



**Università degli Studi di
Bologna
Scuola di Ingegneria**

Corso di Reti di Calcolatori T

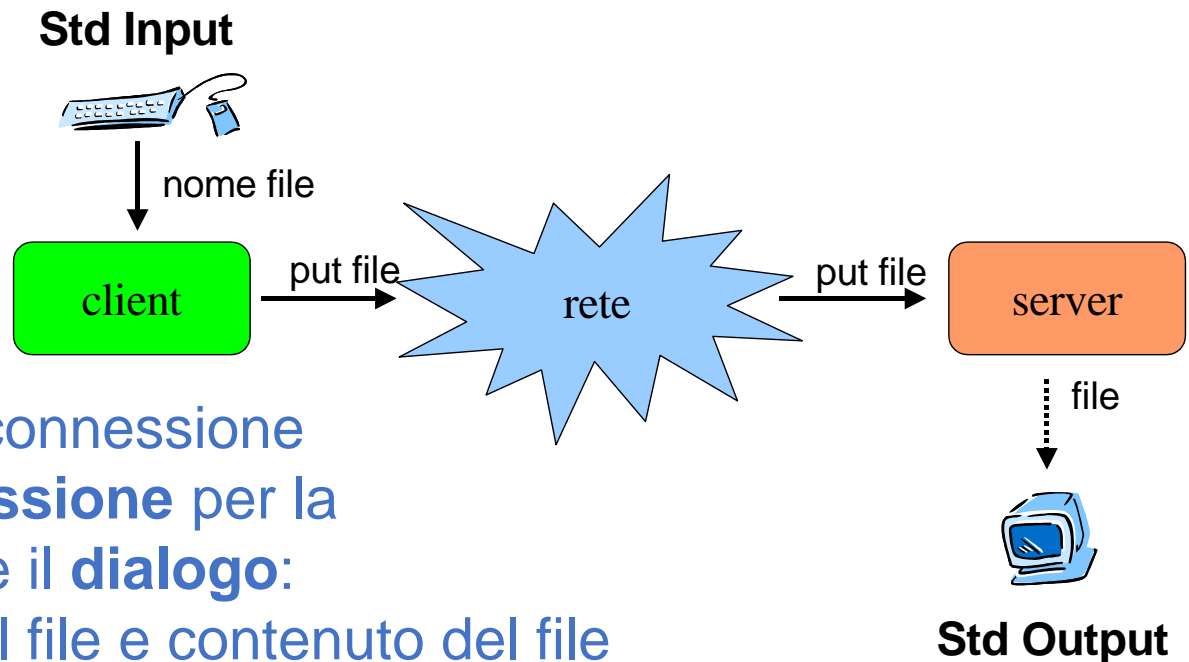
Esercitazione 2 (svolta) Socket Java con connessione

