Corso sul linguaggio Java

Modulo JAVA8

B2 – Accesso diretto

Prerequisiti

- Programmazione ad oggetti
- Conoscenza classi di base di I/O
- Tecnica della programmazione
- Organizzazione ed accesso ai file

Introduzione

In questa Unità introduciamo i **file random** di Java, una classe particolare di *stream* che consente **l'accesso diretto**, e mostriamo una semplice applicazione di questa tecnica.

Successivamente, facendo sempre uso delle due classi *Libro* e *Libreria* create nell'Unità precedente a proposito dell'accesso sequenziale, modifichiamo l'applicazione *Libreria* per consentire **l'accesso diretto** ai dati sul file (che è sempre organizzato sequenzialmente)

Utilizzo di RandomAccessFile

La classe Java **per i file random è RandomAccessFile** che consente:

- apertura degli *stream* in lettura, scrittura e lettura/scrittura:
  - **RandomAccessFile** fout **=new RandomAccessFile**(filename, "rw");
- accesso diretto al record in base alla posizione *pos*:
  - fout.**seek**((**long**) (pos – 1) * 100);

Applichiamo questi nuovi strumenti per scrivere alcune procedure di gestione di uno stream di interi (per semplicità).

Utilizzo di RandomAccessFile
- inserimento

```
Importazione packages
public class Inserimento
{   String filename;
    RandomAccessFile raf;
    public Inserimento (String fname)   {  filename=fname;   }
    public static void main(String args[])
    {  int c;          long pos;
       JFileChooser dlgFile = new JFileChooser();   // per selezione file
       try { Inserimento i = new Inserimento ("valori.dat");
             seleziona file nella finestra di dialogo e aprilo;
             while (valore letto da input !=0)
             {   leggere posizione, spostare l'indicatore, scrivere valore c} }
             i.raf.close();
          catch (IOException e)  {System.out.println("Errore!");        }
    }
}
```


Utilizzo di RandomAccessFile
- inserimento

```
seleziona file nella finestra di dialogo e aprilo
if (dlgFile.showOpenDialog (null) = = JFileChooser.APPROVE_OPTION)
{   File file = dlgFile.getSelectedFile();
    i.raf = new RandomAccessFile(file, "rw");}

valore letto da input !=0
(c = Integer.parseInt(JOptionPane.showInputDialog("Valore: "))!=0)

leggere posizione, spostare l'indicatore, scrivere valore c
pos = Integer.parseInt(JOptionPane.showInputDialog("Posizione: "));
i.raf.seek((long)(pos-1)*4);
i.raf.writeInt(c);
```

Utilizzo di RandomAccessFile
- stampa

```
Inportazione packages
public class Stampa
{   String filename;
    RandomAccessFile raf;
    public Stampa(String fname)  {  filename=fname;    }
    public static void main (String args[]) throws IOException
    {       int l;
     Stampa s= new Stampa("valori.dat");
     try
     {      s.raf=new RandomAccessFile(s.filename, "r");
            while ((l=s.raf.readInt())! = -1)    System.out.println(l);
     }
     catch (EOFException e)
     {      s.raf.close();   System.out.println("Fine file");   }
    }
}
```


Utilizzo di RandomAccessFile
- ricerca

```
Inportazione packages
public class Ricerca
{   String filename;
    RandomAccessFile raf;
    public Ricerca(String fname)  {  filename=fname;  }
    public static void main(String args[])
    {   try  {  Ricerca r = new Ricerca("valori.dat");
                r.raf = new RandomAccessFile(r.filename, "r");
                int c;
                long pos = Integer.parseInt(Joption…("Posizione: "));
                r.raf.seek((long)(pos-1)*4);
                c = r.raf.readInt();
                System.out.println("Trovato: " + c);
                r.raf.close();
             }
        catch (IOException e)  {  System.out.println("Fine file!");  }
    }
}
```

