Programmazione di reti
Corso B

15 Marzo 2016
Lezione 4

Contenuti

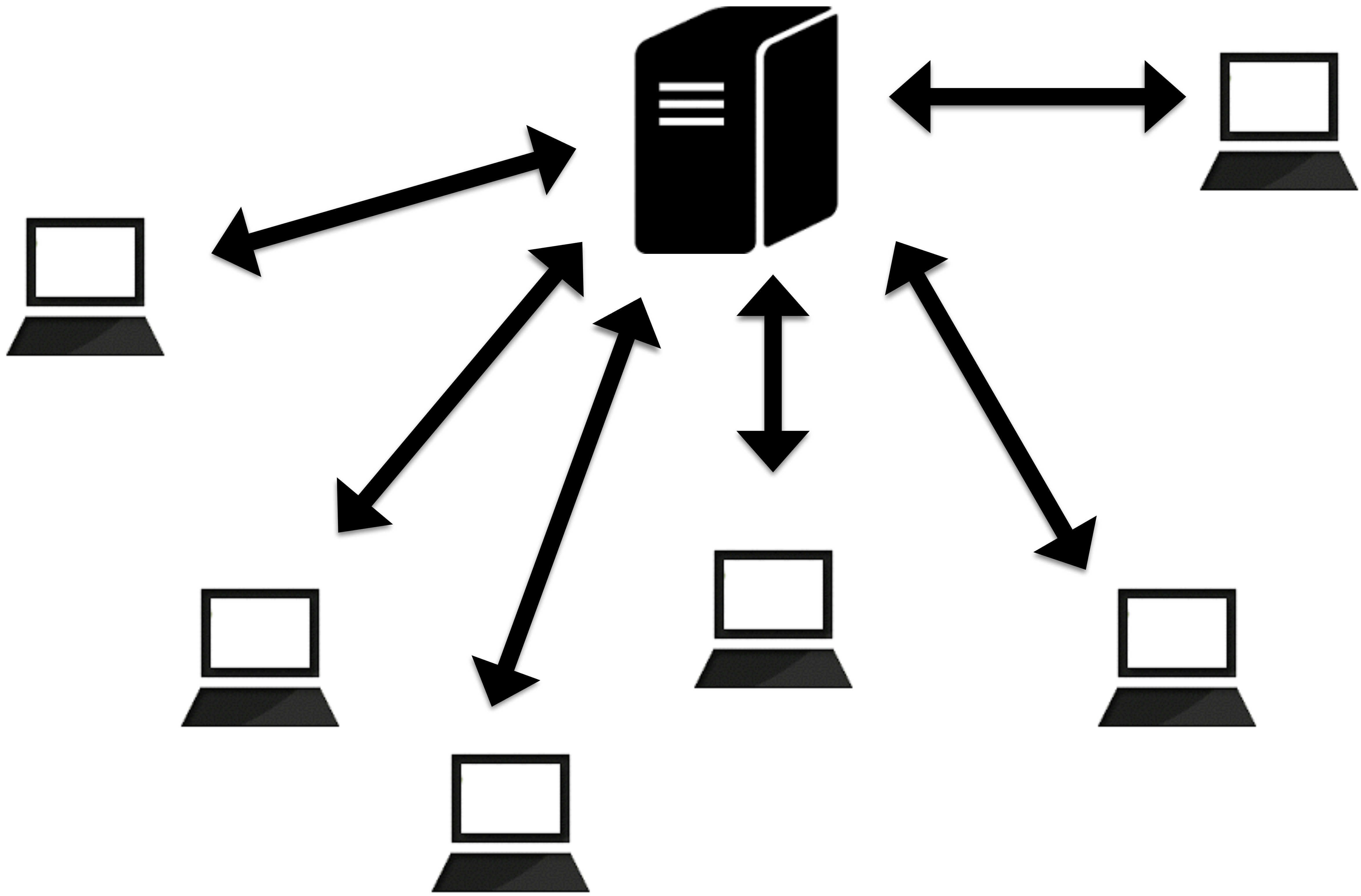
- Programmazione di reti
  - Indirizzi di rete
  - Java IO
  - Java *socket*

Programmazione di reti

- Programmi che comunicano usando la rete
- Paradigmi
  - *Client-server*
  - *Peer-to-peer*

