Reti e Laboratorio
Modulo Laboratorio III
AA. 2026-2027

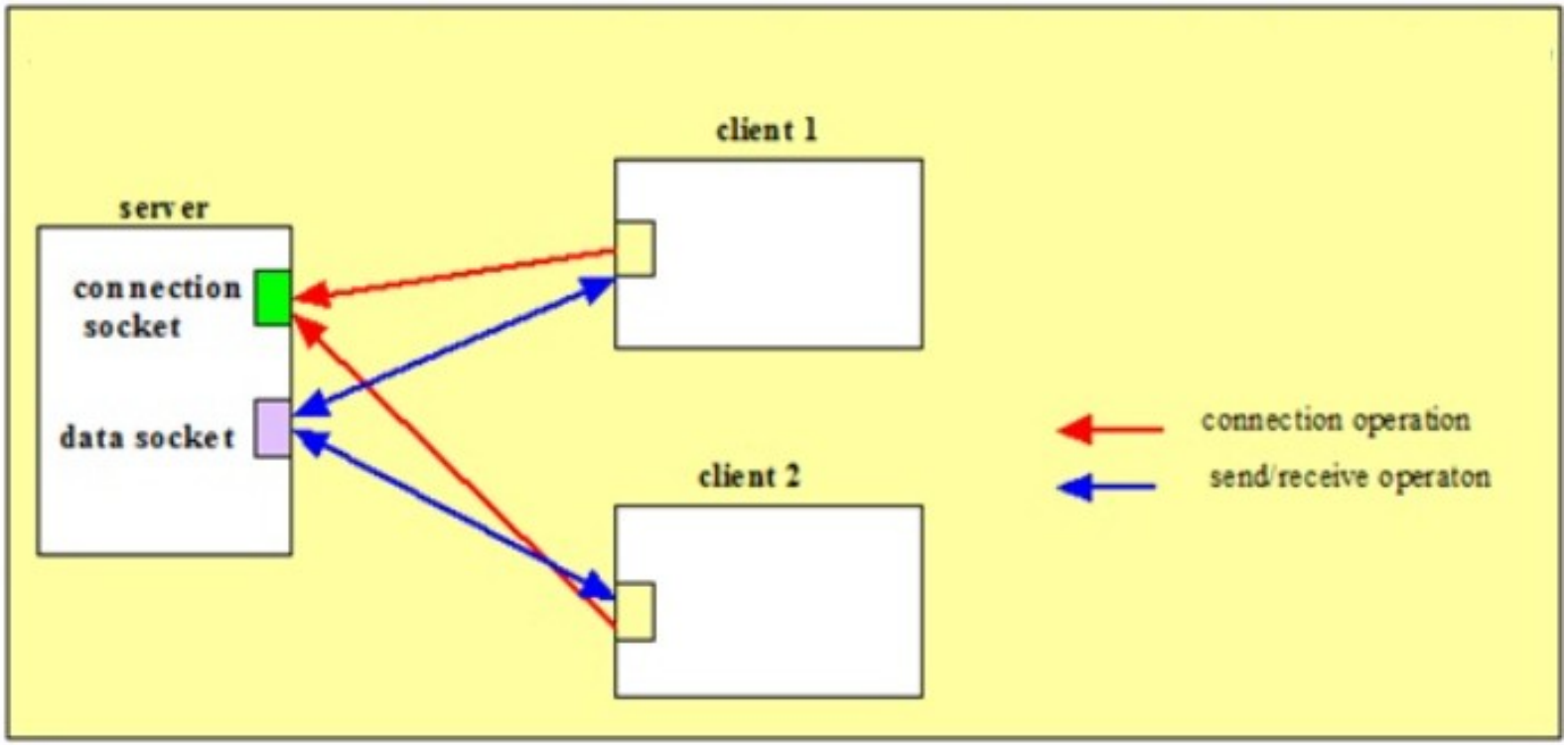
docente: Laura Ricci

laura.ricci@unipi.it

Lezione7
Sockets for servers
UDP Sockets

29/10/2026

SOCKET LATO SERVER



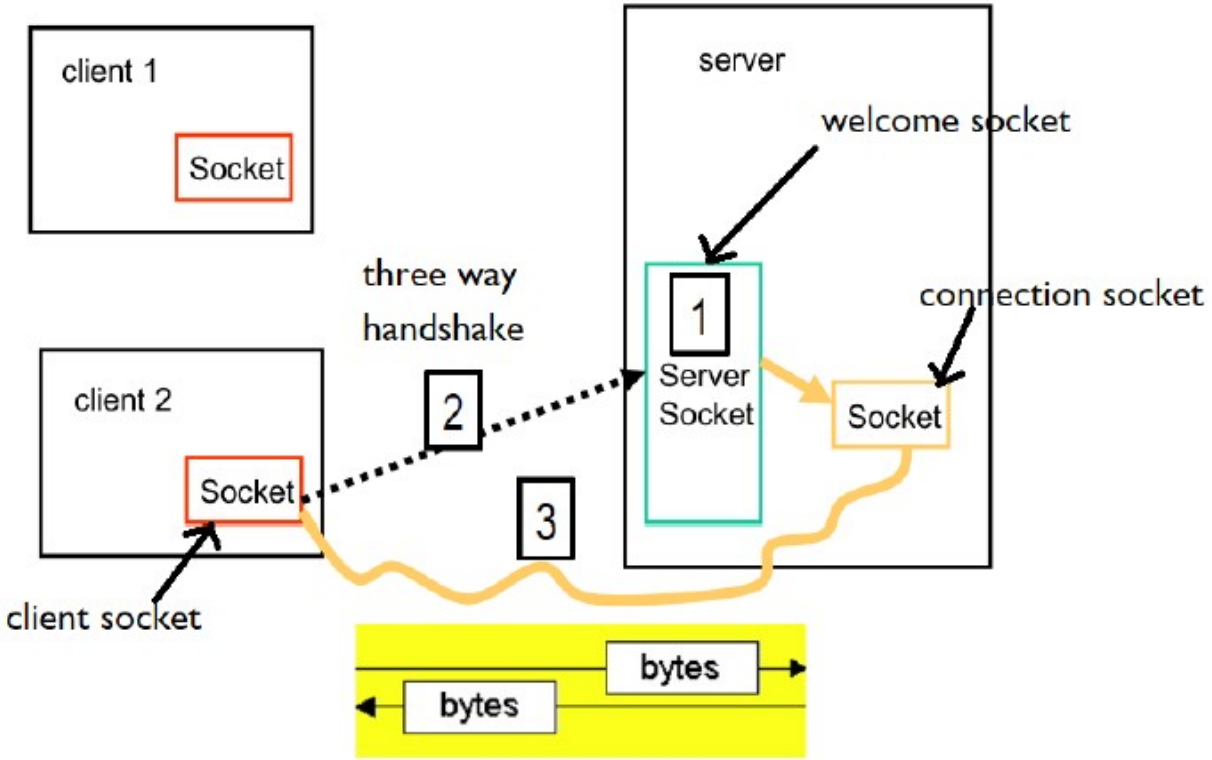
due tipi di socket lato server:

- connection (welcome)socket
- data socket

SOCKET LATO SERVER

- lato server, esistono due tipi di socket TCP:
  - welcome (passive, listening) sockets: utilizzati dal server per accettare le richieste di connessione
  - connection (active) sockets: connettono il server ad un particolare client e supportano lo streaming di byte con essi
- il client crea un active socket per richiedere la connessione
- il server accetta una richiesta di connessione sul welcome socket
  - crea un proprio connection socket che rappresenta il punto terminale della sua connessione con il client
  - la comunicazione vera e propria avviene mediante la coppia di active socket presenti nel client e nel server

SOCKET LATO SERVER

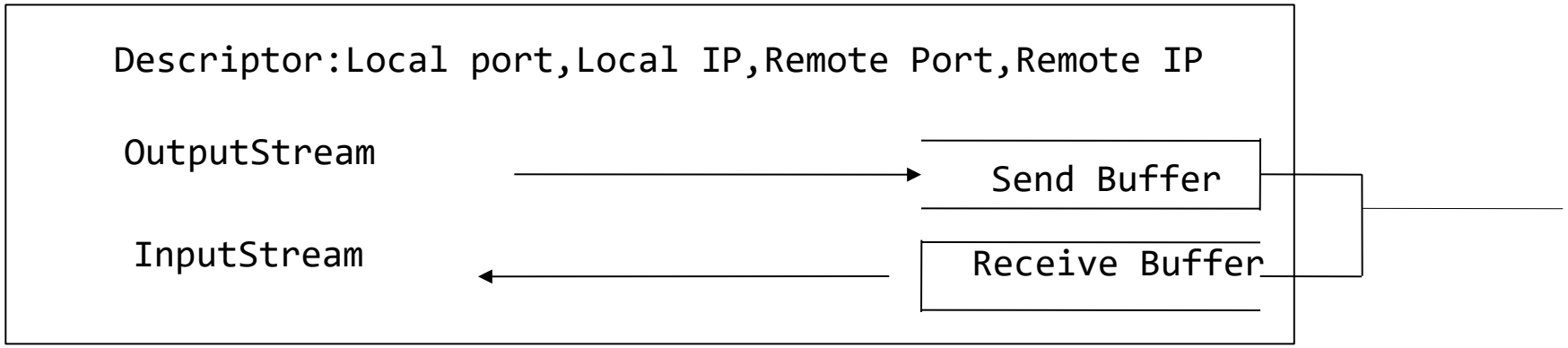


SOCKET LATO SERVER

- il server pubblica un proprio servizio
  - gli associa un welcome socket, sulla porta remota `PS`, all'indirizzo `IPS`
  - usa un oggetto di tipo `ServerSocket`
- il client crea un `Socket` e lo connette all'endpoint `IPS + PS`
- la creazione del socket effettuata dal client produce in modo atomico la richiesta di connessione al server
  - three way handshake completamente gestito dal supporto
  - se la richiesta viene accettata
    - il server crea un socket dedicato per l'interazione con quel client
    - tutti i messaggi spediti dal client vengono diretti automaticamente sul nuovo socket creato.

STREAM BASED COMMUNICATION

- dopo che la richiesta di connessione viene accettata, client e server associano streams di bytes di input/output ai socket dedicati a quella connnessione, poichè gli stream sono unidirezionali
  - a seconda del servizio può essere necessario un solo stream di output dal server verso il client, oppure una coppia di stream da/verso il client
- la comunicazione avviene mediante lettura/scrittura di dati sullo stream
- eventuale utilizzo di filtri associati agli stream



Struttura del Socket TCP

JAVA STREAM SOCKET API: LATO SERVER

`java.net.ServerSocket:` costruttori

```
public ServerSocket(int port)throws BindException, IOException
public ServerSocket(int port,int length) throws BindException,
                                                IOException
```

- costruisce un listening socket, associandolo alla porta `port`
- `length`: lunghezza della coda in cui memorizzare le richieste di connessione: se la coda è piena, ulteriori richieste di connessione sono rifiutate

```
public ServerSocket(int port,int length,Inetaddress bindAddress)....
```

- permette di collegare il socket ad uno specifico indirizzo `IP` locale
- utile per macchine dotate di più schede di rete, ad esempio un host con due indirizzi `IP`, uno visibile da Internet, l'altro visibile solo a livello di rete locale
- se voglio servire solo le richieste in arrivo dalla rete locale, associo il connection socket all'indirizzo `IP` locale

JAVA STREAM SOCKET API: LATO SERVER

- accettare una nuova connessione dal connection socket

```
public Socket accept( ) throws IOException
```

metodo della classe **ServerSocket**.

