Corso sul linguaggio Java

Modulo JAVA6

A2 – I file binari

Prerequisiti

- Programmazione base in Java
- Utilizzo di classi e oggetti
- Modello produttore consumatore
- Operazioni logiche su struttura file

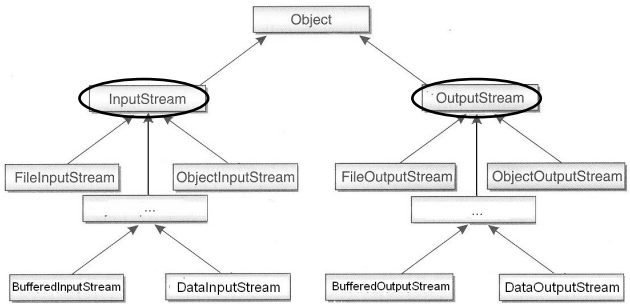
Introduzione

La necessità di memorizzare grandi quantità di dati e di elaborarli in modo veloce ed efficiente, viene soddisfatta in Java mediante i **file binari.**

Vediamo le classi necessarie e i relativi metodi, per poter realizzare programmi di gestione di file binari.

File binari

Le classi per trattare file binari sono **InputStream** (per gli *stream* di input) e **OutputStream** (per gli *stream* di output) che sono **classi astratte**, per cui, normalmente usiamo le loro sottoclassi indicate in figura.



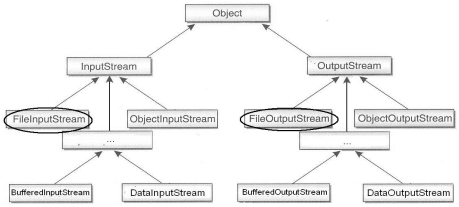
File binari

In Java, gli *stream* binari possono essere:

- *stream* binari propriamente detti, formati da sequenze di byte (**FileInputStream** e **FileOutputStream**)
- *stream* binari di tipi primitivi (**DataInputStream**, **DataOutputStream**)
- *stream* binari di oggetti (**ObjectInputStream**, **ObjectOutputStream**)

Stream binari

- **FileInputStream** per gli *stream* di input
- **FileOutputStream** per gli *stream* di output



Metodi:

- **int read()** legge un byte dallo *stream* (-1 se lo *stream* è finito)
- **void write** (**int** b) scrive l'intero b nello *stream* (convertito in **byte**)
- **void flush**() svuota il buffer di output
- **void close()** chiude lo *stream*

Stream binari

ATTIVITA': scrivere un'applicazione che consenta di copiare il file *copiaFile.class* in *nuovoFile.class*.

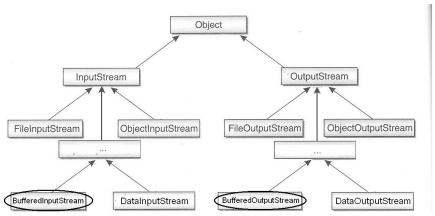
```
import java.io.* ;
public class copiaFile
{  public static void main(String[ ] args)
   {try  {FileInputStream fin = new FileInputStream("copiaFile.class");
          FileOutputStream fout= new FileOutputStream("NuovoFile.class");
           int byteLetto; int nCicli=0;
           while ((byteLetto=fin.read())!=-1)
            {    fout.write(byteLetto); nCicli++;  }
           fin.close( );    fout.close( );
           System.out.println("nCicli = " + nCicli);
   }
   catch (IOException e)
   {  System.out.println ("Errore sul file");  }
   }
}  // fine classe
```

Aggiungendo **true** sullo *stream* di output lo si apre in accodamento

Stream binari bufferizzati

Anche con i file binari è consentita la **bufferizzazione** mediante le classi:

- **BufferedInputStream** per *stream bufferizzati* di input
- **BufferedOutputStream** per *stream bufferizzati* di output



Metodi:

- **int read(byte** b[ ]) legge un array dallo *stream* (-1 se lo *stream* è finito)
- **void write (byte** b[ ]) scrive un array nello *stream*
- **void flush**() svuota il buffer di output
- **void close()** chiude lo *stream*

Stream binari bufferizzati

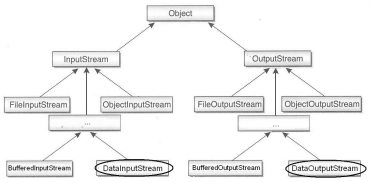
```
import java.io.* ;
public class copiaBufferizzata
{ public static void main(String[ ] args)
  {try { dichiarazione stream di input e output
        BufferedInputStream bufin = new BufferedInputStream (fin);
        BufferedOutputStream bufout = new BufferedOutputStream (fout);
        byte array[] = new byte[50];
        int nCicli=0;
        while ((bufin.read(array))!=-1)
        { bufout.write(array);   bufout.flush();    nCicli++;   }
        bufin.close( );   bufout.close( );
        System.out.println("nCicli = " + nCicli);
       }
       catch (IOException e)
       { System.out.println("Errore sul file");  }
  }
}  // fine classe
```

Buffer di lettura scrittura

ATTIVITA': confrontare il valore di *nCicli* eseguendo *CopiaBufferizzata* e *CopiaFile* per la copiatura di uno stesso file.