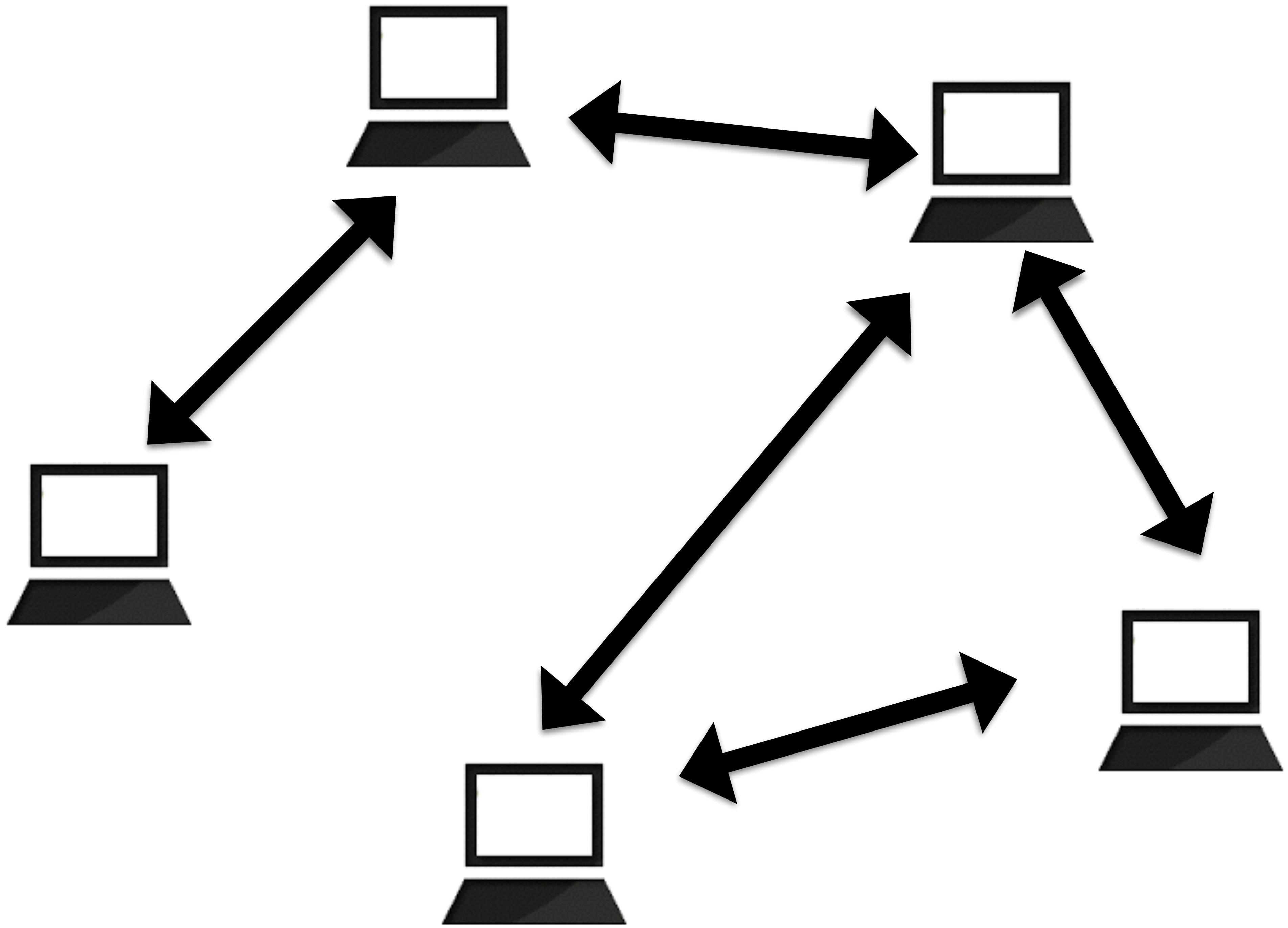
Paradigma *client-server*

- *Server*
  - offre un servizio ai clienti
  - gestisce la logica del programma - ha una visione completa del sistema
- Cliente
  - usa il servizio offerto
  - ha una conoscenza limitata del sistema



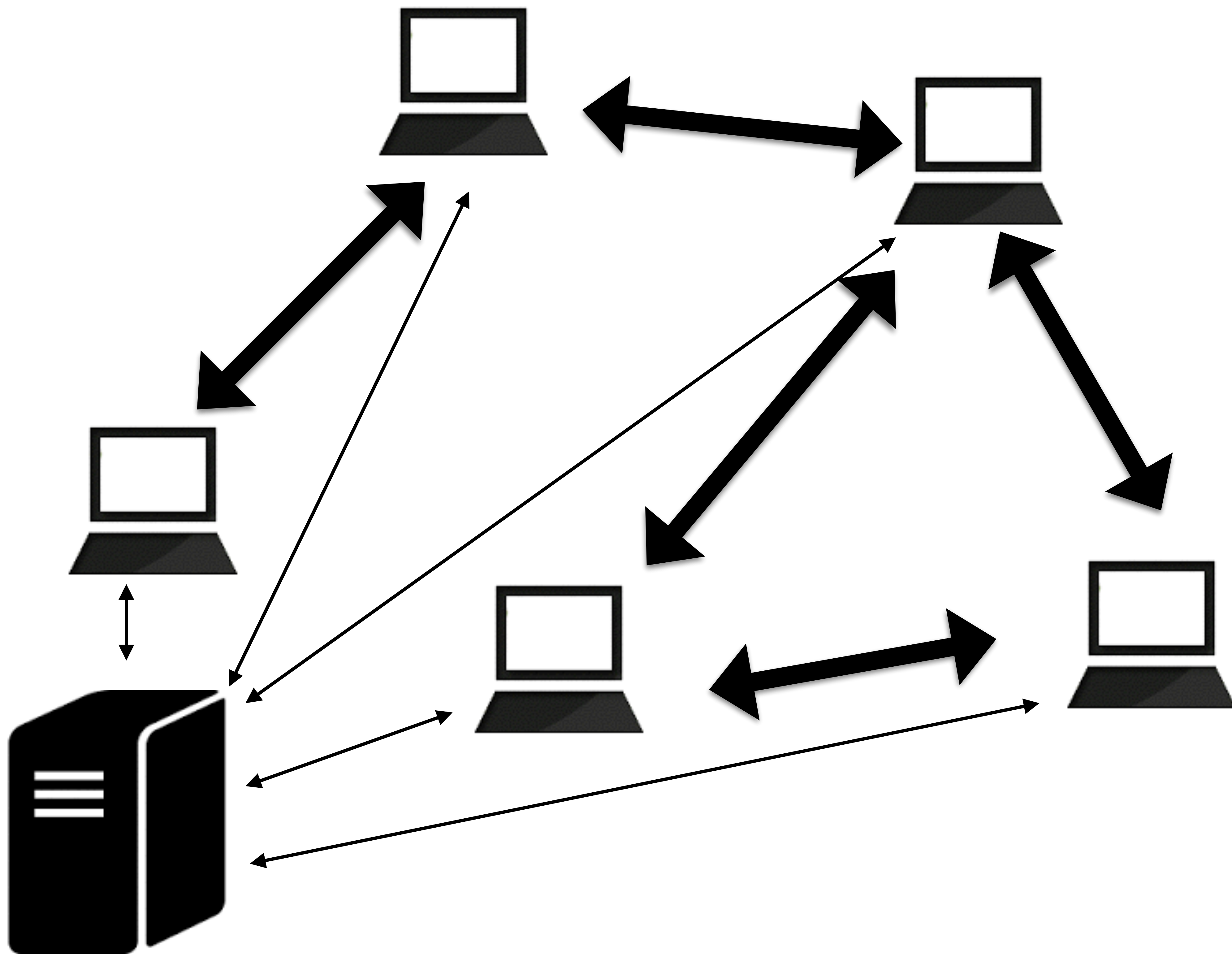
Paradigma *peer-to-peer*

- Tutti i nodi sono uguali
- Applicazioni comunicano direttamente, senza un server
- Tutti i nodi hanno una conoscenza parziale del sistema



Sistemi ibridi

- Un *server* aiuta a trovare i *peer* con cui comunicare
    - La comunicazione avviene *peer-to-peer*
- Super-nodi che fanno anche da *server*



Comunicazione tra applicazioni

- Usando *Socket* - a distanza - usando la rete
- Primo passo : identificare le applicazioni nella rete
- Secondo passo: inviare i dati
    - Si usano pacchetti chiamati *datagram* - contengono indirizzo e dati
    - *Connection-based* : TCP - in ordine - usando s*ocket*
    - *Packet-based* : UDP - messaggi indipendenti, più veloce, meno affidabile

Indirizzi

- ogni host della rete viene identificata dall'IP (IPv4 o IPv6)
- difficili da ricordare => *hostname* più facili (e.g. elearning.unipi.it)
- DNS (*domain name system*) - mappa gli indirizzi IP alla *hostname*
- in Java : `InetAddress`

