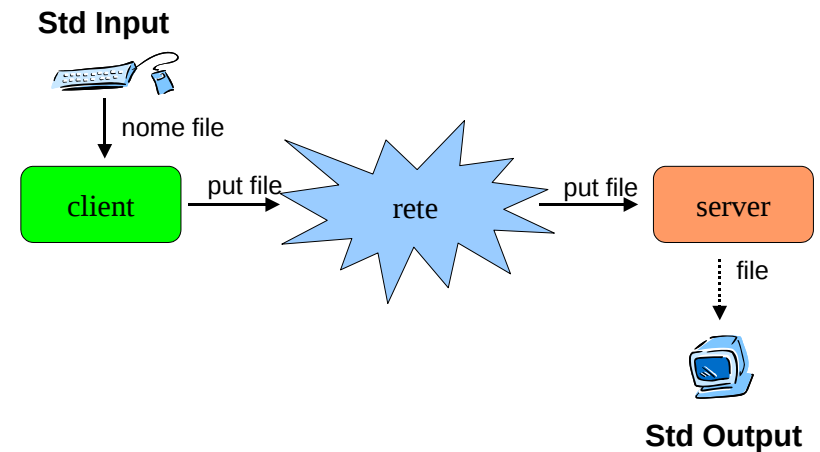


3° Esercitazione (svolta): Socket Java con connessione

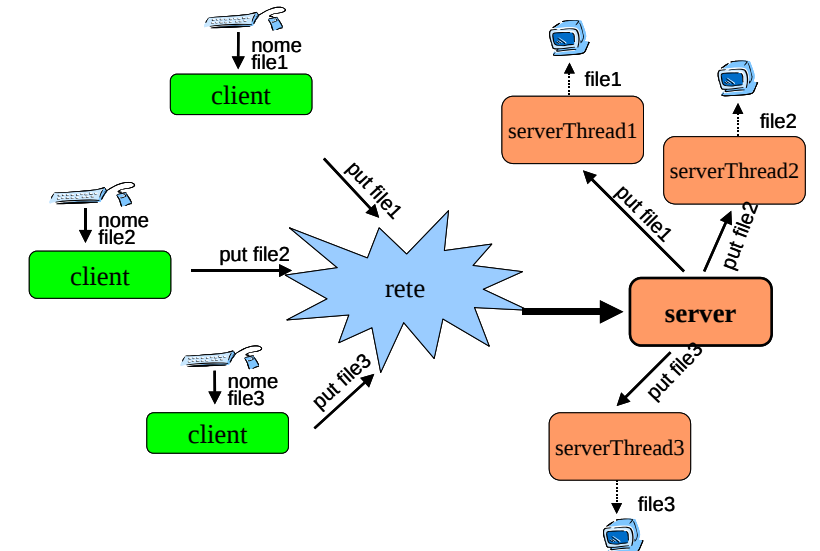
Sviluppare un'applicazione C/S che effettui il **trasferimento di un file binario** dal client al server (**put**). In particolare nel servizio:

- 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.
- il server attende una richiesta di connessione da parte del client (su **java.net.ServerSocket**), usa la socket (**java.net.Socket**) prodotta dalla connessione per creare uno **stream di input** da cui riceve il nome del file e successivamente il file che **salverà** nel file system locale nella directory nella quale viene lanciato. Fatto ciò il server invia 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.

Server Sequenziale



Server Concorrente



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**

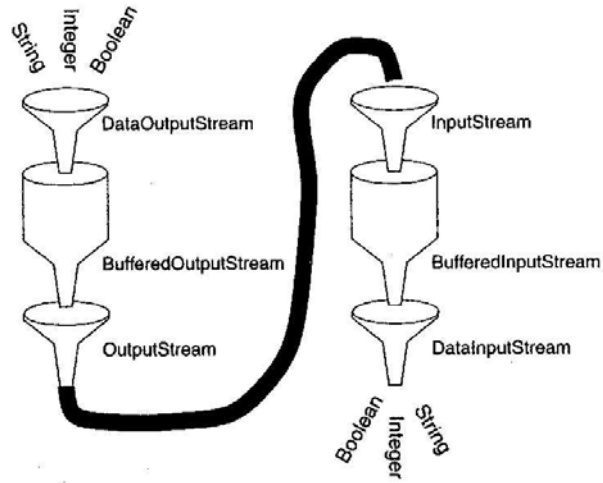
Un semplice filtro a linee

Il seguente filtro riceve linee da standard input, e riporta sullo standard output solamente le linee che contengono 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");
                    // e se fosse fuori dal ciclo?
                    output.flush();
                }
            }
        } catch (IOException e) {
            System.out.println("Problemi: ");
            e.printStackTrace();
        }
    }
}
```

Filtraggi e Stream



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