Stream binari tipi primitivi

- **DataInputStream** per gli *stream* di input
- **DataOutputStream** per gli *stream* di output



UTF è una particolare codifica dei caratteri, che estende il codice **ASCII**

Metodi:

- **boolean readBoolean**()
- **char readChar()**
- **double readDouble()**
- **float readFloat()**
- **int readInt()**
- **String readUTF()**
- **void flush()**
- **void writeBoolean** (**boolean** v)
- **void writeChar** (**int** v)
- **void writeDouble** (**double** v)
- **void writeFloat** (**float** v)
- **void writeInt** (**int** v)
- **void writeUTF** (**String** s)
- **void close** ()

Stream binari tipi primitivi

```
import java.io.* ;
public class tipiIntegrali
{  public static void main(String[ ] args)
   {
      try
      {
         String filename = "numeri.dat";
         int i;   double d;  char c;   final int N=10;
         FileOutputStream fout= new FileOutputStream (filename);
         DataOutputStream dataout = new DataOutputStream (fout);
         for (i=1; i <= N; i++)
         {  d = Math.random();  dataout.writeInt(i);
            dataout.writeDouble(d);   dataout.writeChar(64+i);
         }
         dataout.close( );
```

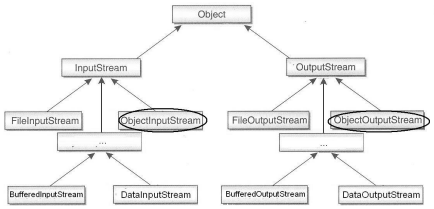

Stream binari tipi primitivi

```
         FileInputStream fin = new FileInputStream (filename);
         DataInputStream datain = new DataInputStream (fin);
         try
         {   while (true)
             {
                 i = datain.readInt();
                 d = datain.readDouble();
                 c = datain.readChar();
                 System.out.println(i + "\t" + d + "\t" + c);
             }
         }
         catch (EOFException E)
         {   datain.close( );   }
         }
         catch (IOException E)
         {   System.out.println("Errore sul file");   }
   }     // fine main
}  // fine classe
```

Utilizza fine file come eccezione

Stream binari di oggetti

- **ObjectInputStream** per gli *stream* di input
- **ObjectOutputStream** per gli *stream* di output



Metodi:

- **Object readObject**()
- **void writeObject (Object obj)**
- **void close** ()
- **void flush** ()

Stream binari di oggetti

La gestione di oggetti su disco richiede alcune osservazioni:

- il salvataggio di oggetti su disco prende il nome di **serializzazione** (implementa l'interfaccia **Serializable**) poiché ogni oggetto viene dotato di un numero seriale univoco.
- se un oggetto è registrato più di una volta, viene registrato solo il suo numero seriale
- i file contenenti oggetti serializzati NON consentono accesso diretto

Stream binari di oggetti

Si consideri la classe seguente:

```
import java.io.*;
class Libro implements Serializable
{
   String   autore,
            editore;
   Double  prezzo;
   Libro (String a, String e, Double p)
   {
         autore=a;
         editore=e;
         prezzo=p;
   }
}
```

Vediamo come oggetti di questo tipo possono essere registrati su file.

Stream binari di oggetti

```
import java.io.* ;
public class FileDiOggetti
{  public FileDiOggetti()
   {String filename = "libri.dat";
    Libro l1 = new Libro("Manzoni", "Paravia", 3.50);
    Libro l2 = new Libro("Alighieri","Einaudi", 4.25);
    try
    {      FileOutputStream fout= new FileOutputStream (filename);
           ObjectOutputStream out = new ObjectOutputStream (fout);
           out.writeObject(l1);   out.writeObject(l2);
           out.flush();     out.close();
           FileInputStream fin = new FileInputStream (filename);
           while (true)
           { try   {   Libro l = (Libro) fin.readObject();
                       System.out.println(l.autore + " " + l.editore + " " + l.prezzo);
                   }
             catch(EOFException e)     {   break;            }
           }
           in.close();
     }  …segue classe
```

Stream binari di oggetti

```
    catch (ClassNotFoundException e)
    { e.printStackTrace();      }
    catch (IOException E)
    { System.out.println("Errore in lettura");  }
    }
    public static void main(String args[])
    {    new FileDiOggetti();    }
}   // fine classe
```


Argomenti

- File binari
- *Stream* binari
- *Stream* binari bufferizzati
- *Stream* binari tipi primitivi
- *Stream* binari di oggetti

Altre fonti di informazione

- P.Gallo, F.Salerno – Java, ed. Minerva Italica
- M. Bigatti – Il linguaggio Java, ed. Hoepli