InetAddress

- Memorizzano l’IP e il *hostname*
- Creati usando metodi statici *factory*

`InetAddress getLocalHost()`
restituisce l’indirizzo e nome del *local host*

`InetAddress  getLoopbackAddress()`
indirizzo 127.0.0.1, nome “localhost”

InetAddress

- `static InetAddress getByName(String name)`

```
InetAddress address =
InetAddress.getByName(“elearning.unipi.it”);
```

Lancia eccezione se hostname non esiste

```
InetAddress address =
InetAddress.getByName(“131.114.18.105”);
```

Non lancia eccezione, mette *hostname*=“131.114.18.105”

- `static InetAddress  getByAddress(byte[] addr)`

```
byte[] rawAddress={(byte)131,114,18,105};
InetAddress address = InetAddress.getByAddress(rawAddress);
```

Non lancia eccezione, mette *hostname*=“131.114.18.105”

```java
public class Address {
   public static void main(String[] args) {
      String stringAddress="131.114.18.105";
      String name="elearning.unipi.it";
      byte[] rawAddress={(byte)216,58,(byte)212,68};
      try {
         InetAddress address = InetAddress.getByName(name);
         System.out.println("Found "+address.getHostName()+
               " with IP "+address.getHostAddress());
         address = InetAddress.getByName(stringAddress);
         System.out.println("Found "+address.getHostName()+
               " with IP "+address.getHostAddress());
         address = InetAddress.getByAddress(rawAddress);
         System.out.println("Found "+address.getHostName()+
               " with IP "+address.getHostAddress());
         address = InetAddress.getLocalHost();
         System.out.println("Running on "+address.getHostName()+
               " with IP "+address.getHostAddress());
         address = InetAddress.getLoopbackAddress();
         System.out.println("Local "+address.getHostName()+
               " with IP "+address.getHostAddress());
      } catch (UnknownHostException e) {
         System.out.println("No such host: "+name);
      } catch (IOException e) {e.printStackTrace();}
   }
}
```

```
Found elearning.unipi.it with IP 131.114.18.105
Found hosting1.sid.unipi.it with IP 131.114.18.105
Found mil01s24-in-f68.1e100.net with IP 216.58.212.68
Running on Alinas-Air.lan with IP 192.168.1.73
Local localhost with IP 127.0.0.1
```

Indirizzi

- le applicazioni di rete usano i *socket* per comunicare
- un *socket* viene identificato da un numero di *port* sulla *host*
- qualche *port* sono riservati per vari servizi/protocolli - si possono comunque usare se non già usati sulla machina
- un IP + un numero di *port* identificano un’applicazione su internet

Java IO

- Inviare/ricevere dati tramite la rete non e così diverso da scrivere/leggere da un *file*
- *Stream* - flussi di dati
    - unidirezionali : *input / output*, ordinati (FIFO)
    - bloccanti - il *thread* si ferma fin che l'operazione finisce
    - il fonte dei dati e astratizzato dall'interfaccia - implementato da classi diverse
        - *file*
        - *bytearray*
        - rete

OutputStream

```
public abstract void write(int b) throws IOException
```

scrive un solo *byte* (meno significativo del int)

```
public void write(byte[] data) throws IOException
public void write(byte[] data, int offset, int
length)
throws IOException
```

scrivono più *byte*

```
public void flush() throws IOException
```

svuota i *buffer* - per evitare *deadlock* e prima di chiudere

```
public void close() throws IOException
```

chiude lo *stream* (mette *end-of-stream* nella coda)

InputStream

```
public abstract int read() throws IOException
```

Legge 1 *byte*, restituisce -1 se *stream* è finito

```
public int read(byte[] input) throws IOException
```

Prova di leggere abbastanza *byte* per riempire l'array. Restituisce il numero di *byte* letti. Può restituire 0, che vuol dire che non ha trovato nessun *byte* ready. Non si blocca.

```
public int read(byte[] input, int offset, int length) throws
IOException
```

Riempie l'array a partire da offset. Se length=0 *end-of-stream* restituisce 0.

```
int read = 0;
int toRead = 1024;
byte[] input = new byte[toRead];
while (read < toRead) {
  int readNow = in.read(input, read, toRead - read);
  if (readNow == -1) break; // end of stream
    read += readNow;
}
```

InputStream

```
public long skip(long n) throws IOException
```
Salta un numero di byte.

```
public int available() throws IOException
```
Il minimo numero di byte available

```
public void close() throws IOException
```
Chiude lo stream

Dati

- Scrivere nei *stream* : ogni tipo di dati deve essere trasformato in un **`byte[]`**

- Leggere dai *stream*: i **`byte[]`** devono essere interpretati (decodificati per ottenere il dato originale)

```java
public class Streams {
   public static void main(String[] args) {
      int integer=2435;
      double real=2456754.6;
      char character='A';
      String name="Alina";
      int nameByteLength=name.getBytes().length;
      FileOutputStream out=null;
      FileInputStream in = null;
      try {
         out= new FileOutputStream("data.dat");
         out.write(int2ByteArray(integer));
         out.write(double2ByteArray(real));
         out.write(int2ByteArray(character));
         out.write(name.getBytes());
         System.out.println("I wrote "+integer+" "+real+" "+character+" "+name);
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }finally{
         try {if (out != null) out.close();
         } catch (IOException e) {}
      }
```

```java
    try {
       in= new FileInputStream("data.dat");

       byte[] bytes=new byte[Integer.BYTES];
       in.read(bytes);
       int readInteger=byteArray2Int(bytes);

       bytes=new byte[Long.BYTES];
       in.read(bytes);
       double readReal=byteArray2Double(bytes);

       bytes=new byte[Integer.BYTES];
       in.read(bytes);
       char readCharacter=(char)(byteArray2Int(bytes));

       bytes=new byte[nameByteLength];
       in.read(bytes);
       String readName=new String(bytes);

       System.out.println("I read "+readInteger+" "+readReal+" "+readCharacter+"
"+readName);
    } catch (FileNotFoundException e) {e.printStackTrace();
    } catch (IOException e) {e.printStackTrace();
    }finally{try {
          if (in != null)
             in.close();} catch (IOException e) {}
    }}}
```

```java
public static byte[] int2ByteArray(int x)
{
   byte[] result= new byte[Integer.BYTES];
   for (int i=0;i<Integer.BYTES;i++)
   {
      result[Integer.BYTES-i-1]=(byte) (x>>(i*8));
   }
   return result;
}

public static byte[] double2ByteArray(double x)
{
   byte[] result= new byte[Long.BYTES];
   long bits=Double.doubleToLongBits(x);
   for (int i=0;i<Long.BYTES;i++)
   {
      result[Long.BYTES-i-1]=(byte) (bits>>(i*8));
   }
   return result;
}
```

```java
public static int byteArray2Int(byte[] bytes){
   int result=0;
   for (byte b : bytes){
      result=(result<<8)+(b>=0?b:(b+256));
   }
   return result;
}

public static double byteArray2Double(byte[] bytes){
   long result=0;
   for (byte b : bytes){
      result=(result<<8)+(b>=0?b:(b+256));
   }
   return Double.longBitsToDouble(result);
}
```

```
Output:
I wrote 2435 2456754.6 A Alina
I read 2435 2456754.6 A Alina
```