- quando il processo server invoca il metodo `accept()`, pone il server in attesa di nuove connessioni
- bloccante: se non ci sono richieste, il server si blocca (possibile utilizzo di time-outs)
- quando c'è almeno una richiesta, il processo si sblocca e costruisce un nuovo socket tramite cui avviene la comunicazione effettiva tra client e server

PORT SCANNER LATO SERVER

- ricerca dei servizi attivi sull'host locale

```
import java.net.*;
public class LocalPortScanner {
   public static void main(String args[])
     {for (int port= 1; port<= 1024; port++)
         try   {ServerSocket server = new ServerSocket(port);}
         catch (BindException ex)
            {System.out.println(port + "occupata");}
         catch (Exception ex) {System.out.println(ex);}
      } }
```

CICLO DI VITA TIPICO DI UN SERVER

```
// instantiate the ServerSocket
ServerSocket servSock = new ServerSocket(port);
while (! done) // oppure while(true) {
     // accept the incoming connection
     Socket sock = servSock.accept();
    // ServerSocket is connected ... talk via sock
     InputStream in = sock.getInputStream();
     OutputStream out = sock.getOutputStream();
     //client and server communicate via in and out and do their work
     sock.close();
               }
servSock.close();
```

MULTITHREADED SERVER

- nello schema precedente, la fase "communicate and work" può essere eseguita in modo concorrente da più threads
- un thread per ogni client, gestisce le interazioni con quel particolare client
- il server può gestire le richieste in modo più efficiente
- tuttavia...threads: anche se processi lightweigth ma utilizzano risorse !
  - esempio: un thread che utilizza `1MB` di `RAM`. `1000` thread simultanei possono causare problemi !
- Soluzioni alternative:
  - thread Pooling
  - `ServerSocketChannels` di `NIO`

IL SERVIZIO: "QUOTE OF THE DAY" IN JAVA

```
package QuoteServer;

import java.io.*;
import java.net.*;
import java.util.Random;

public class QuoteServer {
    public static void main(String[] args) {
        int port = 2017; // la porta "classica" del servizio QOTD (Quote of the Day)
        // Alcune frasi di esempio
        String[ ] quotes = {
            "Posso resistere a tutto tranne che alla tentazione - Oscar Wilde",
            "L'istruzione è ciò che resta dopo che si è dimenticato ciò che si è
                                imparato a scuola - Albert Einstein",
            "Sii il cambiamento che vuoi vedere nel mondo - Mahatma Gandhi ",
            "Il successo non è definitivo, il fallimento non è fatale: ciò che conta è
             il coraggio di continuare - Winston Churchill",
            "Scegli un lavoro che ami, e non dovrai lavorare neppure un giorno in vita
             tua - Confucio"
           };
```

porte 0-1023 privilegiate

IL SERVIZIO: "QUOTE OF THE DAY" IN JAVA

```
try (ServerSocket serverSocket = new ServerSocket(port)) {
      System.out.println("📜 Quote of the Day server attivo sulla porta " + port);
      while (true) {
        try (Socket clientSocket = serverSocket.accept()){
            System.out.println("🔗 Connessione da: "+ clientSocket.getInetAddress());
            try (PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
             { // ultimo parametro a TRUE = Autoflush
              String quote = quotes[new Random().nextInt(quotes.length)];
              out.println(quote);
              System.out.println("✨ Inviata: " + quote);
            } catch (IOException e) {
                  System.err.println("❌ Errore nella scrittura: " + e.getMessage()); }}
        catch (IOException e) {
                  System.err.println("❌ Errore con un client: " + e.getMessage()); }}
    } catch (IOException e) {
            System.err.println("❌ Errore nell'avvio del server: " + e.getMessage());
      } } }
```

si ferma qui ed aspetta, quando un client si connette restituisce un nuovo Socket

servizio della richiesta

"QUOTE OF THE DAY" MULTITHREADED

```
package QuoteServerMT;
import java.io.*;
import java.net.*;
import java.util.Random;

public class QuoteServerMT{
    public static void main(String[] args) {
        int port = 1717; // porta scelta >1024
        String[ ] quotes = {
            "Posso resistere a tutto tranne che alla tentazione - Oscar Wilde",
            "L'istruzione è ciò che resta dopo che si è dimenticato ciò che si è
                                                imparato a scuola - Albert Einstein",
            "Sii il cambiamento che vuoi vedere nel mondo - Mahatma Gandhi ",
            "Il successo non è definitivo, il fallimento non è fatale: ciò che conta
             è il coraggio di continuare - Winston Churchill",
            "Scegli un lavoro che ami, e non dovrai lavorare neppure un giorno in
             vita tua - Confucio"
           };
```

"QUOTE OF THE DAY" MULTITHREADED

```
try (ServerSocket serverSocket = new ServerSocket(port)) {
        System.out.println("📜 Quote of the Day server multithreaded attivo
                           sulla porta " + port);
         ExecutorService pool = Executors.newFixedThreadPool(20);
         while (true) {
            Socket clientSocket = serverSocket.accept();
            pool.execute(new ClientHandler(clientSocket,quotes));}
      }
catch (IOException e) {
        System.err.println("❌ Errore di IO: " +  e.getMessage());}}}
```

"QUOTE OF THE DAY" MULTITHREADED

```
// Classe interna per gestire i client
private static class ClientHandler implements Runnable {
     private Socket clientSocket;
     String[] quotes;
     public ClientHandler(Socket socket, String[] quotes) {
            this.clientSocket = socket;
            this.quotes = quotes;
             }
```

"QUOTE OF THE DAY" MULTITHREADED

```
public void run() {
     try (PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true))
         { String quote = quotes[new Random().nextInt(quotes.length)];
            out.println(quote);
            System.out.println("✨ Inviata a "+clientSocket.getInetAddress() + ": "+quote);
         } catch (IOException e) {
                System.err.println("❌ Errore con il client: " + e.getMessage());
                } finally {
                    try {
                         clientSocket.close();
                     } catch (IOException e)
                {
                System.err.println("❌ Errore nella chiusura del client:" + e.getMessage());
                     }
                 }
             }
         }
     }
```

