

# ESERCITAZIONE 2

Socket in Java con connessione

# FileUtility

```
public class FileUtility {  
    static protected void trasferisci_a_byte_file_binario(DataInputStream src, DataOutputStream dest, long length) throws IOException {  
        int numByteRead = 0;  
        int buffer = 0;  
  
        try {  
            while (numByteRead < length) {  
                buffer = src.read(); // Leggo un byte dalla DataInputStream  
                dest.write(buffer); // Scrivo quel byte sul DataOutputStream  
                numByteRead++; // Incremento il contatore del numero di byte letti  
            }  
            dest.flush();  
        } catch (EOFException e) {  
            System.out.println("Riscontrati i seguenti problemi: ");  
            e.printStackTrace();  
        }  
    }  
}
```

Il metodo creato nella classe FileUtility legge da DataInputStream un byte alla volta e lo scrive sul DataOutputStream fino al raggiungimento della lunghezza del file (passata come parametro).

# Client (1)

```
try {  
    while ((dirname = stdin.readLine()) != null) {  
        System.out.print("Client Started.\n\n^D(Unix)/^Z(Win)+invio per uscire.\n");  
        System.out.print("Inserire la dimensione minima dei file per la directory " + dirname + ": ");  
  
        threshold = Long.parseUnsignedLong(stdin.readLine()); // Lancia eccezioni in caso di errori  
        directory = new File(dirname);  
        if (!directory.isDirectory()) {  
            System.out.println("Directory non presente: " + dirname);  
            System.out.print("\n^D(Unix)/^Z(Win)+invio per uscire, oppure reinserisci una directory: ");  
            continue;  
        }  
    }  
}
```

Si legge ciclicamente il nome di una directory e la dimensione minima dei file, in modo da poter trasmettere soltanto quelli di dimensione maggiore. Si controlla anche che la directory esista.

# Client (2)

```
String response = null;
```

```
try {
    for (File f : directory.listFiles()) {
        if (f.length() < threshold) {
            continue;
        }

        inFile = new FileInputStream(f);
        outSock.writeUTF(f.getName()); // Invio il nome del file

        response = inSock.readUTF();
        if (response.equals("salta")) {
            System.out.println("Il file " + f.getName() + " esiste già nel server");
            continue;
        } else if (response.equals("attiva")) {
            System.out.println("Invio lunghezza file: " + f.length());
            outSock.writeLong(f.length());
            System.out.println("Inizio la trasmissione di " + f.getName());

            try {
                FileUtility.trasferisci_a_byte_file_binario(new DataInputStream(inFile), outSock, f.length());
                inFile.close();
                System.out.println("Trasmissione di " + f + " terminata ");
            }
        }
    }
}
```

Per ogni file della directory si controlla che la sua dimensione sia maggiore della soglia (variabile *threshold*). I nomi dei file che rispettano il requisito vengono inviati al server, gli altri vengono scartati. Se il server richiede il file ("attiva"), si invia la relativa lunghezza e il file.

# Server

```
try {
    serverSocket = new ServerSocket(port);
    serverSocket.setReuseAddress(true);
    System.out.println("Server: avviato ");
    System.out.println("Server: creata la server socket: " + serverSocket);
} catch (Exception e) {...}
```

Si crea la socket con cui il server riceve le richieste.

```
try {
    while (true) {
        System.out.println("Server: in attesa di richieste...\n");

        try {
            clientSocket = serverSocket.accept();
            clientSocket.setSoTimeout(30000);
            System.out.println("Server: connessione accettata: " + clientSocket);
        } catch (Exception e) {
            System.err.println("Server: problemi nella accettazione della connessione: " + e.getMessage());
            e.printStackTrace();
            continue;
        }

        // Servizio delegato ad un nuovo thread
        try {
            new ServerThread(clientSocket).start();
        } catch (Exception e) {...}
    }
}
```

Il server rimane attivo in attesa di richieste. Ad ogni richiesta crea (e avvia) un ServerThread che si occupa della ricezione dell'intera directory da un client.

# ServerThread

```
while (true) {
    fileName = inSock.readUTF();
    FileOutputStream outFile = null;
    String result = null;
    File curFile = new File(fileName);

    if (curFile.exists()) {
        result = "salta";
        outSock.writeUTF(result);
    } else {
        result = "attiva";
        outFile = new FileOutputStream(fileName);
        long fileLength = 0;

        outSock.writeUTF(result);
        outSock.flush();
        fileLength = inSock.readLong();

        System.out.println("Ricevo il file " + fileName);
        FileUtility.trasferisci_a_byte_file_binario(inSock, new DataOutputStream(outFile), fileLength);
        System.out.println("Ricezione del file " + fileName + " e copia nel server terminata\n");
        outFile.close();
    }
}
```

Il ServerThread legge il nome di un file dal DataInputStream e controlla se esiste o meno nella directory del Server. Se esiste, comunica al Client che non è necessario inviarlo ("salta"), altrimenti richiede l'invio ("attiva"). Nel secondo caso, ne legge la lunghezza e poi salva in memoria il file.