Java IO

- I *filter* - usati con stream per trasformare dati prima di scrivere/ dopo aver letto
  - criptare/decriptare
  - comprimere
  - ottimizzare usando *buffer*
- I *reader/writer* - specializzati sul lavoro su testo

Filter

```
BufferedOutputStream out = new
BufferedOutputStream(new
FileOutputStream(“data.txt”), int 1024);
```

Usa un *buffer* di 1kb. Migliore performance. La scrittura può essere forzata facendo `flush()`. Simile per `BufferedInputStream`.

```
CipherOutputStream out = new
CipherOutputStream(new BufferedOutputStream(new
FileOutputStream(“data.txt”)),Cipher.getInstance
(“AES/CBC/NoPadding"));
```

Codifica i dati usando AES, poi usa un buffer sul stream grezzi.

Filter classes

```
DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(“data.dat”)));
```

Metodi per scrivere vari tipi di dati

```
public final void writeBoolean(boolean b) throws IOException
public final void writeByte(int b) throws IOException
public final void writeShort(int s) throws IOException
public final void writeChar(int c) throws IOException
public final void writeInt(int i) throws IOException
public final void writeLong(long l) throws IOException
public final void writeFloat(float f) throws IOException
public final void writeDouble(double d) throws IOException
public final void writeChars(String s) throws IOException
public final void writeBytes(String s) throws IOException
public final void writeUTF(String s) throws IOException
```

`DataInputStream` legge gli stessi tipi di dati.

Reader/Writer

- Simili a *Input/OutputStream* però lavorano con caratteri **Unicode** (lunghezza dipende dalla codifica) invece di *byte*

- opzione migliore per lavorare con testo - indipendenza dalla piattaforma e supporto a vari tipi di *encoding*

- *encoding* - mappa un carattere a un numero (un codice)

- il codice di un carattere può essere diverso in *encoding* diversi

Writer

```
void write(char[] text, int offset, int length) throws
IOException
void write(int c) throws IOException
void write(char[] text) throws IOException
void write(char[] text, int offset, int length) throws
IOException
void write(String s) throws IOException
void write(String s, int offset, int length) throws
IOException
void flush() throws IOException
void close() throws IOException
```

Interfaccia quasi uguale al `OutputStream`, però avendo il `char` alla base.

Reader

```
abstract int read(char[] text, int offset, int length)
throws IOException
int read() throws IOException
int read(char[] text) throws IOException
long skip(long n) throws IOException
void close() throws IOException
```

metodi simili agli `InputStream`

```
boolean ready()
```

controlla se c'è qualcosa da leggere, non quanti caratteri ci sono

Reader/Writer

Da usare senza un *stream* grezzo:

`FileReader/Writer`
scrivere/leggere testo da un *file*

`StringReader/Writer`
scrivere/leggere testo da una stringa

`CharArrayReader/Writer`
scrivere/leggere testo da un *array* di `char`

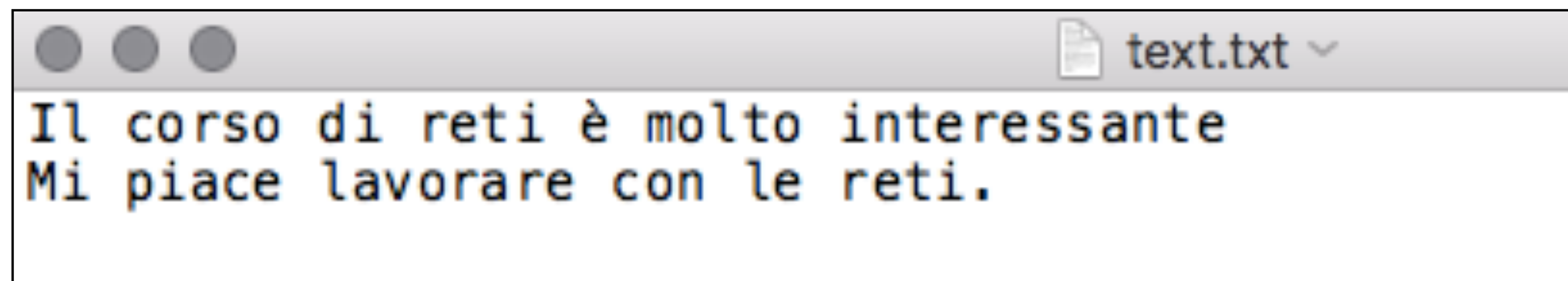
`BufferedReader/Writer`
funzionalità di buffer sopra *reader/writer* grezzi

OutputStreamWriter / InputStreamReader

Da usare con un stream grezzo sotto. Traduce byte in caratteri in base alla encoding usata.

```
public OutputStreamWriter(OutputStream out)
public OutputStreamWriter(OutputStream out, String encoding)
throws UnsupportedEncodingException

public InputStreamReader(InputStream in)
public InputStreamReader(InputStream in, String encoding)
throws UnsupportedEncodingException

public String getEncoding()
```

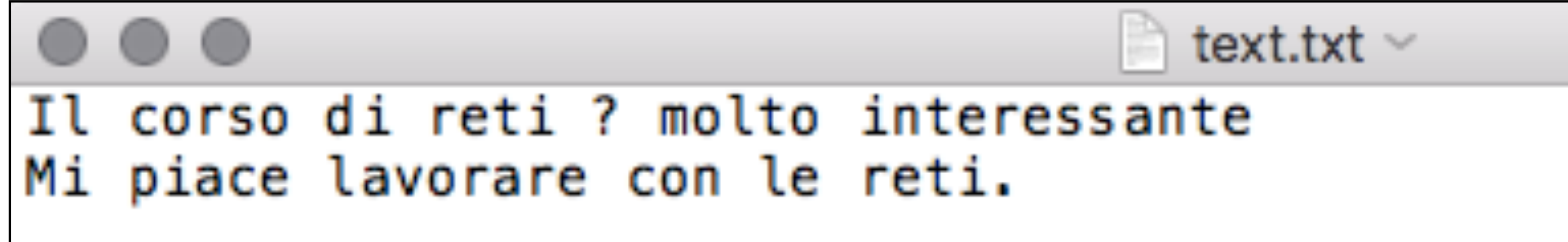
```java
public class WriterTest {

   public static void main(String[] args) {
      BufferedWriter w=null;
      try {
         w= new BufferedWriter(new OutputStreamWriter(
           new FileOutputStream("text.txt")));
         w.write("Il corso di reti è molto interessante\r\n”);
         w.write("Mi piace lavorare con le reti.");
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }finally{
         try {w.close();} catch (IOException e) {}
      }
   }
}
```



