Esame del 13 Luglio 2006 (IA)

Lorenzo Gallucci

Esame del 13 Luglio 2006 (IA) --- testo

- Si realizzi un'applicazione client-server basata sui socket che implementi il seguente schema di comunicazione:
- Il server riceve dal client un intero *n* e poi un double k, quindi una sequenza di *n* double. A questo punto il server calcola il numero di double della sequenza che sono maggiori del numero k e restituisce al client il risultato (*num*).

Esame del 13 Luglio 2006 (IA) --- testo

- Il client legge da tastiera l'intero *n*, il double *k* e la sequenza di *n* double, li invia al server ed attende di ricevere il risultato dal server stesso. Una volta ricevuto il numero *num* dal server gli risponde con una sequenza di *num* zeri che confermano la ricezione e poi chiude il socket. La stessa sequenza viene stampata dal server sul proprio standard output. Sarà gradita un'implementazione che preveda la possibilità di avere più client (ovviamente utilizzando i thread).
- Esempio: se *n*=**5,** *k*=**18.57** e la sequenza [**9.67 77.53 19.18 57.63 2.33]** il server restituirà 3 poiché i valori maggiori di 18.57 sono 3 (**77.53 19.18 57.63)**. Il client gli invierà quindi 3 zeri.

Modellazione - responsabilità delle classi

- Il server deve:
  - Invocare ciclicamente accept() su un ServerSocket
  - Per ogni socket generato, creare un Thread che se ne prenda cura
- Il thread che gestisce, lato server, i client, deve:
  - Leggere i dati dal socket
  - Calcolare il risultato richiesto (# elementi, degli n, che sono > k)
  - Inviare il risultato
  - Rileggere la conferma inviata dal client
  - Chiudere il socket

Modellazione - responsabilità delle classi

- Il client deve:
  - Leggere i dati da input
  - Aprire un socket per comunicare con il server
  - Inviare i dati
  - Leggere il risultato
  - Inviare una conferma sul socket
  - Chiudere il socket

- … come leggere i dati da input?

Lettura di valori da tastiera: la classe Scanner

- java.util.Scanner…
  - Si basa su InputStream
  - Consente di leggere interi, float, stringhe, …
    - Per leggere interi: int nextInt()
    - Per leggere double: double nextDouble()
    - Ecc.
  - È necessario uno spazio od un “invio” tra un dato e il successivo
- Esempio di utilizzo:

```
public static void main(String[] args)
 {
    Scanner scanner = new Scanner(System.in);
    int intero = scanner.nextInt();
    double virgolaMobile = scanner.nextDouble();
    System.out.println(intero + " - " + virgolaMobile);
 }
```

Soluzione – classe Client

```
package sistemidielaborazioneinrete.e13Luglio2006IA;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class Client extends Thread
{
   private int            n;

   private double         k;

   private double[]       valori;

   private int            maggioriDiK;
```

Soluzione – classe Client

```
private final int        porta;

private final String     host;

private InputStream      inputStream;

private OutputStream     outputStream;

private DataOutputStream dataOutputStream;

private DataInputStream  dataInputStream;

private Socket           socket;
```

Soluzione – classe Client

```
 public void leggiValori()
{
    Scanner scanner = new Scanner(System.in);
    System.out.print("Dammi n (intero): ");
    n = scanner.nextInt();
    System.out.print("Dammi k (double): ");
    k = scanner.nextDouble();
    valori = new double[n];
    // Leggi gli n double da System.in
    for (int i = 0; i < n; i++)
    {
        System.out.print("Dammi il numero in posizione " + (i + 1) + "/" + n
 + ": ");
        valori[i] = scanner.nextDouble();
    }
}
```

Soluzione – classe Client

```
public Client(String host, int porta)
    {
        this.host = host;
        this.porta = porta;
    }

    @Override
    public void run()
    {
        try
        {
            interagisciConIlServer();
        }
        catch (IOException e)
        {
            throw new RuntimeException("Non riesco a gestire il socket lato client", e);
        }
    }
```

Soluzione – classe Client

```
private void interagisciConIlServer() throws IOException
{
    preparaSocket();
    try
    {
        inviaRichiesta();

        leggiRisultato();
        inviaConferma();
    }
    finally
    {
        socket.close();
    }
}
```

Soluzione – classe Client

```
private void preparaSocket() throws
    UnknownHostException, IOException
    {
        socket = new Socket(host, porta);
        inputStream = socket.getInputStream();
        outputStream = socket.getOutputStream();
        dataOutputStream = new
    DataOutputStream(outputStream);
        dataInputStream = new
    DataInputStream(inputStream);
    }
```

Soluzione – classe Client

```
private void inviaRichiesta() throws IOException
    {
        dataOutputStream.writeInt(n);
        dataOutputStream.writeDouble(k);
        // Scrivi gli interi
        for (int i = 0; i < n; i++)
        {
            dataOutputStream.writeDouble(valori[i]);
        }
    }
```

Soluzione – classe Client

```
private void leggiRisultato() throws IOException
{
    // Leggi il risultato
    maggioriDiK = dataInputStream.readInt();
    System.out.println("Risultato ottenuto: " +
maggioriDiK + " numeri su " + n + " sono maggiori di
" + k);
}
```