**Antonio Corradi, Armir Bujari
Giuseppe Martuscelli, Lorenzo Rosa,
Andrea Sabbioni**

Anno accademico 2023/2024

ARCHITETTURA DI SUPPORTO A UN TRASFERIMENTO FILE: SERVER SEQUENZIALE

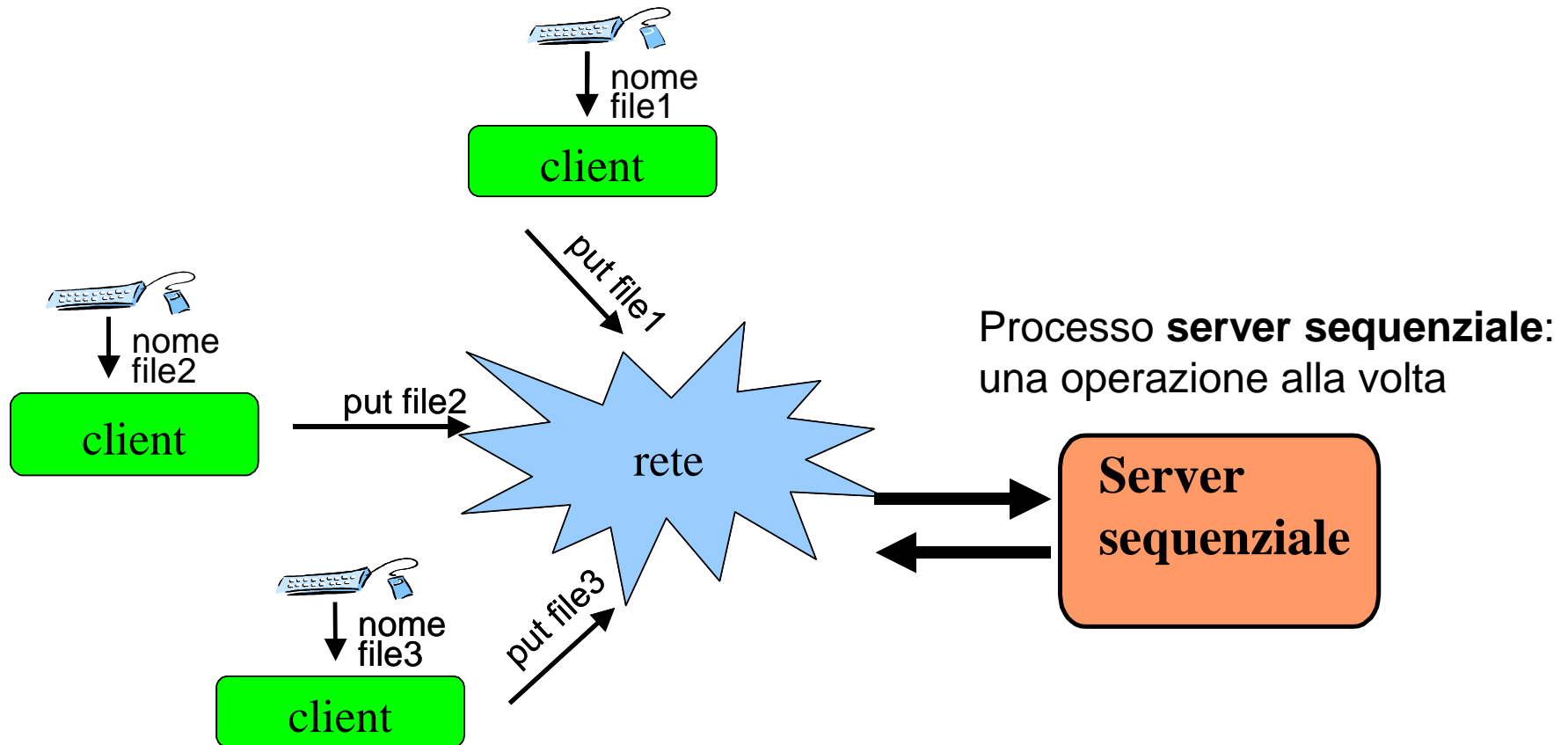
Si vogliono trasferire file binari da una macchina mittente ad una altra destinataria con un processo client e un servitore sul destinatario