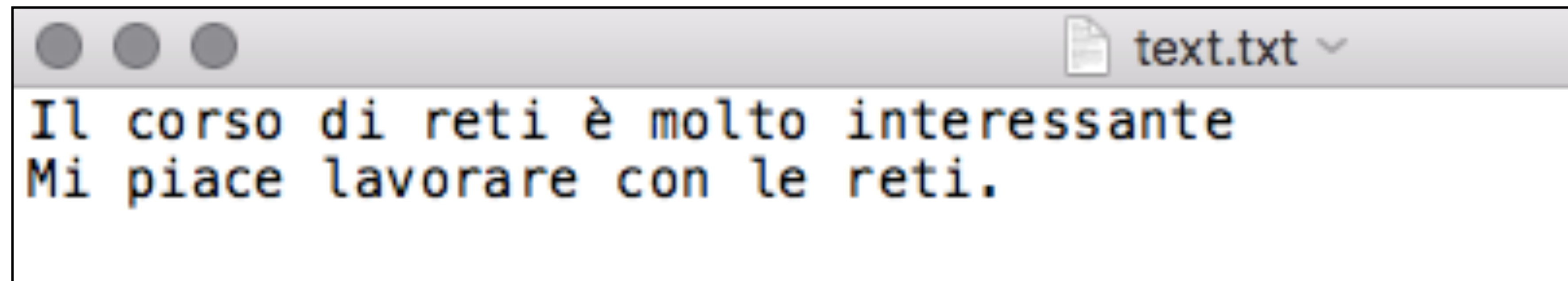
```java
public class ReaderTest {

   public static void main(String[] args) {
      BufferedReader r=null;
      try {
         r= new BufferedReader(new InputStreamReader
            (new FileInputStream("text.txt")));
         String line= r.readLine();
         while (line!=null){
            System.out.println("Read: "+line);
            line=r.readLine();
         }
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }finally{
         try {r.close();} catch (IOException e) {}
      }
   }

}
```

```
Read: Il corso di reti è molto interessante
Read: Mi piace lavorare con le reti.
```

```java
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("text.txt"),"US-ASCII"));
```



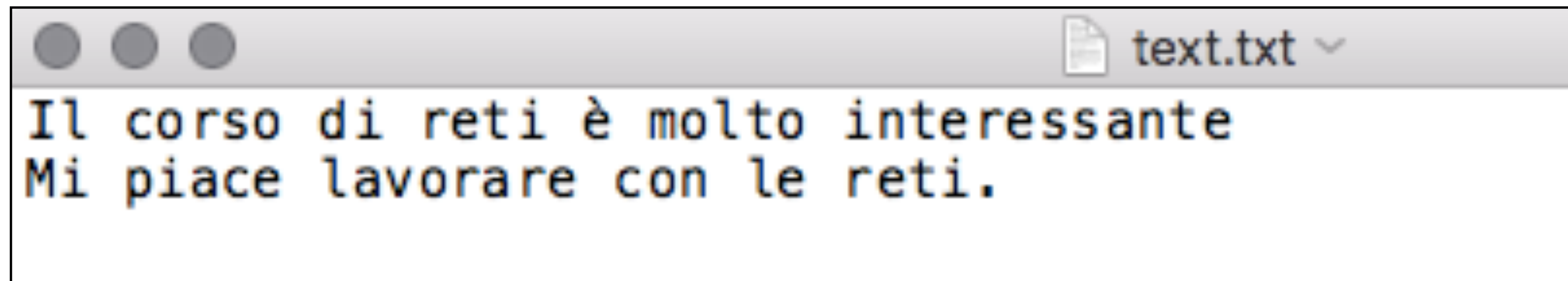
```
w= new BufferedWriter(new OutputStreamWriter(
      new FileOutputStream("text.txt"),"UTF-8"));
```




```
r= new BufferedReader(new InputStreamReader
    (new FileInputStream("text.txt"),"UTF-16"));
```

```
Read:  鏄潲獯椠牥瑩?ら浯汴漠楮瑥獳慮瑥൯ൡ黧⁰楡捥慶潲慲攠捯渠汥?整椮
```

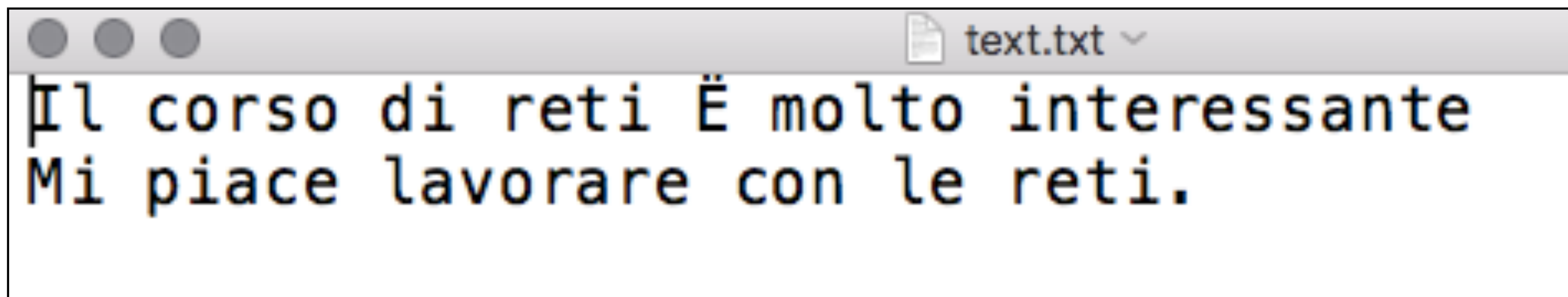
```
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("text.txt"),"UTF-16"));
```




```
r= new BufferedReader(new InputStreamReader
   (new FileInputStream("text.txt"),"UTF-8"));
```

```
Read: ??Il corso di reti ? molto interessante
Read:
Read: Mi piace lavorare con le reti.
```

