I/O Streams in Java

Lorenzo Gallucci

I/O Streams

- La libreria Java dispone di un'API sofisticata per l'interfacciamento I/O
 - Il package di riferimento è `java.io`
 - Due classi principali
 - `InputStream`, per l'input di dati
 - `OutputStream`, per l'output di dati
 - Molteplici implementazioni, a seconda del tipo di sorgente o di destinazione
-

I/O Streams

- Metodi disponibili in `InputStream`:
 - ❑ `int read()` → legge un singolo byte, restituendo -1 se si è arrivati alla fine dello stream
 - ❑ `int read(byte[] buf)` → riempie l'array specificato, restituendo il numero di byte effettivamente letti
 - ❑ `int read(byte[] buf, int offset, int len)` → riempie la porzione da `offset` ad `offset + len` dell'array specificato, restituendo i byte effettivamente letti
 - ❑ `void close()` → chiude lo stream
 - TUTTI i metodi possono lanciare una `IOException`!!
-

I/O Streams

- Metodi duali in OutputStream:
 - `void write(int b)` → scrive un singolo byte (solo gli 8 bit inferiori sono significativi)
 - `void write (byte[] buf)` → scrive tutti i byte l'array specificato
 - `void write (byte[] buf, int offset, int len)` → scrive, dell'array specificato, unicamente i byte dalla posizione offset alla posizione offset + len
 - `void close()` → chiude lo stream
 - TUTTI i metodi possono lanciare una IOException!!
-

I/O Streams

- Due stream sono molto importanti:
 - System.in
 - System.out
 - Entrambi sono agganciati alla console utente, ma in direzioni diverse
 - System.out serve a visualizzare messaggi
 - System.in ad accettare degli input
 - System.out è, in effetti, un tipo particolare di OutputStream, PrintStream, che offre migliori possibilità di output e metodi più semplici
-

I/O Streams

- Come comunicare in maniera più strutturata?
 - Vi sono due tipi di stream:
 - DataInputStream
 - DataOutputStream
 - ... che offrono capacità di I/O “migliori”; in particolare:
 - Con un Data*Stream è possibile leggere/scrivere oggetti String, interi, booleani, ecc.
 - La logica delle due classi si occupa di tradurre opportunamente in byte i dati scambiati
-

DataInputStream (1/2)

- DataInputStream offre i metodi:
 - ❑ byte readByte() → legge il prossimo byte
 - ❑ int readUnsignedByte() → legge il prossimo byte, visto senza segno (inglobato in un int)
 - ❑ short readShort() → legge il prossimo short int (16 bit)
 - ❑ int readUnsignedShort() → legge il prossimo short int, visto senza segno (inglobato in un int)
 - ❑ boolean readBoolean() → legge il prossimo booleano
-

DataInputStream (2/2)

- DataInputStream offre i metodi:
 - ❑ `int readInt()` → legge il prossimo intero (32 bit)
 - ❑ `long readLong()` → legge il prossimo intero lungo (64 bit)
 - ❑ `float readFloat()` → legge il prossimo float (32 bit)
 - ❑ `double readDouble()` → legge il prossimo double (64 bit)
 - ❑ `char readChar()` → legge il prossimo char
 - ❑ `String readUTF()` → legge la prossima stringa, in formato UTF-8 [combina i metodi precedenti]
-

DataOutputStream (1/2)

- **DataOutputStream** offre i metodi:
 - ❑ `void writeByte(int i)` → scrive un byte (8 bit)
 - ❑ `void writeShort(int i)` → scrive uno short int (16 bit)
 - ❑ `void writeBoolean(boolean b)` → scrive un booleano
 - ❑ `void writeInt(int i)` → scrive un intero (32 bit)
 - ❑ `void writeLong(long l)` → scrive un intero lungo (64 bit)
-

DataInputStream (2/2)

- DataInputStream offre i metodi:
 - ❑ void writeFloat(float f) → scrive un float (32 bit)
 - ❑ void writeDouble(double d) → scrive un double (64 bit)
 - ❑ void writeChar(char c) → scrive un char
 - ❑ void writeUTF(String s) → scrive una stringa, in formato UTF-8 (compatibile con readUTF())
-

ByteArray*Stream

- Esiste una categoria particolare di stream di I/O: quelli che scrivono e leggo *in memoria* (senza coinvolgere reali risorse di I/O, come i file)
 - In Java sono rappresentati da due classi:
 - *ByteArrayOutputStream*, che accumula i dati ricevuti, write dopo write, e li mette a disposizione con un metodo *toByteArray()*
 - *ByteArrayInputStream*, che consente di leggere i dati contenuti in un *byte[]* come se provenissero da uno stream
-

Esempio con Data*Stream

- Combinando gli oggetti visti finora, è possibile:
 - ❑ Preparare uno stream (1) che accumula i dati in memoria
 - ❑ Scrivere su di esso una stringa, per mezzo di un secondo stream (2)
 - ❑ Recuperare dallo stream (1) i dati accumulati e “rimetterli in circolo” per mezzo di un nuovo stream (3)
 - ❑ Recuperare da (3), per mezzo di uno stream (4), la stringa scritta
-
- ❑ Mostrare il risultato in output

Esempio con Data*Stream

■ Codice:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream(); /* 1 */
DataOutputStream dataOutputStream = new DataOutputStream(baos); /* 2 */
dataOutputStream.writeUTF("prova");

/* 3*/
ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());
DataInputStream dataInputStream = new DataInputStream(bais); /* 4 */
String prova = dataInputStream.readUTF();

System.out.println(prova);
```