Università degli Studi di Pisa
Dipartimento di Informatica

Lezione n.5
LPR-Informatica Applicata
Invio di Oggetti su UDP

13/3/2006

Laura Ricci

LA CLASSE BYTEARRAYOUTPUTSTREAM

- Definisce una struttura dati

  protected byte buf [ ];

  protected int count

  buf memorizza i bytes che vengono scaricati sullo stream

  count indica quanti sono i bytes memorizzati in buf

- Costruttori

  ByteArrayOutputStream( )

  crea un buf di 32 bytes (ampiezza di default)

  ByteArrayOutputStream (*int* size)

  crea un buf di dimensione size

LA CLASSE BYTEARRAYOUTPUTSTREAM

- Ogni volta che un byte viene scritto sull'oggetto *ByteArrayOutputStream()*, il byte viene automaticamente memorizzato nel buffer

- Se il buffer risulta pieno, la sua lunghezza viene automaticamente raddoppiata

- Il risultato è che si ha l'impressione di scrivere su uno stream di lunghezza non limitata

- Metodi per la scrittura sullo stream
  - *public synchronized void* write (*int* b)
  - *public synchronized void* write (*byte* b [ ], *int* off, *int* len)

LA CLASSE BYTEARRAYOUTPUTSTREAM

- Metodi per la gestione dello stream
  - *public int* size( ) restituisce count, cioè il numero di bytes memorizzati nello stream (NON la lunghezza del vettore buf!)
  - *public synchronized void* reset( ) assegna 0 a count. In questo modo lo stream risulta vuoto e tutti i dati precedentemente scritti vengono eliminati.
  - *public synchronized byte* toByteArray ( ) restituisce un vettore in cui sono stati copiati tutti i bytes presenti nello stream. NON modifica count, per cui lo stream NON viene resettato

LA CLASSE OBJECTOUTPUTSTREAM

- Il codice sorgente della libreria è molto complesso

- Costruttore

  public ObjectOutputStream (*OutputStream* out) *throws Exception*

  scrive sullo stream un header necessario per ricostruire gli oggetti

  header = costituito da due short, 4 bytes (costanti STREAM_MAGIC, STREAM_VERSION)

  se l'header viene cancellato lo stream risulta corrotto.

LA CLASSE OBJECTOUTPUTSTREAM

```
....
public class Test
    { public static void main(String Args[ ]) throws Exception
          {
            ByteArrayOutputStream bout = new ByteArrayOutputStream ( );
            System.out.println (bout.size( ));
            // Stampa 0
            ObjectOutputStream out= new ObjectOutputStream(bout);
            System.out.println (bout.size( ));
           // Stampa 4
```

...(continua pagina successiva)

LA CLASSE OBJECTOUTPUTSTREAM

```
out.writeObject("prova");
System.out.println (bout.size( ));
// Stampa 12
bout.reset ( );
out.writeObject("prove");
System.out.println (bout.size( ));
// Stampa 8= 12-4.
```

….(continua pagina successiva)

IMPORTANTE!:

- la reset ha distrutto l'header dello stream.
- Se si tenta di ricostruire gli oggetti dallo stream, verrà segnalata un'eccezione di tipo *StreamCorruptedException*

LA CLASSE OBJECTOUTPUTSTREAM

bout.*reset*( );

out = *new ObjectOutputStream* (bout);

out.*writeObject* ("prova");

System.out.println (b.out.size( ));

// Stampa 12. La creazione di un nuovo stream ha ricreato l'header dello Stream

.....(continua pagina successiva)

LA CLASSE OBJECTOUTPUTSTREAM

```
out = new ObjectOutputStream (bout);
out.writeObject("prova");
System.out.println (bout.size( ));
bout.reset();
out.writeObject("prova");
System.out.println (bout.size( ));
 // Dovrebbe stampare 8, perchè ha tolto l'header, ma ha ricreato la stringa prova. Invece stampa 5.
```

ATTENZIONE: La classe si ricorda se un oggetto è già stato inserito nello stream ed in quel caso, non lo riscrive, ma inserisce un "riferimento" al precedente.

Questo può provocare problemi se si scrive più volte lo stesso oggetto sullo stream, modificandone lo stato

LA CLASSE OBJECTOUTPUTSTREAM

```
out = new ObjectOutputStream (bout);
out.writeObject("prova");
System.out.println (bout.size( ));
// STAMPA 12
bout.reset( )
out.reset( )
out.writeObject("prova")
System.out.println (bout.size( ));
// STAMPA 8
```

- L'istruzione di *reset* ( ) sull'ObjectOutputStream annulla lo stato dello Stream. Se invio di nuovo l'oggetto "prova" lo stream non ha più memoria, non "ricorda" di averlo inviato in precedenza

UDP: SERIALIZZAZIONE DI OGGETTI

Esercizio:

Il Server *ServerScuola* mantiene informazioni circa le assenze giustificate e quelle non giustificate effettuate dagli alunni e le invia ad un *ClientGenitore,* tramite un collegamento UDP.

- definizione di un oggetto messaggio serializzabile contenente due interi (assenze giustificate, assenze non giustificate)

- spedizione/ricezione oggetti mediante *ObjectInput/OutputStream*

Nota: La versione presentata è notevolmente semplificata per mettere in evidenza i problemi principali. Sviluppare la versione completa.