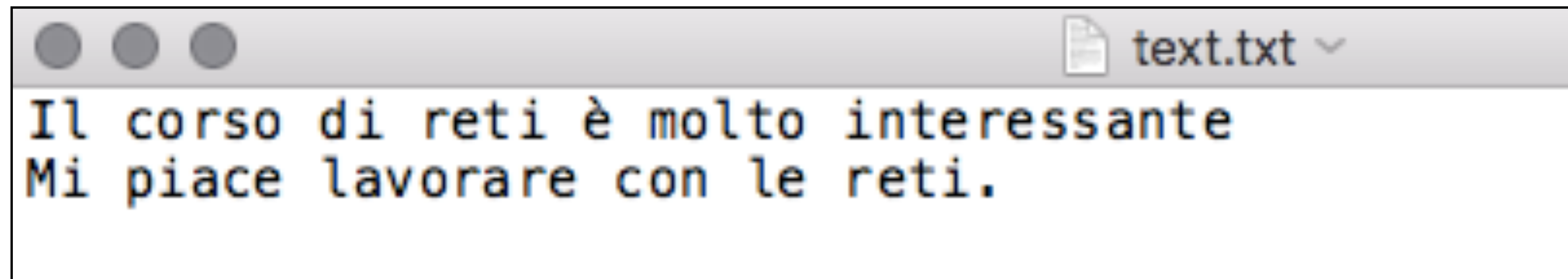
```
w= new BufferedWriter(new OutputStreamWriter(
      new FileOutputStream("text.txt"),"windows-1252"));
```




```
r= new BufferedReader(new InputStreamReader
    (new FileInputStream("text.txt"),"UTF-8"));
```

```
Read: Il corso di reti ? molto interessante
Read: Mi piace lavorare con le reti.
```

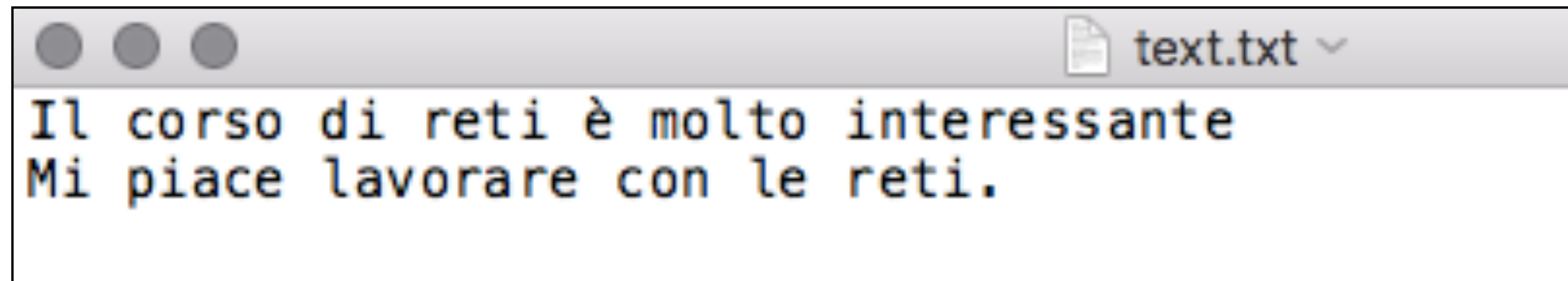
```
w= new BufferedWriter(new OutputStreamWriter(
      new FileOutputStream("text.txt"),"UTF-8"));
```




```
r= new BufferedReader(new InputStreamReader
   (new FileInputStream(“text.txt"),"windows-1252"));
```

```
Read: Il corso di reti Ã¨ molto interessante
Read: Mi piace lavorare con le reti.
```

```
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("text.txt"),"UTF-8"));
```




```
r= new BufferedReader(new InputStreamReader
    (new FileInputStream(“text.txt"),"Big5"));
```

```
Read: Il corso di reti 癡 molto interessante
Read: Mi piace lavorare con le reti.
```

Comunicazione TCP

- Tramite *socket*
- un *socket* è una estremità (end-point) di una connessione bidirezionale tra due programmi eseguiti nella rete
- Classe `java.net.Socket` usata da cliente e *server*
- Classe `java.net.ServerSocket` usata dal *server*

Comunicazione TCP

- Passi:
  - Il *server* apre un `ServerSocket` e aspetta connessioni dai clienti
  - Il cliente apre un `Socket` e si connette al *server*
  - Quando un cliente arriva, il *server* crea un `Socket` dedicato a questo cliente
  - *Server* e client comunicano in base al protocollo stabilito, usando sempre `Socket` (`ServerSocket` serve solo per aspettare clienti e iniziare la connessione)

Socket

- Creare un *socket* e aprire una connessione

```
Socket(InetAddress address, int port)

Socket(InetAddress address, int port, InetAddress
localAddr, int localPort)
```

  - localAddr=null => qualsiasi indirizzo locale, localPort=0 => qualsiasi port locale
  - lancia eccezioni

```
Socket(String address, int port)

Socket(String address, int port, InetAddress localAddr,
int localPort)
```

Socket

- Creare un *socket* senza aprire una connessione

```
Socket()
```

- Aprire la connessione con metodo `connect()` - dopo aver creato un `SocketAddress`

```
Socket socket = new Socket();
SocketAddress address = new
InetSocketAddress("www.google.com", 80);
socket.connect(address);
```

```java
public class SocketProps{
   public static void main(String[] args) {
      Socket socket = new Socket();
      try {
         SocketAddress address = new InetSocketAddress("www.google.com", 80);
         socket.connect(address);
         System.out.println("Connected to " + socket.getInetAddress()
         + " on port " + socket.getPort() + " from port "
         + socket.getLocalPort() + " of "
         + socket.getLocalAddress());
      } catch (UnknownHostException ex) {
      System.err.println("I can't find teh host ");
      } catch (IOException ex) {
      System.err.println(ex);
      } finally {
         try {
            socket.close();
         } catch (IOException e) {}
      }
   }
}
```

```
Connected to www.google.com/172.217.16.4 on port 80 from port 58386 of /131.114.218.134
```

Socket

- Fissare un *timeout* : evita blocchi causati da un server non-responsive

```
void setSoTimeout(int timeout)
```

  - lancia `java.net.SocketTimeoutException` se un’operazione (read/write) prende più di timeout, pero il *socket* rimane valido

Socket

- Chiudere la connessione - rilascia il *port* sulla *host*

```
void close() throws IOException


void shutdownInput() throws IOException
void shutdownOutput() throws IOException
```

Socket

- Altre opzioni:

`void setTcpNoDelay(boolean on) throws IOException`
invia dati subito dopo `send()` senza aspettare che pacchetto sia pieno

`void setSoLinger(boolean on, int seconds) throws IOException`
se ci sono pacchetti da inviare dopo `Socket.close()`, questi vengono inviati comunque se *linger* e OFF. Se ON, i pacchetti vengono scartati dopo aver aspettato il numero di secondi (metodo `close()` si blocca).

```
void setReceiveBufferSize(int size)
throws IOException, IllegalArgumentException
void setSendBufferSize(int size)
throws IOException, IllegalArgumentException
```
cambia la dimensione dei buffer - sono sempre uguali - può essere ignorato

Socket

- Altre opzioni:

`public void setKeepAlive(boolean on) throws IOException`

Attiva verifiche di *keepalive*. Se *server* non risponde dopo qualche minuto il *socket* viene chiuso

`public void setReuseAddress(boolean on) throws IOException`

Dopo aver chiuso il *socket*, il *port* diventa riutilizzabile da subito. Deve essere richiamato prima di `connect()`

Verificare connettività

`boolean isClosed()`

`true` se il *socket* e stato chiuso

`boolean isConnected()`

`true` se il *socket* è mai stato connesso

per verificare se un *socket* ha una connessione attiva:

```
isConnected() && ! isClosed()
```

Socket

- Ottenere i *stream* per inviare e ricevere messaggi

```
InputStream  getInputStream()

OutputStream getOutputStream()
```

- Connessione è *full duplex*
- Gli *input/output stream* sono standard: usati come prima, abbinandoli con *filter* o *reader/writer*
- Protocollo viene definito dal programmatore

ServerSocket

- Un *socket* speciale che aspetta delle connessioni
- Non sa in anticipo ne chi sono i clienti ne quando arriveranno
- Con `Socket` se l'altra parte della connessione non è pronta, il *socket* non si connette
- il `ServerSocket` esiste ed è attivo anche senza nessun cliente
- i `ServerSocket` non sono usati per trasmettere dati dall'applicazione, servono solo ad aspettare, negoziare la connessione con i nuovi clienti e creare un `Socket` normale per usare nell'applicazione.

`ServerSocket`

- *Server workflow*
  - Crea un oggetto di tipo `ServerSocket` associato ad un port locale
  - Aspetta che i clienti arrivino
  - Quando un cliente arriva, crea un oggetto di tipo `Socket` connesso al cliente
  - Comunica col cliente
  - Chiude il `Socket`
  - Aspetta di nuovo

`ServerSocket`

- *Server workflow* con *multithreading*
  - Crea un oggetto di tipo `ServerSocket` associato ad un port locale
  - Aspetta che i clienti arrivino
  - Quando un cliente arriva, crea un oggetto di tipo `Socket` connesso al cliente
  - Crea un *task* responsabile con la gestione del cliente e lo avvia in un *thread*
  - Aspetta di nuovo
- Il *task* che gestisce il cliente
  - Comunica col cliente
  - Chiude il `Socket`

Creare `ServerSocket`

- *Socket* già associato con un port (bound)

```
public ServerSocket(int port)
```

  - usare 0 per scegliere un port a caso

```
public ServerSocket(int port, int backlog)
```

  - *backlog* indica il numero di richieste da mettere in coda (le altre vengono respinte). Il numero può essere ignorato o limitato.

```
public ServerSocket(int port,int backlog, InetAddress
localAddress)
```

  - associa al *socket* solo l'interfaccia specificata (altrimenti il *server* ascolta su tutte le interfacce attive)

Creare `ServerSocket`

- Socket non associato con un port (unbound) - bisogna richiamare metodo `bind()`
  - `public ServerSocket()`
  - `public void bind(SocketAddress endpoint)`

Aspettare connessioni

`public Socket accept()`

- blocca fin che un cliente arriva
- restituisce un `Socket` connesso al cliente
- Il nuovo *socket* funziona esattamente come il *socket* cliente di prima, può essere utilizzato per inviare o ricevere dati

Metodi

`public void close()`
chiude il *server socket* e anche i *client socket* generati

`public boolean isClosed()`
`true` se il *socket* e chiuso

`public boolean isBound()`
`true` se *socket* ha mai fatto `bind` (anche nel costruttore)

Per verificare se un *server socket* è aperto e attivo,
dobbiamo usare `isBound() && !isClosed()`

Metodi

`public InetAddress getInetAddress()`
`public int getLocalPort()`

`public void setSoTimeout(int timeout) throws SocketException`

Imposta il tempo per cui `accept()` si blocca. Lancia `SocketTimeoutException` se nessun cliente arriva , però il *socket* rimane attivo.

`public void setReuseAddress(boolean on) throws SocketException`

Se attivata, dopo aver chiuso il *socket*, il port diventa riutilizzabile da subito. Deve essere richiamato prima di `bind()`

Eccezioni

Vari metodi dichiarano di lanciare `IOException`. In realtà, lanciano sottoclassi di `IOException`

```
class SocketException extends IOException
```

```
class BindException extends SocketException
```

se *socket* e già usato o in (0,1024) e non sono root (su Unix)

```
class ConnectException extends SocketException
class NoRouteToHostException extends SocketException
class ProtocolException extends IOException
```

```
class SocketTimeoutException extends InterruptedIOException
```

timeout su `ServerSocket.accept()`

si possono gestire tutti i tipi di eccezioni per avere un controllo dettagliato

```
public class EchoServer {
   public static void main(String[] args) {
      ServerSocket server= null;
      try {
         server = new ServerSocket();
         server.bind(new InetSocketAddress(InetAddress.getLocalHost(), 1500));
         while (true){
            System.out.println("Waiting for clients");
            Socket client=server.accept();
            BufferedReader reader=null;
            BufferedWriter writer=null;
            String message;
            try{
               reader= new BufferedReader(new InputStreamReader(
                     client.getInputStream()));
               writer= new BufferedWriter(new OutputStreamWriter(
                     client.getOutputStream()));
               while ((message= reader.readLine() )!=null){
                     System.out.println("Client sent: "+message);
                     writer.write(message+"\r\n");
                     writer.flush();
               }
```

```
            catch (IOException e){
              System.out.println("Client closed connection or some error appeared");
            } finally{
              if (writer!=null) writer.close();
              if (reader!=null) reader.close();
              client.close();
            }
        }
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally{
        if (server!=null)
            try {server.close();
            } catch (IOException e) {}
    }
  }
}
```

```
public class EchoClient {
  public static void main(String[] args) {
    Socket socket=new Socket();
    BufferedReader reader=null;
    BufferedWriter writer=null;
    try{
      socket.setSoTimeout(100000);
      socket.setTcpNoDelay(true);
      socket.connect(new InetSocketAddress(
        InetAddress.getLocalHost(),1500));
      reader= new BufferedReader(new InputStreamReader(
          socket.getInputStream()));
      writer= new BufferedWriter(new OutputStreamWriter(
          socket.getOutputStream()));
      BufferedReader localReader= new BufferedReader(
        new InputStreamReader(System.in));
      System.out.println("Type 'exit' to stop, any message to send
to server:");
```

```
            String reply="";
            String choice;
            while (!(choice= localReader.readLine().trim()).equals("exit"))
            {
                    writer.write(choice+"\r\n");
                    writer.flush();
                    reply= reader.readLine();
                    System.out.println("Server sent back: "+reply);
            }
            System.out.println("Communication ended by client “);
        } catch (SocketException e) {
            System.out.println("Server closed connection or an error appeared.");
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Server closed connection or an error appeared.");
        }finally{
            try {
                if (reader!=null) reader.close();
                if (writer!=null) writer.close();
                socket.close();
            }catch (IOException e) {}
        }
    }
}
```