TCP ED UDP: CONFRONTO

- in certi casi TCP offre "più di quanto necessario"
  - non interessa garantire che tutti i messaggi vengano recapitati
  - si vuole evitare l'overhead dovuto alla ritrasmissione dei messaggi
  - non è necessario leggere i dati nell'ordine con cui sono stati spediti

- UDP supporta una comunicazione connectionless e fornisce un insieme molto limitato di servizi, rispetto a TCP
  - aggiunge un ulteriore livello di indirizzamento a quello offerto dal livello IP, quello delle porte
  - offre un servizio di scarto dei pacchetti corrotti

- uno slogan per UDP:

"Timely, rather than orderly and reliable delivery"

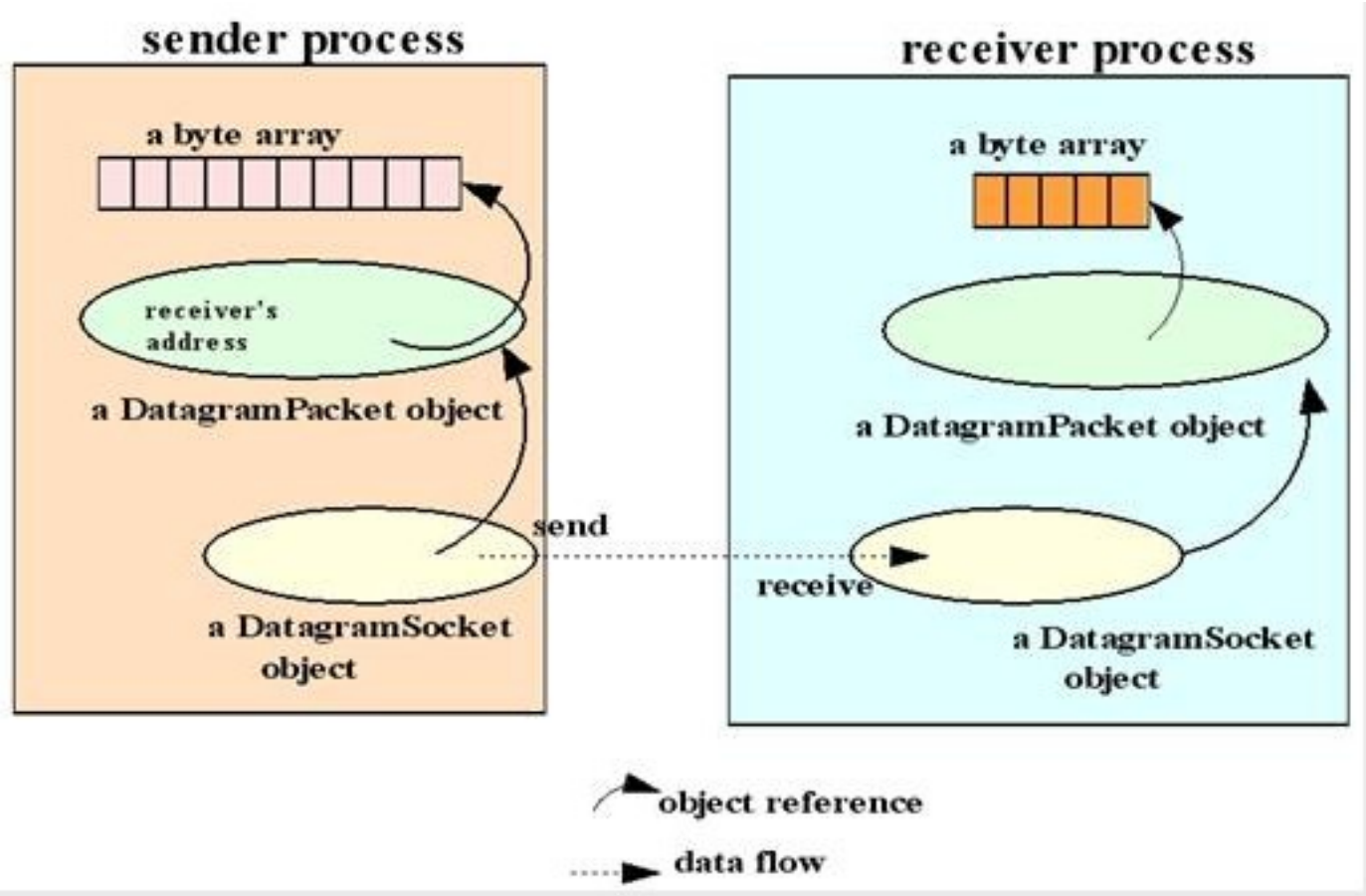
UDP: QUANDO USARLO?

- stream video/audio: meglio perdere un frame che introdurre overhead nella trasmissione di ogni frame
- tutti gli host di un ufficio inviano, ad intervalli di tempo brevi e regolari, un keep-alive ad un server centrale
  - la perdita di un keep alive non è importante
  - non è importante che il messaggio spedito alle `10:05` arrivi prima di quello spedito alle `10:07`
- compravendita di azioni, le variazioni di prezzo tracciate in uno "stock ticker"
  - la perdita di una variazione di prezzo può essere tollerata per titoli azionari di minore importanza
  - il prezzo deve essere controllato al momento della compravendita
- alcuni servizi su `UDP`: `DNS`, prime versioni di `NFS`, `TFTP` (trivial file transfer protocol), alcuni protocolli peer-to-peer

CONNECTION ORIENTED VS. CONNECTIONLESS

- JAVA socket API: interfacce diverse per UDP e TCP
- TCP: Stream Sockets
  - "apertura di una connessione", collegare il client socket al server
- UDP: Datagram Sockets
  - non è richiesto un collegamento prima di inviare una lettera
  - piuttosto è necessario specificare l'indirizzo del destinatario per ogni lettera spedita
  - lettera = pacchetto
    - ogni pacchetto, chiamato DatagramPacket, è indipendente dagli altri e contiene l'informazione per il suo instradamento

