ESERCITAZIONE 1

Socket in Java senza connessione

RSClient.java > DiscoveryServer.java

```
try{
    socket = new DatagramSocket();
    socket.setSoTimeout(30000);
    packet = new DatagramPacket(buf, buf.length, addr, port);
    System.out.println("\nRSClient: avviato");
    System.out.println("Creata la socket: " + socket);

} catch (SocketException e){
    System.out.println("Problemi nella creazione della socket: ");
    e.printStackTrace();
    System.out.println("RSClient: interrompo...");
    System.exit(1);
}
```

Creazione di un clausula Try-Catch per contenere l'inizializzazione della Socket Datagram e del Packet Datagram e per cattuare l'eventuale eccezzione SocketException.

RSClient.java > RowSwapServer.java

```java
while((insert = stdIn.readLine()) != null || row1 == -1) {

    boStream.reset();
    row1 = -1;
    row2 = -1;

    row1 = Integer.parseInt(insert);

    System.out.println("Inserire seconda riga con cui invertire");

    while(row2 == -1 && (insert = stdIn.readLine()) != null) {
        row2 = Integer.parseInt(insert);
    }

    boStream.reset();
    doStream.writeUTF(row1 + " " + row2);
    data = boStream.toByteArray();
    packet.setData(data, 0 , data.length);
    socket.send(packet);
    System.out.println("Richiesta inviata correttamente al RowSwap");

    packet.setData(buf);
    socket.receive(packet);

    biStream = new ByteArrayInputStream(packet.getData(), 0 , packet.getLength());
    diStream = new DataInputStream(biStream);
    risposta = diStream.readUTF();

    res = Integer.valueOf(risposta);

    if(res != -1){
        System.out
        .println("Operazione svolta con successo");
        System.out
        .print("\n^D(Unix)/^Z(Win)+invio per uscire, altrimenti inserire prima riga da invertire: ");
    }

    else {
        System.out.println("RowSwap: operazione non contemplata");
        System.exit(9);
    }
}
```

Doppio ciclo per la selezione delle righe da scambiare nel file.
Il primo inizializza la richiesta e richiede nello stdin la prima riga; il secondo si occupa semplicemente di richiedere la riga.
Controllo finale con variabile “res” per la comunicazione dell’esito.

```java
try {
    row1 = Integer.parseInt(insert);
} catch(NumberFormatException e) {

    System.out.println("Problema interazione da console: ");
    e.printStackTrace();
    System.out
    .print("\n^D(Unix)/^Z(Win)+invio per uscire, altrimenti inserire numero riga 1: ");
    continue;
}
```

La clausula catch non termina l’esecuzione, ma fornisce una possibiltà di ripetere la lettura del numero linea.

DiscoveryServer.java

```java
// controllo argomenti input: numero dispari di argomenti, almeno 3 (serverPort file1 port1)
if (args.length%2 == 1 && args.length >= 3) {

    //salvataggio porta per socket server
    try {
        porta = Integer.parseInt(args[0]);
        // controllo che la porta sia nel range consentito 1024-65535
        if (porta < 1024 || porta > 65535) {
            System.out.println("Usage: java DiscoveryServer [serverPort>1024] file1 [port1>1024] file2 [port2>1024]...");
            System.exit(1);
        }
    } catch (NumberFormatException e) {
        System.out.println("Usage: java DiscoveryServer [serverPort>1024] file1 [port1>1024] file2 [port2>1024]...");
        System.exit(1);
    }
```

Controllo numero arogomenti (min 3) e controllo che la porta del DiscoveryServer sia nel range [1024-65535].

```java
ports = new int[args.length/2];
files = new String[args.length/2];
```

Strutture dati che contengono filename e porte.

DiscoveryServer.java

```java
if (port < 1024 || port > 65535 || !f.isFile()) {
    System.out.println("Usage: ...");
} else {
    for(int z = 0; z < nServers; z++){
        if(port == ports[z]) {
            doppia = true;
            System.out.println("porta doppia: " +port);
        }
    }

    if(!doppia){
        files[j] = args[i-1];
        ports[j] = port;
        nServers++;
        j++;
    }   doppia = false;
}
```

Controllo validità delle porte ed esistenza del file.
Controllo che le porte non siano duplicate ed inserimento di fileName e delle porte nelle strutture dati.

```java
int risposta = -1;
boolean trovato = false;

for(int i = 0; !trovato && i < nServers; i++) {
    if (files[i].equals(richiesta)) {
        trovato = true;
        risposta = ports[i];
    }
}
```

Elaborazione della richiesta dell'utente e risposta con esito positive o negative (se il file non esiste).

RowSwapServer.java & LineUtility.java

```
line1 = LineUtility.getLine(fileName, numLine1);
line2 = LineUtility.getLine(fileName, numLine2);
BufferedReader br = new BufferedReader(new FileReader(fileName));
String line = null;
int numLine = 1;
PrintWriter pw = new PrintWriter(fileName + ".tmp", "UTF-8");

while ((line = br.readLine()) != null ) {
    if (numLine == numLine1) {
        pw.println(line2);
    } else if (numLine == numLine2) {
        pw.println(line1);
    } else {
        pw.println(line);
    }
    numLine++;
}
br.close();
pw.close();
```

LineUtility legge la riga scelta in maniera statica.
Tramite un BufferedReader leggiamo il file linea per linea.
In base ad un indice, controlliamo il numero di linea e tramite le apposite clausole IF, invertiamo le righe, scrivendo su un file creato con estensione .tmp.

```
for (int i = 1; i <= numLinea; i++) {
    linea = in.readLine();
    if (linea == null) {
        in.close();
        throw new IOException("Linea non trovata");
    }
}
```

Abbiamo modificato LineUtility rimuovendo getNextLine in quanto non utilizzato e al posto di restituire una stringa “Linea non trovata”, lanciamo un’eccezione che notifica l’errore al RowSwapServer che blocca lo scambio righe, se la riga non esiste.