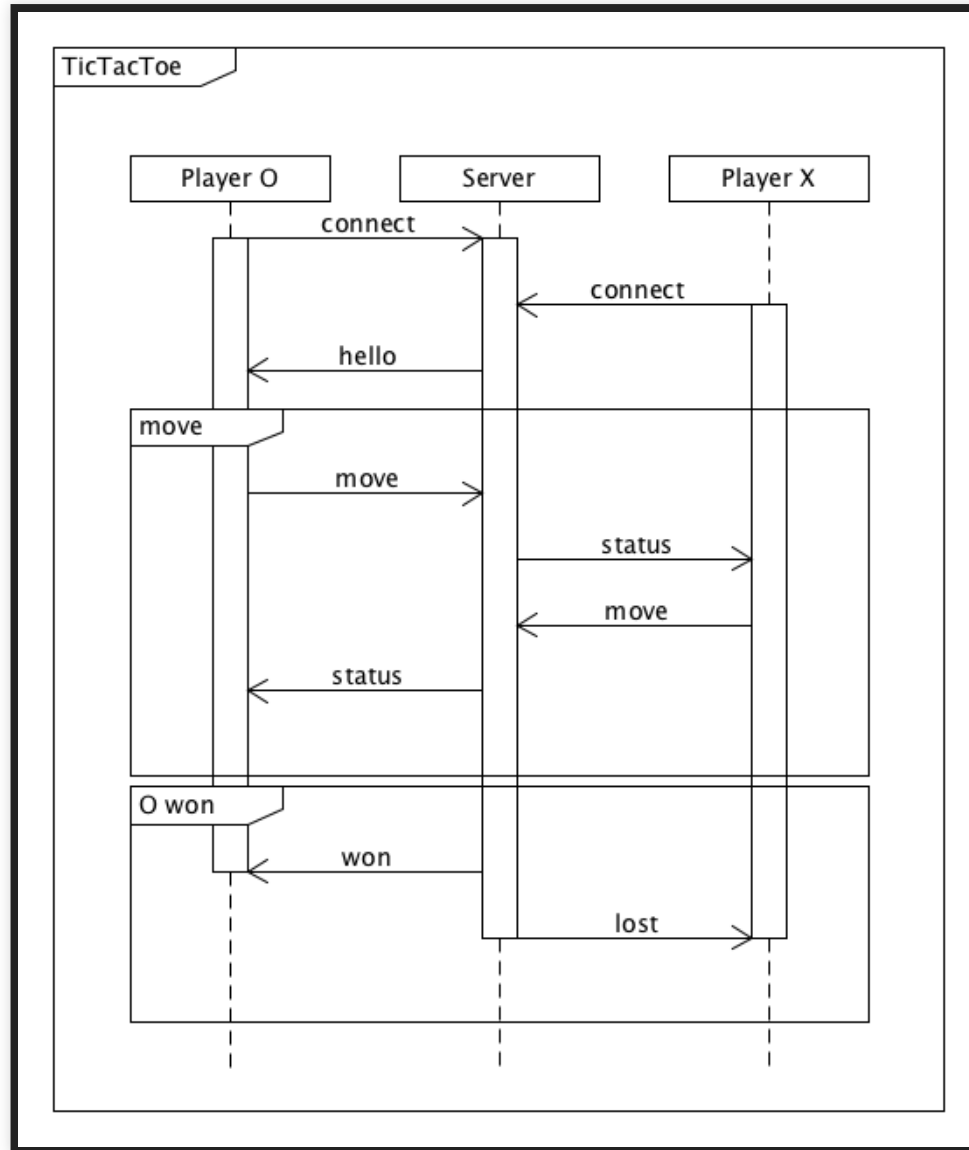
Il server deve:

- rispondere alla prima connessione salutando il primo giocatore
- rispondere alla seconda connessione salutando il secondo giocatore

- richiedere la mossa da ciascun giocatore al suo turno mostrandogli lo stato del gioco
- individuare la conclusione della partita e chiudere le connessioni

Il client deve:

- collegarsi al server
- interpretare la risposta con lo stato della partita
- effettuare una mossa a caso fra quelle legali



pcd2018.sockets.ToeClient

```
try (  
    Socket socket = new Socket("127.0.0.1",  
        ToeServer.PORT_GAME);  
    PrintWriter out = new PrintWriter(  
        socket.getOutputStream(), true);  
    BufferedReader in = new BufferedReader(  
        new InputStreamReader(socket.getInputStream()));) {  
    System.out.println("Connected.");  
    String line;  
    boolean done = false;  
    while (!done && (line = in.readLine()) != null) {
```



```
if (line.startsWith("PLAYER")) {  
    // gestiamo una mossa  
    line = in.readLine(); // prima riga  
    line = in.readLine(); // seconda riga  
    line = in.readLine(); // terza riga  
    line = in.readLine(); // mosse disponibili  
    String[] split = line.split("\\s");  
    String move = split[rnd.nextInt(split.length)];  
    out.println(move);  
    out.flush();  
}
```

```
} else if (line.startsWith("Hello")) {  
    // partita iniziata  
    System.out.println(line);  
}  
else {  
    // partita finita  
    done = true;  
    System.out.println(line);  
}
```

pcd2018.sockets.ToeServer

```
class GameServer implements Runnable {  
  
    int port;  
    Socket[] sockets = new Socket[2];  
    PrintWriter[] outs = new PrintWriter[2];  
    BufferedReader[] ins = new BufferedReader[2];  
    Game game = new Game();  
}
```

```
try (ServerSocket serverSocket = new ServerSocket(port);) {  
    // attendi che i giocatori si colleghino  
    connectPlayers(serverSocket);  
    // dai al primo giocatore la situazione iniziale  
    GameResult status = game.status();  
    outs[0].println(status);  
    outs[0].flush();  
}
```

```
// finché la partita non è conclusa...
while (!status.end) {
    // attendi la mossa dal giocatore
    String move = ins[status.next].readLine();
    // eseguila
    status = game.move(status.next, Integer.parseInt(move));
    if (!status.end) {
        // informa l'altro giocatore
        outs[status.next].println(status);
        outs[status.next].flush();
    }
}
```

```
// comunica il risultato
System.out.println(status);
if (status.valid) {
    outs[status.next].println("You won.");
    outs[(status.next + 1) & 0x1].println("You lost.");
    System.out.println("Player " + (
        status.next == 0 ? "O" : "X") + " won.");
} else {
    outs[0].println("Tied.");
    outs[1].println("Tied.");
    System.out.println("The game is a tie.");
}
```

```
// chiudi le risorse  
outs[0].close();  
outs[1].close();  
ins[0].close();  
ins[1].close();  
sockets[0].close();  
sockets[1].close();
```