UDP IN JAVA

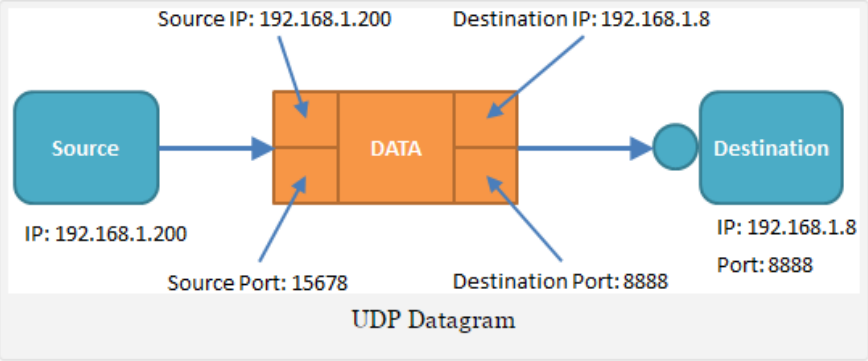

sender process
a byte array
receiver's address
a DatagramPacket object
a DatagramSocket object
send
receive
receiver process
a byte array
a DatagramPacket object
a DatagramSocket object
object reference
data flow

IL DATAGRAM UDP

- dimensione massima teorica di un pacchetto: 65536 bytes
  - IP header= 20 bytes, UDP header=8 bytes
  - molte piattaforme limitano la dimensione massima a 8192 bytes
- in JAVA un datagram UDP è rappresentato come un'istanza della classe DatagramPacket

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	



- il mittente deve inizializzare
  - il campo DATA
  - destination IP e destination port
- source IP inserito automaticamente
- source port può essere effimera

DATAGRAMPACKET: 6 COSTRUTTORI

- 2 costruttori per ricevere i dati

```
public DatagramPacket(byte[ ] buffer, int length)
public DatagramPacket(byte[ ] buffer, int offset, int length)
```

- 4 costruttori per inviare dati

```
public DatagramPacket(byte[ ] buffer, int length,
       InetAddress remoteAddr, int remotePort)
public DatagramPacket(byte[ ] buffer, int offset, int length,
       InetAddress remoteAddr, int remotePort)
public DatagramPacket(byte[ ] buffer, int length,  SocketAddress destination)
public DatagramPacket(byte[ ] buffer, int offset, int length,
       SocketAddress  destination)
```

- in ogni caso un riferimento ad un vettore di byte buffer che contiene i dati da spedire oppure quelli ricevuti, e che poi devono essere copiai nel payload o dal payload
- eventuali informazioni di addressing, se il `DatagramPacket` deve essere spedito

RICEVERE DATI: COSTRUZIONE DATAGRAM

```
public DatagramPacket (byte[ ] buffer, int length)
```

- definisce la struttura utilizzata per memorizzare il pacchetto ricevuto.
- il buffer viene passato vuoto alla receive che lo riempie con il payload del pacchetto ricevuto
- se settato l'offset, la copia avviene a partire dalla posizione indicata
- il parametro length
  - indica il numero massimo di bytes che possono essere copiati nel buffer
  - deve essere minore o uguale di `buffer.length`, altrimenti viene sollevata eccezione
- la copia del payload termina quando
  - l'intero pacchetto è stato copiato
  - oppure quando length bytes sono stati copiati, se il payload è più grande
  - `getLength` restituisce il numero di bytes effettivamente copiati

INVIARE DATI: COSTRUZIONE DEL DATAGRAM

```
public DatagramPacket(byte[] buffer,int length,InetAddress destination,int port)
```

- definisce il `DatagramPacket` da inviare
- length indica il numero di bytes che devono essere copiati dal byte buffer nel pacchetto, a partire dal byte `0` o da offset
  - solleva un'eccezione se `length` è maggiore di `buffer.length`
  - se il byte buffer contiene più di `length` bytes, questi non vengono copiati
- destination + port individuano il destinatario
- molti altri costruttori sono disponibili
- notare che, per essere memorizzato nel buffer, il messaggio deve essere trasformato in una sequenza di bytes. Per generare vettori di bytes:
  - il metodo `getBytes()`
  - la classe `java.io.ByteArrayOutputStream`

LA CLASSE DATAGRAM SOCKETS: COSTRUTTORI

- `DatagramSocket()` crea un socket e lo collega ad una qualsiasi porta libera disponibile sull'host locale

```
try {DatagramSocket client=new DatagramSocket();//send packets}
```

  - utilizzato generalmente lato client, per spedire datagrammi
  - `GetLocalPort( )` per reperire la porta allocata
  - il server può inviare la risposta, prelevando l'indirizzo del mittente (IP+porta) dal pacchetto ricevuto
- `DatagramSocket(int p)` crea un socket e lo collega alla porta specificata, sull'host locale
  - il server crea un socket collegato ad una porta che rende nota ai clients
  - la porta è allocata a quel servizio (porta non effimera)
  - solleva un'eccezione quando la porta è già utilizzata, oppure se non si hanno i diritti

LA CLASSE DATAGRAM SOCKETS: COSTRUTTORI

- `DatagramSocket`: crea un socket e lo collega ad una qualsiasi porta libera disponibile sull'host locale

```
try {
    DatagramSocket client = new DatagramSocket();
     //send packets }
```

- utilizzato generalmente lato client, per spedire datagrammi
- per reperire la porta allocata utilizzare il metodo `getLocalPort( )`
- esempio:
  - un client si connette ad un server mediante un socket collegato ad una porta anonima.
  - il server preleva l'indirizzo del mittente (IP+porta) dal pacchetto ricevuto e può così inviare una risposta.
  - quando il socket viene chiuso, la porta viene utilizzata per altre connessioni.