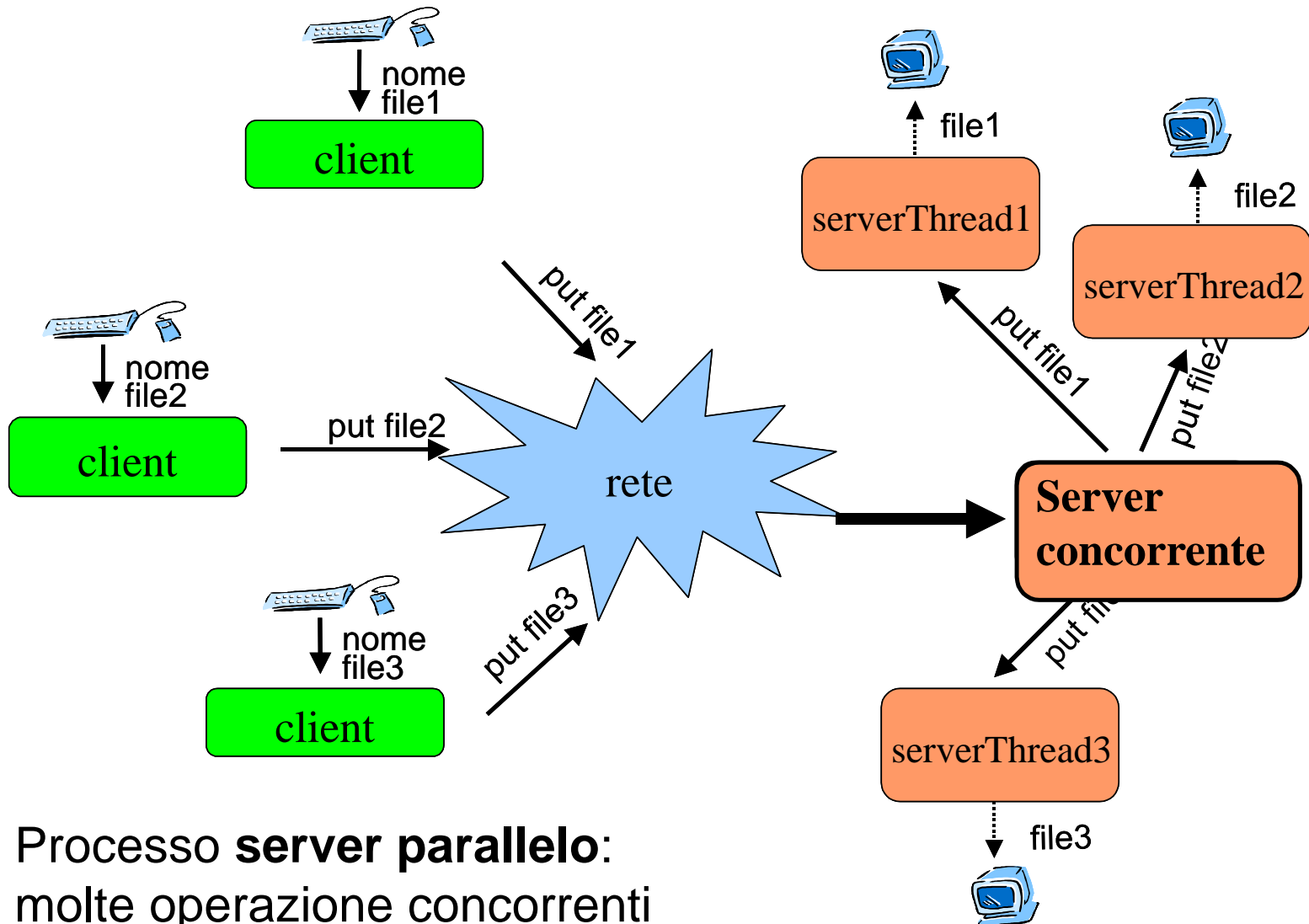
Socket connesse

- **Creazione** della connessione
- **Uso della connessione** per la **comunicazione** e il **dialogo**:
invio del nome del file e contenuto del file
- **Distruzione** della connessione
- Si ripetono le operazioni sopra in ciclo fino a EOF

ARCHITETTURA DI SUPPORTO A UN TRASFERIMENTO FILE: SERVER SEQUENZIALE



ARCHITETTURA DI SUPPORTO A UN TRASFERIMENTO FILE: SERVER CONCORRENTE



Processo **server parallelo**:
molte operazione concorrenti

SPECIFICA

Sviluppare un'applicazione C/S che effettui il **trasferimento di un file binario** dal client al server (**put**)