```
DataOutputStream outSock = new
    DataOutputStream(new FileOutputStream(nomeFile));
```

FileUtility:

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 per file binario: il Client

1. Creazione socket (e la **bind?**) e settaggio 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());
```

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 per file binario: 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));
```

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();
```

PutFileClient per file binario

```
public class PutFileClient {

    public static void main(String[] args) throws
        IOException {
        // Argomenti: host e porta server
        InetAddress addr = null;
        int port = -1;
        /* controllo argomenti */
        try{
            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){
            ...
            System.exit(2);
        }
        // oggetti per comunicazione e lettura file
        Socket socket = null;
        FileInputStream inFile = null;
        DataInputStream inSock = null;
        DataOutputStream outSock = null;
        BufferedReader stdIn = new BufferedReader(
            new InputStreamReader(System.in));
        String nomeFile = null;
        System.out.print("\n^D(Unix)/^Z(Win)+invio "+
            "per uscire, altrimenti nome file: ");
    }
```

```
try{
    while ((nomeFile=stdIn.readLine())!=null){
        if(new File(nomeFile).exists()){
            // creazione socket
            try{
                socket = new Socket(addr, port);
                // setto il timeout per non bloccare
                //indefinitivamente il client
                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 richiesto */
        try{
            inFile = new FileInputStream(nomeFile);
        }
        catch(FileNotFoundException e){...}
    }
```

```

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

```

PutFileServerSeq per file binario

```

public class PutFileServerSeq {
    // porta di default
    public static final int PORT = 1050;

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

```

```

try {
    clientSocket = serverSocket.accept();
    // per evitare il blocco indefinito
    clientSocket.setSoTimeout(30000);
}
catch (Exception e) {... continue;}
String nomeFile;
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 */
FileOutputStream outFile = null;
String esito;
if (nomeFile == null) {
    clientSocket.close();
    continue;
} else {
    File curFile = new File(nomeFile);
    if (curFile.exists()) {
        try {
            esito = "File sovrascritto";
            // distruggo il file
            curFile.delete();
        }
        catch (Exception e) {... continue;}
    }
}

```

```

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

```

PutFileServerCon per file binario

```
class PutFileServerThread extends Thread {

private Socket clientSocket = null;

public PutFileServerThread(Socket clientSocket)
{ this.clientSocket = clientSocket; }

public void run() {
    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) {...}
        /* ricezione file */
        FileOutputStream outFile = null;
        String esito;
        if (nomeFile == null) {
            clientSocket.close();
            return;
        }
    }
}
```

```
    } else {
        // controllo esistenza file
        File curFile = new File(nomeFile);
        if (curFile.exists()) {
            try {
                esito = "File sovrascritto";
                // distruggo il file da sovrascrivere
                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));
        // chiusura file e socket
        outFile.close();
        /* NOTA: è il figlio che fa la close! */
        clientSocket.shutdownInput();
        outSock.writeUTF(esito +
            ", file salvato lato server");
        outSock.flush();
        clientSocket.shutdownOutput();
    }
    catch (SocketTimeoutException te) {...}
    catch (Exception e) {...}
}
catch (Exception e) {... System.exit(3);}
}
```

```

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

    public static void main(String[] args) throws
    IOException {
        int port = -1;

        /* controllo argomenti */
        try {
            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 {
            while (true) {
                try {
                    // bloccante
                    clientSocket = serverSocket.accept();
                    clientSocket.setSoTimeout(30000);
                }
                catch (Exception e) {... continue;}
            }
        }
    }
}

```

```

        // servizio delegato ad un nuovo thread
        try {
            new PutFileServerThread(clientSocket)
                .start();
        }
        catch (Exception e) {... continue;}
    }
    catch (Exception e) {... System.exit(2);}
}

```

/* NOTA!!! La close del client socket viene
* fatta dal FIGLIO, il PADRE NON DEVE fare la
* close, altrimenti si hanno interferenze
* perchè c'è memoria condivisa
*/