DECIDERE LA DIMENSIONE DEL DATAGRAMPACKET

- ad ogni socket sono associati due buffers: uno per la ricezione ed uno per la spedizione, gestiti dal sistema operativo, non dalla JVM

```
import java.net.*;
  public class udpproof {
  public static void main (String args[])throws Exception
              {DatagramSocket dgs = new DatagramSocket( );
               int r = dgs.getReceiveBufferSize();
               int s = dgs.getSendBufferSize();
               System.out.println("receive buffer"+r);
               System.out.println("send buffer"+s); } }
```

- stampa prodotta : **receive buffer 8192 send buffer 8192**
- in generale la dimensione massima di un pacchetto è 64k bytes, ma in molte piattaforme è 8k
- pacchetti più grandi vengono in generale troncati
- safety: `DatagramPacket` minori di 512 bytes

I METODI SET

`void setData(byte[ ] buffer)`

- setta il payload di "this" packet, prendendo i dati dal buffer

`void setData(byte[ ] buffer, int offset, int length)`

- setta il payload di "this" packet, prendendo dati da una parte del buffer
- utile quando si deve mandare una grande quantità di dati

```
int offset = 0;
DatagramPacket dp = new DatagramPacket(bigarray, offset, 512);
int bytesSent = 0;
while (bytesSent < bigarray.length) {
      socket.send(dp);
      bytesSent += dp.getLength( );
      int bytesToSend = bigarray.length - bytesSent;
      int size = (bytesToSend > 512) ? 512 : bytesToSend;
      dp.setData(bigarray, bytesSent, size);
```

`void setPort(int iport)`

- setta la porta nel datagram

I METODI SET

`void setLength(int length)`

- setta la lunghezza del payload del `Datagram`

`void setAddress(InetAddress iaddr)`

- setta l'`InetAddress` della macchina a cui il payload è diretto
- utile quando si deve mandare lo stesso `Datagram` a più destinatari

```
DatagramSocket socket= new DatagramSocket();
 String s = "Really important message";
byte [] data = s.getBytes("UTF-8");
DatagramPacket dp = new DatagramPacket(data, data.length);
 dp.setPort(2000);
 String network = "128.238.5.";
 for (int host =1; host <255; host++)
    {InetAddress remote = InetAddress.getByName(network+host);
     dp.setAddress(remote);
     socket.send(dp);
     System.out.println("sent");}
```

I METODI SET

```
void setSocketAddress(SocketAddress addr)
```

- utile per inviare risposte

```
DatagramPacket input = new DatagramPacket(new byte[8192], 8192);
socket.receive(input);
DatagramPacket output = new DatagramPacket ("Hello
                                here".getBytes("UTF-8"),11);
SocketAddress address = input.getSocketAddress();
output.setSocketAddress(address);;
socket.send(output);
```

I METODI GET

`InetAddress getAddress()`
- restituisce l'indirizzo IP della macchina a cui il `Datagram` è stato inviato oppure della macchina da cui è stato spedito

`int getPort( )`
- restituisce il numero di porta sull'host remoto a cui il `Datagram` è stato inviato, oppure della macchina da cui è stato spedito

`byte[] getData()`
- restituisce un byte array contenente i dati del buffer associato al `Datagram`
- ignora offset e lunghezza

`int [] getLength(), int [] getOffset()`
- restituiscono la lunghezza/offset del `Datagram` da inviare o da ricevere

`SocketAddress getSocketAddress()`
- restituisce (IP+numero di porta) del `Datagram` sull'host remoto cui il `Datagram` è stato inviato, o dell'host a cui è stato spedito

INVIARE E RICEVERE DATAGRAM

invio di pacchetti `sock.send (dp)`

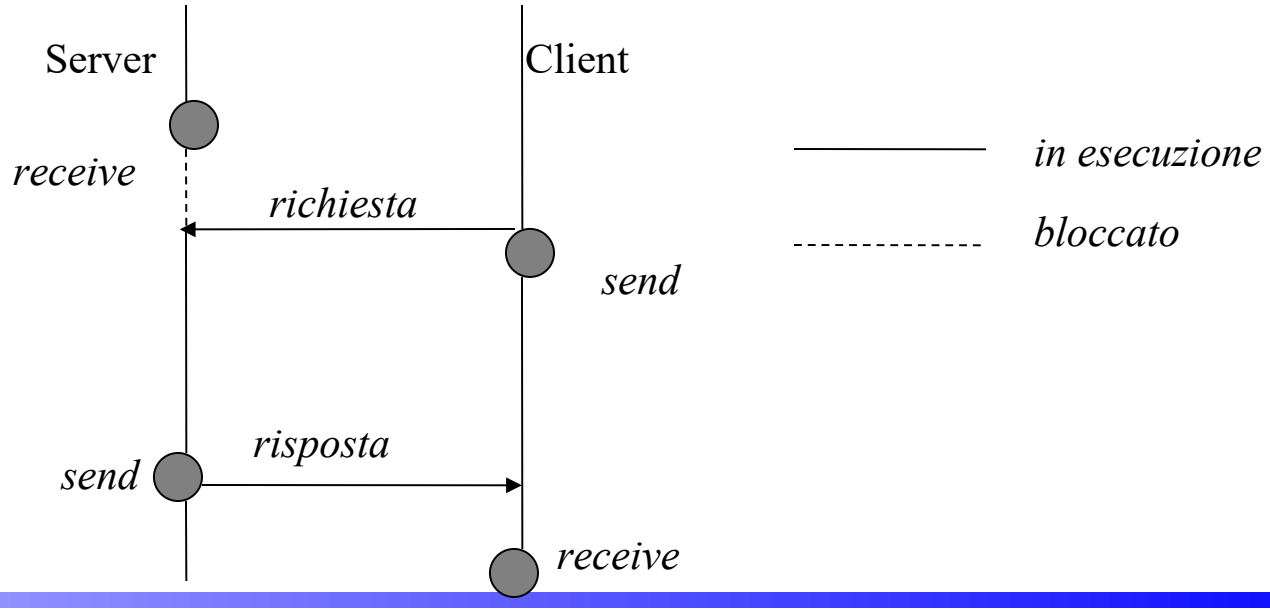
- sock è il socket attraverso il quale voglio spedire il `Datagram` dp

ricezione di pacchetti `sock.receive(dp)`

- sock è il socket attraverso il quale ricevo il `Datagram` dp
- riceve un `Datagram` dal socket
- riempe il buffer associato al socket con i dati ricevuti
- il `Datagram` ricevuto contiene anche indirizzo IP e porta del mittente