Il file binario non è un file testo

Il **Client** chiede all'utente il *nome del file* da trasferire, si connette al server (con `java.net.Socket`), crea uno **stream di output** sulla connessione attraverso cui inviare **il file selezionato, preceduto dal suo nome**. Fatto ciò, il client attende l'esito dell'operazione, e ricevuto l'esito torna a proporre *una nuova richiesta di trasferimento all'utente fino a consumare tutto l'input*

Il **Server** attende una richiesta di connessione da parte del client (su `java.net.ServerSocket`), usa la socket prodotta dalla richiesta di connessione (`java.net.Socket`) per creare uno **stream di input** da cui riceve il *nome del file* e successivamente il *contenuto del file* che **salverà nel file system locale** nella directory in cui è stato lanciato. Il server invia poi l'esito dell'operazione e chiude la connessione

Vi sono due possibili casi (esiti), quello di **sovra-scrittura** del file e quello di **creazione** di nuovo file, ognuno dei quali può terminare con successo o meno

FILTRO

Un filtro è un **programma** che **consuma tutto il suo input** e **porta l'uscita sull'output**



Possiamo pensare di combinarne in una **pipeline**, oppure di utilizzare la **ridirezione** dello standard input/output

Un filtro potrebbe ad esempio **leggere fino alla fine del file uno stream di input**, trasferendo i dati letti sullo stream di output , come vedremo più avanti

Diverse tipologie di filtri: a **caratteri**, a **linee**, a **byte**, ...

Nel seguito vediamo un semplice **filtro a linee**: `FiltroSemplice` e tra poco vedremo un altro esempio di **filtro a byte**:

`trasferisci_a_byte_file_binario`

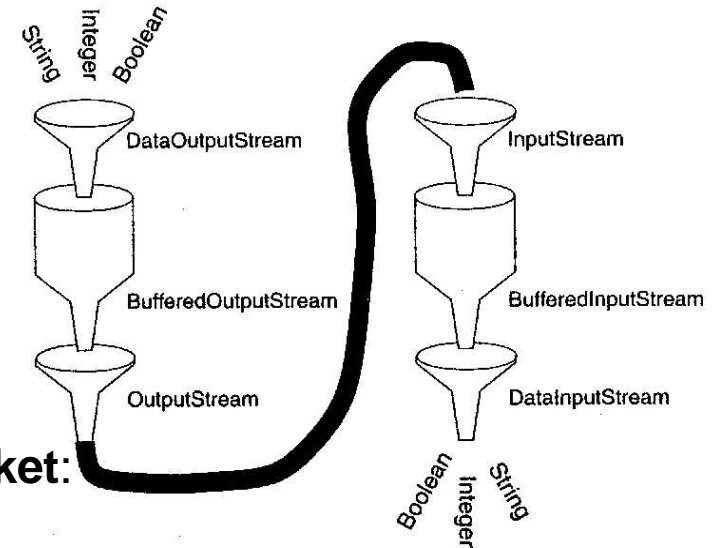
UN SEMPLICE FILTRO A LINEE

Il seguente filtro **riceve linee da standard input**, e **riporta sullo standard output solo le linee con il carattere 'a'**

```
public class FiltroSemplice {
    public static void main(String[] args) {
        String line;
        BufferedReader input =
            new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter output =
            new BufferedWriter(new OutputStreamWriter(System.out));
        System.err.println("\nMsg per utente:");
        try {
            while ((line = input.readLine()) != null)
                if (line.lastIndexOf('a') >= 0) output.write(line +
"\n");
            output.flush();    // svuotiamo il buffer
        }
        catch (IOException e) {
            System.out.println("Problemi: ");
            e.printStackTrace();
        }
    }
}
```