`try` con risorse

```
ServerSocket server= null;
try {
 ………………….
} catch (Exception e) {
        e.printStackTrace();
     } finally{
        if (server!=null)
          try {server.close();
          } catch (IOException e) {}
     }
```

Funziona per tutte le risorse che implementano interfaccia AutoCloseable

```
try (ServerSocket server= new ServerSocket();) {
   ……….. .
   } catch (Exception e) {
        e.printStackTrace();
   }
```

```java
public class EchoServer {
    public static void main(String[] args) {
        try (ServerSocket server= new ServerSocket();) {
            server.setReceiveBufferSize(100);
            server.bind(new InetSocketAddress(InetAddress.getLocalHost(), 1500));
            while (true){
                System.out.println("Waiting for clients");
                String message;
                try (Socket client=server.accept();
                        BufferedReader reader=new BufferedReader(new InputStreamReader(
                                client.getInputStream()));
                        BufferedWriter writer= new BufferedWriter(new OutputStreamWriter(
                                client.getOutputStream()));){
                    while ((message= reader.readLine() )!=null){
                        System.out.println("Client sent: "+message);
                        writer.write(message+"\r\n");
                        writer.flush();
                    }
                } catch (IOException e){
                    System.out.println("Client closed connection or some error appeared");
                }
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }}}
```

Tempo di risposta del server

- I clienti sono gestiti uno alla volta
- Se un cliente invia messaggi all’infinito, gli altri non avranno mai risposta (vanno in *timeout*)
- Soluzione: *multithreaded server*

```java
public class MultithreadedEchoServer {

    public static void main(String[] args) {
        try (ServerSocket server = new ServerSocket()) {
            server.bind(new InetSocketAddress(InetAddress.getLocalHost(), 1500));
            while (true){
                try { //not try with resources
                    Socket client=server.accept();
                    EchoClientHandler handler = new EchoClientHandler(client);
                    new Thread(handler).start();
                } catch (IOException e){
                    System.out.println("Some error appeared");
                }
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class EchoClientHandler implements Runnable{
    Socket client;

    public EchoClientHandler(Socket client) {
        this.client=client;
    }
    @Override
    public void run() {
        try (BufferedReader reader= new BufferedReader(
                    new InputStreamReader(this.client.getInputStream()));
                BufferedWriter writer= new BufferedWriter(
                    new OutputStreamWriter(this.client.getOutputStream()));){
            this.client.setSoTimeout(200000);
            String message="";
            while ((message= reader.readLine())!=null){
                    System.out.println("Client sent: "+message);
                    writer.write(message+"\r\n");
                    writer.flush();
            }
        } catch (IOException e){
            System.out.println("Client closed connection or an error appeared.");
        } finally{
            try {this.client.close();
            } catch (IOException e) {}
        }}}
```

DoS attack

- Facendo un *thread* per ogni cliente, il numero di *thread* può crescere velocemente
- il *server* diventa lento o non risponde più
- soluzione: *thread pool*
- in questo caso il nostro *server* è comunque suscettibile a DoS- basta creare dei clienti che non si fermano mai di inviare messaggi
- si può risolvere impostando un numero massimo di messaggi per cliente

```java
public class ThreadPoolEchoServer {
   public static void main(String[] args) {
      ExecutorService es=null;
      try (ServerSocket server = new ServerSocket()) {
         server.bind(new InetSocketAddress(InetAddress.getLocalHost(), 1500));
         es= Executors.newFixedThreadPool(100);
         while (true){
            try { //not try with resources
               Socket client=server.accept();
               EchoClientHandler handler = new EchoClientHandler(client);
               es.submit(handler);
            } catch (IOException e){
               System.out.println("Some error appeared");
            }
         }
      } catch (UnknownHostException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      } finally{
         if (es != null)
            es.shutdown();
      }
   }
}
```

```java
public class EchoClientHandler implements Runnable{
    static int MAX_MSG=10;
    Socket client;
    int messageCount;

    public EchoClientHandler(Socket client) {
        this.client=client;
        this.messageCount=0;
    }
    @Override
    public void run() {
        try (BufferedReader reader= new BufferedReader(
                new InputStreamReader(this.client.getInputStream()));
                BufferedWriter writer= new BufferedWriter(
                        new OutputStreamWriter(this.client.getOutputStream()));){
            this.client.setSoTimeout(200000);
            String message="";
            while (this.messageCount<MAX_MSG && (message= reader.readLine())!=null){
                this.messageCount++;
                System.out.println("Client sent: "+message);
                String responseFooter="";
                if(this.messageCount==MAX_MSG)
                    responseFooter=" This was your last message";
                writer.write(message+responseFooter+"\r\n");
                writer.flush();
            }
        } catch (IOException e){
            System.out.println("Client closed connection or an error appeared.");
        } finally{ try {this.client.close();
            } catch (IOException e) {}}}}
```

Esercizi

- PortScanner
- Scrivere un programma che trova tutti i port aperti su una macchina.
- Il programma prende l'IP o il hostname della macchina dalla linea di commando, e prova tutti i port, per verificare quali sono aperti.
- Per quelli aperti scrive un messaggio sullo schermo.

Esercizi

- MiniFTP
- Scrivere un servizio di trasferimento di file:
- Il cliente invia al server il nome di un file (di testo), preso dalla linea di comando.
- Il server risponde spedendo al cliente il contenuto del file, riga per riga.
- Il cliente salva il contenuto in un file locale con lo stesso nome.
- Gestire situazioni di errore (file not found, etc)