SERIALIZZAZIONE DEGLI OGGETTI

```
import java.io.*;
class messaggio implements Serializable
    { private int nassenzeg;
      private int nassenzeng;
      public messaggio(int na, int ng)
          {this.nassenze    = na;
          this.nassenzeng  = ng; }
      public int getx ( ) {return nassenzeg;};
      public int gety ( ) {return nassenzeng;}
      public void setx(int na)    {nassenzeg    = na};
      public void sety(int nng)   {nassenzeng = nng;} }
```

IL SERVER ASSENZE

```
import java.net.*;
import java.io.*;
import java.util.*;
    public class ServerAssenze
    { public static void main (String Args[ ]) throws Exception
     { InetAddress ia = InetAddress.getByName("LocalHost");
       int port= 1300;
       DatagramSocket ds = new DatagramSocket( );
       ByteArrayOutputStream bout=new ByteArrayOutputStream();
       byte [ ] data=new byte[256] ;
       DatagramPacket dp= new DatagramPacket(data, data.length, ia, port);
```

IL SERVER ASSENZE

```
for (int i=1;i<10;i++)
    {     int na=i; int nr=i;
          messaggio m=new messaggio(na,nr);
          ObjectOutputStream dout = new ObjectOutputStream(bout);
          dout.writeObject(m);
          dout.flush ( );
          data =bout.toByteArray();
          dp.setData(data);
          dp.setLength(data.length);
          ds.send (dp);
          bout.reset ( );
   } }
```

IL SERVER ASSENZE

- E' necessario costruire un nuovo *ObjectOutputStream* per ogni oggetto inviato. Questo permette di rigenerare l'header.

- E' necessario inserire *bout.reset( )* all'interno del ciclo, in modo da eliminare dallo stream i bytes relativi ad oggetti già spediti

- posso eliminare la *bout.reset( )* se sposto l'istruzione *ByteArrayOutputStream* bout=*new ByteArrayOutputStream*( ); all'interno del ciclo for.

- provare a spostare l'istruzione *ObjectOutputStream* dout = *new ObjectOutputStream*(bout) fuori dal ciclo *for:* il destinatario non riesce a ricostruire l'oggetto serializzato (*StreamCorruptedException*).

IL CLIENT ASSENZE

```
import java.net.*;
import java.io.*;
import java.util.*;

public class ClientAssenze{

public static void main (String Args[]) throws Exception
        { InetAddress ia = InetAddress.getByName("LocalHost");
          int port=1300;
          DatagramSocket ds=new DatagramSocket(port);
          byte buffer[]=new byte[256];
          DatagramPacket dpin= new DatagramPacket(buffer, buffer.length);
```

IL CLIENT ASSENZE

```
for (int i=1;i<10;i++)
{ds.receive(dpin);
     ByteArrayInputStream bais= new
    ByteArrayInputStream(dpin.getData ( ));
     ObjectInputStream ois= new ObjectInputStream (bais);
      messaggio m = (messaggio) ois.readObject();
     System.out.println(m.getx());
     System.out.println(m.gety());
    }
```

Provare a vedere cosa accade se si elimina dal server l' istruzione bout.*reset* ( ) dal server!!

INVIO DI PIU' OGGETTI IN UN UNICO PACCHETTO: Il SERVER

```
import java.net.*;
import java.io.*;
public class objserver
{   public static void main(String Args[]) throws Exception
    {Messaggio m;
    InetAddress ia = InetAddress.getLocalHost( );
    int port = 7555;
    DatagramSocket ds=new DatagramSocket();
    ByteArrayOutputStream bout=new  ByteArrayOutputStream( );
    ObjectOutputStream oo= new ObjectOutputStream(bout);
```

INVIO DI PIU' OGGETTI IN UN UNICO PACCHETTO: IL SERVER

```
m = new Messaggio (1,1);
oo.writeObject(m);
m = new Messaggio (2,2);
oo.writeObject(m);
byte [ ] data=bout.toByteArray();
DatagramPacket dp= new DatagramPacket(data,data.length, ia, port);
ds.send(dp);    } }
```

INVIO DI PIU' OGGETTI IN UN UNICO PACCHETTO: IL CLIENT

```
import java.io.*;
import java.net.*;
public class objclient{
public static void main (String Args[]) throws Exception
    {Messaggio m;
    byte [] data = new byte[516];
    DatagramSocket ds = new DatagramSocket(7555);
    DatagramPacket dp =new DatagramPacket(data, data.length);
    ds.receive(dp);
    ByteArrayInputStream bin= new ByteArrayInputStream(data);
    ObjectInputStream oin= new ObjectInputStream(bin);
```

INVIO DI PIU' OGGETTI IN UN UNICO PACCHETTO: IL CLIENT

```
m=(Messaggio) oin.readObject();
System.out.println(m.getx(), m.gety( ));
m=(Messaggio) oin.readObject();
System.out.println(m.getx(), m.gety( ));
        }
 }
```

Il programma stampa correttamente i valori dei due oggetti ricevuti:

(1,1) (2,2)

INVIO DI PIU' OGGETTI IN UN UNICO PACCHETTO: IL SERVER

Modifichiamo ora il server come segue:

……m = new Messaggio (1,1);

oo.*writeObject* (m);

m.setx(2); m.sety(3);

oo.*writeObject*(m);

byte [] data=bout.*toByteArray*();

DatagramPacket dp= *new DatagramPacket*(data,data.length, ia, port);
ds.*send*(dp); …….

- Il server stampa (1,1), (1,1), perché l'oggetto riferito è lo stesso nelle due *writeObject*( )
- Il server può resettare l'*ObjectOutputStream* (oo.reset( )) per annullare lo stato dell'oggetto