Soluzione – classe Client

```
private void inviaConferma() throws
IOException
{
    // Scrivi degli zeri di conferma
    for (int i = 0; i < maggioriDiK; i++)
    {
        dataOutputStream.writeDouble(0.0d);
    }
}
```

Soluzione – classe Server

```
package sistemidielaborazioneinrete.e13Luglio2006IA;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Server extends Thread
{
    private final int    porta;

    private ServerSocket serverSocket;

    public Server(int porta) throws IOException
    {
        this.porta = porta;
        serverSocket = new ServerSocket(porta);
        setDaemon(true);
    }
```

Soluzione – classe Server

```
@Override
public void run()
{
    while (true)
    {
        try
        {
            Socket socket = serverSocket.accept();
            // Crea ed avvia un nuovo thread per
            // prendersi cura del socket appena creato
            CalcolatoreThread calcolatoreThread = new CalcolatoreThread(socket);
            calcolatoreThread.start();
        }
        catch (IOException e)
        {
            System.err.println("Non riesco ad accettare la connessione");
            // Stampa l'errore
            e.printStackTrace();
            // Prosegui con la prossima connessione
        }
    }
}
}
```

Soluzione – classe CalcolatoreThread

```
package sistemidielaborazioneinrete.e13Luglio2006IA;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class CalcolatoreThread extends Thread
{
   private final Socket     socket;

   private InputStream      inputStream;

   private OutputStream     outputStream;

   private DataInputStream  dataInputStream;

   private DataOutputStream dataOutputStream;

   private int              n;

   private double           k;

   private int              maggioriDiK;
```

Soluzione – classe CalcolatoreThread

```
public CalcolatoreThread(Socket socket)
{
    this.socket = socket;
}

@Override
public void run()
{
    try
    {
        gestisciSocket();
    }
    catch (IOException e)
    {
        throw new RuntimeException("Non riesco a gestire il socket lato server", e);
    }
}
```

Soluzione – classe CalcolatoreThread

```
private void gestisciSocket() throws IOException
{
    try
    {
        preparaStream();
        leggiECalcola();
        inviaRisultato();
        rileggiConferma();
    }
    finally
    {
        socket.close();
    }
}
```

Soluzione – classe CalcolatoreThread

```
private void preparaStream() throws
IOException
 {
    inputStream = socket.getInputStream();
    outputStream = socket.getOutputStream();
    dataInputStream = new
DataInputStream(inputStream);
    dataOutputStream = new
DataOutputStream(outputStream);
 }
```

Soluzione – classe CalcolatoreThread

```
private void leggiECalcola() throws IOException
{
    n = dataInputStream.readInt();
    System.out.println("Server: ricevuto n = " + n);
    k = dataInputStream.readDouble();
    System.out.println("Server: ricevuto k = " + k);
    maggioriDiK = 0;
    // Leggi gli n double da socket
    for (int i = 0; i < n; i++)
    {
        double numero = dataInputStream.readDouble();
        System.out.println("Server: ricevuto numero in posizione " + (i + 1) + ",
  valore = " + numero);
        if (numero > k) maggioriDiK++;
    }
    System.out.println("Server: " + maggioriDiK + " valori su " + n + " sono
  maggiori di " + k);
}
```

Soluzione – classe CalcolatoreThread

```
private void inviaRisultato() throws
IOException
{
    // Scrivi il risultato
    dataOutputStream.writeInt(maggioriDiK);
}
```

Soluzione – classe CalcolatoreThread

```
private void rileggiConferma() throws
IOException
{
    // Rileggi la conferma
    for (int i = 0; i < maggioriDiK; i++)
    {
        double valore =
dataInputStream.readDouble();
        if (valore != 0.0) throw new
RuntimeException("Valore inatteso " + valore + "
nella conferma");
    }
}
```

Soluzione – classe Main

```
package sistemidielaborazioneinrete.e13Luglio2006IA;

import java.io.IOException;

public class Main
{
    public static void main(String[] args) throws IOException
    {
        int porta = 2000;
        Server server = new Server(porta);
        server.start();
        Client client = new Client("localhost", porta);
        client.leggiValori();
        client.start();
    }
}
```

Esempio di output

Dammi n (intero): 8
Dammi k (double): 2,3
Dammi il numero in posizione 1/8: 1
Dammi il numero in posizione 2/8: 2
Dammi il numero in posizione 3/8: 3
Dammi il numero in posizione 4/8: 4
Dammi il numero in posizione 5/8: 5
Dammi il numero in posizione 6/8: 2,2
Dammi il numero in posizione 7/8: 2,4
Dammi il numero in posizione 8/8: 2,5
Server: ricevuto n = 8
Server: ricevuto k = 2.3
Server: ricevuto numero in posizione 1, valore = 1.0
Server: ricevuto numero in posizione 2, valore = 2.0
Server: ricevuto numero in posizione 3, valore = 3.0
Server: ricevuto numero in posizione 4, valore = 4.0
Server: ricevuto numero in posizione 5, valore = 5.0
Server: ricevuto numero in posizione 6, valore = 2.2
Server: ricevuto numero in posizione 7, valore = 2.4
Server: ricevuto numero in posizione 8, valore = 2.5
Server: 5 valori su 8 sono maggiori di 2.3
Risultato ottenuto: 5 numeri su 8 sono maggiori di 2.3