FILTRAGGI E STREAM

Stream di input/output come filtri...
innestati uno dentro l'altro



Esempi di creazione stream di input/output da **socket**:

```
DataInputStream inSock =  
    new DataInputStream(socket.getInputStream()) ;  
DataOutputStream outSock =  
    new DataOutputStream(socket.getOutputStream()) ;
```

Esempi di creazione stream di input/output da **file binario**:

```
DataInputStream inFile =  
    new DataInputStream(new FileInputStream(nomeFile)) ;  
DataOutputStream outFile =  
    new DataOutputStream(new FileOutputStream(nomeFile)) ;
```


FILEUTILITY PER TRASFERIMENTO FILE BINARIO

// Metodo statico: **trasferisci_a_byte_file_binario**

```
static protected void
    trasferisci_a_byte_file_binario
    (DataInputStream src, DataOutputStream dest)
        throws IOException
{ // ciclo di lettura da sorgente e scrittura su destinazione
    int buffer = 0;
    try
    { // esco dal ciclo alla lettura di un valore negativo -> EOF
        while ( (buffer = src.read()) >= 0)
            dest.write(buffer) ;

        dest.flush() ;
    }
    catch (EOFException e)
    {   System.out.println("Problemi: ");e.printStackTrace();
    }
}
```

SCHEMA DI SOLUZIONE: IL CLIENT

1. Creazione socket con **bind implicita** e set delle opzioni:

```
socket = new Socket(addr, port);  
socket.setxxx(...);
```

2. Interazione da console con l'utente:

```
BufferedReader stdIn = new BufferedReader(new  
    InputStreamReader(System.in));  
System.out.print("Dammi un nome di file... ");  
String nomeFile = null;  
while( nomeFile=stdIn.readLine() )!=null)
```

3. Creazione dello stream di output sulla socket:

```
outSock =  
    new DataOutputStream(socket.getOutputStream());
```

SCHEMA DI SOLUZIONE: IL CLIENT (ANCORA)

4. Creazione dello stream di input da file binario:

```
inFile =  
    new DataInputStream(new FileInputStream(nomeFile));
```

5. Invio dei dati al server:

```
outSock.writeUTF(nomeFile);  
FileUtility.trasferisci_a_byte_file_binario  
    (inFile, outSock);
```

6. Chiusura del file e della socket (in modo dolce) e lettura esito:

```
inFile.close();  
socket.shutdownOutput();  
...  
esito = inSock.readUTF();  
socket.shutdownInput();
```

SCHEMA DI SOLUZIONE: IL SERVER

1. Creazione socket con **bind implicita** e settaggio opzioni:

```
serverSocket = new ServerSocket(port) ;  
serverSocket.setReuseAddress(true) ;
```

2. Attesa/accettazione di richiesta di connessione:

```
clientSocket = serverSocket.accept() ;
```

3. Creazione dello stream di input sulla socket:

```
inSock = new DataInputStream(clientSocket.getInputStream()) ;
```

4. Creazione dello stream di output su file binario:

```
nomeFile=inSock.readUTF() ;  
outFile = new DataOutputStream(  
    new FileOutputStream(nomeFile) ) ;
```

SCHEMA DI SOLUZIONE: IL SERVER (ANCORA)

5. Ricezione dei dati dal client e invio dei dati sulla console in uscita:

```
FileUtility.trasferisci_a_byte_file_binario  
    (inSock,outFile) ;
```

6. Chiusura del file e della socket (in modo dolce) e invio esito:

```
outFile.close() ;  
socket.shutdownInput() ;  
...  
outSock.writeUTF(esito) ;  
socket.shutdownOutput() ;
```



Ovviamente, si devono sempre fare close() di tutte le socket e i file non più necessari