Specifiche dell'applicazione
- ipotesi di lavoro

Facciamo sull'applicazione *Libreria.java* le seguenti ipotesi:

- il file di dati *filename* si suppone gestito ad accesso diretto ed ordinato in base ad un codice numerico progressivo;
- la funzione di eliminazione si effettua con la cancellazione logica (ad esempio, ponendo il codice a 0) che si può effettuare senza la copia su file ausiliario (il metodo *copia*( ) va eliminato)
- l'applicazione prevede un menu operativo, in cui l'utente sceglie con una lettera l'operazione desiderata;
- i dati nel file vengono gestiti come dati primitivi;

Specifiche dell'applicazione
- cancellazione logica

La cancellazione logica richiede che vengano modificate le procedure di inserimento, stampa e visualizzazione come segue:

- la stampa elencherà i soli record che non hanno zero nel campo codice
- la visualizzazione di un record cancellato logicamente, darà il messaggio di record inesistente;
- l'inserimento di un codice avviene se non esiste già un codice uguale oppure se esiste, ma è stato cancellato logicamente.

Specifiche dell'applicazione
- classi impiegate

Nell'applicazione *Libreria.java* per l'I/O sul file utilizziamo le classi:

- **RandomAccessFile** che utilizza l'archivio sequenziale come **archivio random**, col vantaggio di consentire:
  - apertura degli *stream* in lettura, scrittura e lettura/scrittura.
  - accesso diretto al record in base alla posizione;
- **StringBuffer** che viene di volta in volta caricata con i dati di un record e riempita con spazi, per far risultare tutti i record della stessa lunghezza.

Package java.util.*;

- **StringTokenizer** che consente di estrarre da una stringa di classe **StringBuffer** i vari elementi per ricostruire i vari campi.

La classe **RandomAccessFile**

METODO	EFFETTO
RandomAccessFile *ident* =**new RandomAccessfile** (**String** *file*, **String** *mode*)	Crea uno *stream* di nome *ident*, associato al file esterno *file*. *mode*="r", "rw", "w",...
void close()	Chiude lo *stream* ad accesso random
long getFilePointer()	Restituisce il file pointer nel file corrente
long length()	Restituisce la lunghezza del file
int read()	Legge un byte di dati dal file
int read(byte[] b)	Legge un array di byte dal file
boolean readBoolean()	Legge un valore booleano dal file
char readChar()	Legge un carattere UNICODE dal file
double readDouble()	Legge un valore doppio dal file
float readFloat()	Legge un valore reale dal file
int readInt()	Legge un intero di 32 bit dal file
String readLine()	Legge una stringa dal file fino a fine linea
String readUTF()	Legge una stringa dal file
void seek(**long** pos)	Posiziona il *file pointer* a *pos* byte rispetto all'inizio
int write(byte[] b)	Scrive un array di byte sul file
int write(**int** b)	Scrive un byte sul file
int writeBoolean(**boolean** v)	Scrive un valore booleano sul file
int writeBytes(**String** s)	Scrive una stringa *s* sul file
void writeChar(**int** v)	Scrive un carattere di 2 byte sul file
void writeInt (**int** v)	Scrive un intero di 32 bit sul file
void writeDouble(**double** v)	Scrive un valore doppio sul file
void writeFloat(**float** v)	Scrive un valore reale sul file
void writeUTF(**String** s)	Scrive una stringa sul file

Inserimento

```
public void inserimento (int cod)
{   Libro l=new Libro();
    try
    {   RandomAccessFile fout =new RandomAccessFile(filename, "rw");
            aut = JOption…..;
            lettura altri attributi;
            Libro l = new Libro (cod, aut, edit, tit, pr);
            scriviRec (fout, l);
            fout.close();
        }
    }
    catch (IOException e)
    {       System.out.println("Errore in inserimento!");   }
}
```

Apertura in lettura e scrittura

Stampa

```
public void stampa()
{
    Libro l;
    try
    {
            RandomAccessFile fin=new RandomAccessFile(filename, "r");
            while ((l = leggiRec(fin))!=null)
                    if (l.getCodice()!=0)
                        l.stampaLibro();
            fin.close();
    }
    catch (IOException e)
    {
            System.out.println("Errore in lettura");
    }
}
```

Apertura in lettura

Scrittura e lettura

Per implementare la ricerca è necessario modificare l'implementazione dei metodi *scriviRec*() e *leggiRec*(), visti nel caso dell'accesso sequenziale, per adattarli al file *random*.