COMUNICAZIONE UDP: CARATTERISTICHE

- send non bloccante nel senso che il processo che esegue la send prosegue la sua esecuzione, senza attendere che il destinatario abbia ricevuto il pacchetto
- receive bloccante il processo che esegue la receive si blocca fino al momento in cui viene ricevuto un pacchetto
- Per evitare attese indefinite è possibile associare al socket un timeou.
- Quando il timeout scade, viene sollevata una `InterruptedIOException`



IL SERVIZIO DAYTIME IN UDP

- svilupperemo un daytime client
- il client si collega ad un server noto che offre sulla porta nota il servizio daytime: client in `JAVA`, server su porta nota
- specifica del servizio: `RFC 867`
- successivamente svilupperemo il `DayTimeServer`, un server scritto in `JAVA` per il servizio di `DayTime`

DAYTIME UDP CLIENT: "HOW TO DO"

1. aprire il socket: se si sceglie la porta 0 il sistema sceglie una porta libera "effimera"

```
DatagramSocket socket = new DatagramSocket(0);
```

2. impostare un timeout sul socket, opzionale, ma consigliato

```
setSoTimeout(15000)
```

3. costruire due pacchetti: uno per inviare la richiesta al server, uno per ricevere la risposta

```
InetAddress host = InetAddress.getByName(HOSTNAME);
DatagramPacket request = new DatagramPacket(new byte[1], 1, host , PORT);
byte [] data = new byte[1024];
DatagramPacket response = new DatagramPacket(data, data.length);
```

4. mandare la richiesta ed aspettare la risposta

```
socket.send(request);
socket.receive(response);
```

5. estrarre i byte dalla risposta e convertirli in String

```
String daytime = new String(response.getData(),0,response.getLength(),"Us-ASCII");
System.out.println(daytime);
```

DAYTIME CLIENT

```
import java.io.*;
import java.net.*;
public class DayTimeUDPClient {
  // RFC-867
  private final static int PORT = 13;
  private static final String HOSTNAME = "test.rebex.net";
  public static void main(String[] args) {
    try (DatagramSocket socket = new DatagramSocket(0)) {
       socket.setSoTimeout(15000);
       InetAddress host = InetAddress.getByName(HOSTNAME);
       DatagramPacket request  = new DatagramPacket(new byte[1], 1, host , PORT);
       DatagramPacket response = new DatagramPacket(new byte[1024], 1024);
       socket.send(request);
       socket.receive(response);
       String daytime = new String(response.getData(),0,response.getLength(),"Us-ASCII");
       System.out.println(daytime);
       }
     catch (IOException ex) {  ex.printStackTrace(); }}}
```

try with resources

```
$Java DayTimeUDPClient
Thu, 30 Oct 2025 17:11:15 GMT
```

DAYTIME UDP SERVER

1. aprire un DatagramSocket su una porta "nota" ("well known port")

```
DatagramSocket socket = new DatagramSocket(13)
```

   - porta nota perchè i client devono inviare i packet a quella destinazione
   - a differenza di TCP, stesso tipo di socket per il client e per il server

2. creare un pacchetto in cui ricevere la richiesta del client

```
DatagramPacket request = new DatagramPacket(new byte[1024], 1024);
socket.receive(request);
```

3. creare un pacchetto di risposta

```
String daytime = new Date().toString();
byte[] data = daytime.getBytes("US-ASCII");
InetAddress host = request.getAddress();
int port = request.getPort();
DatagramPacket response = new DatagramPacket(data, data.length, host, port)
```

4. inviare la risosta usando lo stesso socket da cui si è ricevuto il pacchetto

```
socket.send(response);
```

DAYTIME UDP SERVER

```
import java.net.*; import java.util.Date; import java.io.*;
public class DatTimeUDPServer {
  private final static int PORT = 13;
  public static void main(String[] args) {
    try (DatagramSocket socket = new DatagramSocket(PORT)) {
      while (true) {
        try {
          DatagramPacket request = new DatagramPacket(new byte[1024], 1024);
          socket.receive(request);
          System.out.println("ricevuto un pacchetto da"+request.getAddress()+"
                                                          "+request.getPort());
          String daytime = new Date().toString();
          byte[] data = daytime.getBytes("US-ASCII");
          DatagramPacket response = new DatagramPacket(data, data.length,
                                        request.getAddress(), request.getPort());
          socket.send(response);
        } catch (IOException | RuntimeException ex) {ex.printStackTrace();}
      } }
      catch (IOException ex) { ex.printStackTrace();}}}
```

GESTIONE BUFFER RICEZIONE

```
import java.net.*;
public class Sender
   { public static void main (String args[]) {
     try
     {DatagramSocket clientsocket = new DatagramSocket();
     byte[] buffer="1234567890abcdefghijklmnopqrstuvwxyz".getBytes("US-
                                                              ASCII");
      InetAddress address = InetAddress.getByName("Localhost");
      for (int i = buffer.length; i >0; i--) {
          DatagramPacket mypacket = new DatagramPacket(buffer,i,address,
                                                          40000);
          clientSocket.send(mypacket);
          Thread.sleep(200); }
          System.exit(0);}
     catch (Exception e) {e.printStackTrace();}}}
```