PUTFILECLIENT PER FILE BINARIO 1/3

```
public class PutFileClient {

    public static void main(String[] args) throws IOException {
        InetAddress addr = null;
        int port = -1;
        try{ // controllo argomenti
            if(args.length == 2)
            { addr = InetAddress.getByName(args[0]);
              port = Integer.parseInt(args[1]);
            } else{ System.out.println("Usage: ..."); System.exit(1); }
        } //try
        catch(Exception e){ ... }

        // oggetti per comunicazione e lettura file
        Socket socket = null; String esito;
        FileInputStream inFile = null; String nomeFile = null;
        DataInputStream inSock = null; DataOutputStream outSock = null;
        BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
        System.out.print("\n^D(Unix)/^Z(Win)+invio .... Nome file?");
```

PUTFILECLIENT PER FILE BINARIO 2/3

```
try{
    while ((nomeFile=stdIn.readLine())!=null){
        if(new File(nomeFile).exists()){
            try{ // creazione socket
                socket = new Socket(addr, port);
                socket.setSoTimeout(30000);
                inSock = new DataInputStream(socket.getInputStream());
                outSock = new DataOutputStream(socket.getOutputStream());
            } catch(Exception e){... continue;}
        }
        else{System.out.println("File non presente");
            System.out.print("\n^D(Unix)/^Z(Win)..."); continue;
        }
        // Invio file
        try{ inFile = new FileInputStream(nomeFile); }
        catch(FileNotFoundException e){...}
```

PUTFILECLIENT PER FILE BINARIO 3/3

```
try
{
    outSock.writeUTF(nomeFile);
    FileUtility.trasferisci_a_byte_file_binario(
        new DataInputStream(inFile), outSock);
    inFile.close(); // chiusura della socket e del file
    socket.shutdownOutput(); // chiudo in upstream, cioe' invio EOF
}
catch (SocketTimeoutException te) {... continue;}
catch (Exception e){... continue;}
try{ // ricezione esito
    esito = inSock.readUTF();
    socket.shutdownInput(); // chiudo la socket in downstream
}
catch (SocketTimeoutException te) {... continue;}
catch (Exception e){... continue;}
System.out.print("\n^D(Unix)/^Z(Win)..."); // nuova richiesta
} // while
} // try
catch (Exception e){... System.exit(3);}
} // main
} // class
```


PUTFILESERVERSEQ PER FILE BINARIO 1/3

```
public class PutFileServerSeq {
    public static final int PORT = 54321; // porta di default

    public static void main(String[] args) throws IOException
    {int port = -1; String nomeFile; FileOutputStream outFile = null; String esito;
        try // controllo argomenti
        { if (args.length == 1) {
            port = Integer.parseInt(args[0]);
        } else if (args.length == 0) {
            port = PORT;
        } else { // Msg errore...  }
        } //try
        catch (Exception e) {...}
        ServerSocket serverSocket = null; // preparazione socket e in/out stream
        try
        { serverSocket = new ServerSocket(port) ;
            serverSocket.setReuseAddress(true) ;
        }
        catch (Exception e) {...}
        try
        { while (true) // ciclo infinito del server
            { Socket clientSocket = null;
                DataInputStream inSock = null;  DataOutputStream outSock = null;
```

PUTFILESERVERSEQ PER FILE BINARIO 2/3

```
try
{
    clientSocket = serverSocket.accept();
    clientSocket.setSoTimeout(30000);
}
catch (Exception e) {... continue;}
try // creazione stream di I/O
{
    inSock =new DataInputStream(clientSocket.getInputStream());
    outSock =new DataOutputStream(clientSocket.getOutputStream());
    nomeFile = inSock.readUTF();
}
catch (SocketTimeoutException te) {... continue;}
catch (IOException e) {... continue;}
// ricezione file su file nuovo
if (nomeFile == null) { clientSocket.close(); continue;}
else {
    File curFile = new File(nomeFile);
    if (curFile.exists()) {
        try
        {
            esito = "File sovrascritto";
            curFile.delete(); // distruggo il file
        } catch (Exception e) {... continue;}
    }
}
```

PUTFILESERVERSEQ PER FILE BINARIO 3/3

```
} else esito = "Creato nuovo file";
    outFile = new FileOutputStream(nomeFile);
}
try // ricezione file
{ FileUtility.      // N.B. la funzione consuma l'EOF
    trasferisci_a_byte_file_binario(
        inSock, new DataOutputStream(outFile));
    outFile.close(); // chiusura file
    clientSocket.shutdownInput();
    outSock.writeUTF(esito+", file salvato su server");
    clientSocket.shutdownOutput();
}
    catch (SocketTimeoutException te) {... continue;}
    catch (Exception e) {...continue;}
}
}
catch (Exception e) {... System.exit(3);}
}
}
```

PUTFILESERVERCON PER FILE BINARIO 1/4

```
class PutFileServerThread extends Thread {
private Socket clientSocket = null;
public PutFileServerThread(Socket clientSocket)
    { this.clientSocket = clientSocket; }

public void run() // Processo figlio per trattare la connessione
{ DataInputStream inSock;
  DataOutputStream outSock;
  try
  { String nomeFile;
    try // creazione stream
    { inSock = new DataInputStream(clientSocket.getInputStream());
      outSock = new DataOutputStream(clientSocket.getOutputStream());
      nomeFile = inSock.readUTF();
    }
    catch (SocketTimeoutException te) {...}
    catch (IOException ioe) {...} catch (Exception e) {...}
    FileOutputStream outFile = null; String esito;
    // ricezione file: caso di errore
    if (nomeFile == null) {clientSocket.close(); return; }
```

PUTFILESERVERCON PER FILE BINARIO 2/4

```
else { // controllo esistenza file
    File curFile = new File(nomeFile);
    if (curFile.exists()) {
        try // distruggo il vecchio file
        { esito = "File sovrascritto"; curFile.delete(); }
        catch (Exception e) {... return;}
    } else esito = "Creato nuovo file";
    outFile = new FileOutputStream(nomeFile);
}
try {
    FileUtility.trasferisci_a_byte_file_binario
        (inSock, new DataOutputStream(outFile));
    outFile.close(); // chiusura file e socket
    // NOTA: è il figlio che fa la close!
    clientSocket.shutdownInput();
    outSock.writeUTF(esito + ", file salvato lato server");
    clientSocket.shutdownOutput();
}
catch (SocketTimeoutException te) {...}
catch (Exception e) {...}
} catch (Exception e) {... System.exit(3);}
} // run
} // PutFileServerThread
```

PUTFILESERVERCON PER FILE BINARIO 3/4

```
public class PutFileServerCon {
    public static final int PORT = 1050;

    public static void main (String[] args)
        throws IOException {
        int port = -1;
        try // controllo argomenti
        { if (args.length == 1) {port = Integer.parseInt(args[0]); }
          else if (args.length == 0) {port = PORT; }
          else { System.out.println("Usage: ..."); System.exit(1); }
        } //try
        catch (Exception e) {... System.exit(1);}

        ServerSocket serverSocket = null; Socket clientSocket = null;
        try
        { serverSocket = new ServerSocket(port);
          serverSocket.setReuseAddress(true);
        }
        catch (Exception e) {... System.exit(1);}

        try
        { // CICLO PRINCIPALE
```

PUTFILESERVERCON PER FILE BINARIO 4/4

```
while (true)
{try {  clientSocket = serverSocket.accept();
        clientSocket.setSoTimeout(30000);
    } catch (Exception e) {... continue;}
try { // servizio delegato ad un nuovo thread
    new PutFileServerThread(clientSocket).start();

/* NOTA!!! La close della socket di connessione viene fatta dal FIGLIO,
* il PADRE NON DEVE fare la close,
* altrimenti si hanno interferenze perché c'è memoria condivisa
*/

    } catch (Exception e) {... continue;}
} // while
} // try
catch (Exception e) {... System.exit(2);}
} // main
} // PutFileServerCon
```