DATI INVIATI

Dati inviati dal mittente:

```
Length 36 data 1234567890abcdefghijklmnopqrstuvwxyz
Length 35 data 1234567890abcdefghijklmnopqrstuvwxy
Length 34 data 1234567890abcdefghijklmnopqrstuvwx
Length 33 data 1234567890abcdefghijklmnopqrstuvw
Length 32 data 1234567890abcdefghijklmnopqrstuv
Length 31 data 1234567890abcdefghijklmnopqrstu

……..

Length 5 data 12345
Length 4 data 1234
Length 3 data 123
Length 2 data 12
Length 1 data 1
```

GESTIONE BUFFER RICEZIONE

```
import java.net.*;
public class Receiver
{public static void main(String args[]) throws Exception {
    DatagramSocket serverSock= new DatagramSocket(40000);
    byte[] buffer = new byte[100];
    DatagramPacket receivedPacket = new DatagramPacket(buffer,
                                                    buffer.length);
    while (true) {
        serverSock.receive(receivedPacket);
        String byteToString = new String(receivedPacket.getData(),"US-
                                                          ASCII");
        int l=byteToString.length();
        System.out.println(l);
        System.out.println("Length " + receivedPacket.getLength() +
                    " data " + byteToString);}}}
```

DATI RICEVUTI

```
100
Length 36 data 1234567890abcdefghijklmnopqrstuvwxyz
100
Length 35 data 1234567890abcdefghijklmnopqrstuvwxyz
100
Length 34 data 1234567890abcdefghijklmnopqrstuvwxyz
100
Length 33 data 1234567890abcdefghijklmnopqrstuvwxyz
…..
100
Length 2 data 1234567890abcdefghijklmnopqrstuvwxyz
100
Length 1 data 1234567890abcdefghijklmnopqrstuvwxyz
```

GESTIONE DATI RICEZIONE

- per ricevere correttamente i dati

  individuare i dati disponibili specificando offset e lunghezza

```
String byteToString =    new
String(receivedPacket.getData(),0,receivedPacket.getLength(),"US-ASCII")
```

- sempre specificare lunghezza ed offset dei dati ricevuti, anche se si utlizzano stream

RECEIVE CON TIMEOUT

- `SO_TIMEOUT` proprietà associata al socket, indica l'intervallo di tempo, in millisecondi, di attesa di ogni receive eseguita su quel socket
- nel caso in cui l'intervallo di tempo scada prima che venga ricevuto un pacchetto dal socket, viene sollevata una eccezione di tipo `InterruptedException`
- metodi per la gestione di time out

```
public synchronized void setSoTimeout(int timeout) throws
                                                SocketException
```

esempio: se ds è un datagram socket,

```
ds.setSoTimeout(30000)
```

associa un timeout di 30 secondi al socket ds.

COSTRUZIONE/LETTURA DI VETTORI DI BYTES

- i dati inviati mediante UDP devono essere rappresentati come vettori di bytes
- alcuni metodi per la conversione stringhe/vettori di bytes
  - `Byte[] getBytes()`
    - applicato ad un oggetto String
    - restituisce una sequenza di bytes che codificano i caratteri della stringa usando la codifica di default dell'host e li memorizza nel vettore
  - `String (byte[] bytes, int offset, int length)`
    - costruisce un nuovo oggetto di tipo String prelavando length bytes dal vettore bytes, a partire dalla posizione offset
- altri meccanismi per generare pacchetti a partire da dati strutturati:
  - utilizzare i filtri per generare streams di bytes a partire da dati strutturati/ad alto livello

ASSIGNMENT 9: DUNGEON ADVENTURES

- sviluppare un'applicazione client server in cui il server gestisce le partite giocate in un semplice gioco, "`Dungeon adventures`" basato su una semplice interfaccia testuale
- ad ogni giocatore viene assegnato, ad inizio del gioco, un livello `X` di salute e una quantità `Y` di una pozione, `X` e `Y` generati casualmente
- ogni giocatore combatte con un mostro diverso. Anche al mostro assegnato a un giocatore viene associato, all'inizio del gioco un livello `Z` di salute generato casualmente

ASSIGNMENT 9: DUNGEON ADVENTURES

- il gioco si svolge in round, ad ogni round un giocatore può
  - *combattere con il mostro:* il combattimento si conclude decrementando il livello di salute del mostro e del giocatore. Se `LG` è il livello di salute attuale del giocatore e `MG` quello del mostro, tale livello viene decrementato di un valore casuale `X`, con $0 \leq X \leq LG$. Analogamente, per il mostro si genera un valore casuale `K`, con $0 \leq K \leq MG$.
  - *bere una parte della pozione*, la salute del giocatore viene incrementata di un valore proporzionale alla quantità di pozione bevuta, che è un valore generato casualmente
  - *uscire dal gioco*. In questo caso la partita viene considerata persa per il giocatore
- il combattimento si conclude quando il giocatore o il mostro o entrambi hanno un valore di salute pari a 0.
- se il giocatore ha vinto o pareggiato, può chiedere di giocare nuovamente, se invece ha perso deve uscire dal gioco.

ASSIGNMENT 9: DUNGEON ADVENTURES

- sviluppare una applicazione client server che implementi `Dungeon adventures`
  - il server riceve richieste di gioco da parte dei cliente e gestisce ogni connessione in un diverso thread
  - ogni thread riceve comandi dal client li esegue. Nel caso del comando "combattere", simula il comportamento del mostro assegnato al client
  - dopo aver eseguito ogni comando ne comunica al client l'esito
  - comunica al client l'eventuale terminazione del del gioco, insieme con l'esito
- il client si connette con il server
  - chiede iterativamente all'utente il comando da eseguire e lo invia al server. I comandi sono i seguenti `1:combatti`, `2: bevi pozione`, `3: esci del gioco`
  - attende un messaggio che segnala l'esito del comando
  - nel caso di gioco concluso vittoriosamente, chiede all'utente se intende continuare a giocare e lo comunica al server