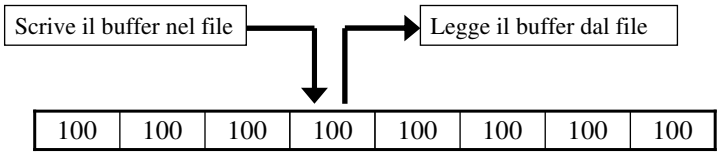
Per gestire l'accesso diretto è necessario:

1. che i record abbiano tutti la stessa lunghezza;
2. usare i seguenti metodi della classe **RandomAccessFile**:
   - **String readLine**() // legge un record
   - **void writeBytes** (**String** s) // scrive un record
   - **void seek** (**long** pos) // effettua accesso diretto

Scrittura e lettura

Per considerare i record a lunghezza fissa, occorre che:

- il metodo *scriviRec*() trasformi l'oggetto *l* di classe *Libro* in una stringa di lunghezza fissa (ad es. 100 caratteri) terminata con "\n";
- il metodo *leggiRec*() legga dal file una stringa (ad es. di 100 caratteri) e la restituisca come oggetto di classe *Libro*.



Scrittura
- uso StringBuffer

Poiché per gli oggetti di classe String non sono modificabili, occorre la classe **StringBuffer** che ha i seguenti metodi:

Package standard (**java.lang**)

- **StringBuffer** rec = **new StringBuffer** (**int** n); // costruttore
- **StringBuffer append** (**String** str) // accoda una stringa *str*
- **StringBuffer delete** (**int** inizio, **int** fine); // cancella da *inizio* a *fine*
- **String toString**() // trasforma in stringa

Scrittura
- uso StringBuffer

In questo modo il metodo *scriviRec*():

- crea un oggetto *rec* di classe **StringBuffer** di 100 caratteri;
- appende in *rec* uno ad uno i singoli attributi dell'oggetto *l* (con eventuale conversione) utilizzando, ad esempio, un carattere separatore (es. ";");
- aggiunge spazi fino al 100-esimo carattere;
- trasforma *rec* in una stringa;
- accede con **seek**() alla posizione richiesta, tenendo conto che:
  - il parametro di **seek**() deve essere un **long** che indica i byte da saltare;
  - se *pos* è la posizione da scrivere, occorre **seek** ((*pos-1*)*100)
- scrive la stringa mediante **writeBytes**().

Scrittura
- uso StringBuffer

```
public void scriviRec (RandomAccessFile raf, Libro l)
{   StringBuffer rec = new StringBuffer (100);
    try
    {       c=l.getCodice();
            aut=l.getAutore();
            rec.delete(0, 100);                    // pulisce il buffer
            rec.append(String.valueOf(c) + ";" + aut + ";“ + ……altri campi);
            for (int i=rec.length(); i<99; i++)    // aggiunge spazi in coda
               rec.append(" ");
            rec.append("\n");                      // aggiunge il fine linea
            raf.seek((long)(c-1)*100);
            raf.writeBytes(rec.toString());
    }
    catch (IOException e)
    {       System.out.println("Errore");   }
}
```

Separatore dei campi

Lettura
- uso StringTokenizer

Una volta scritti, possiamo rileggere i record con il metodo

String readLine() (che legge una stringa)..

Ma come possiamo estrarre le varie sottostringhe per individuare gli attributi?

Funzioni classe StringTokenizer	
METODO	**EFFETTO**
StringTokenizer (**String** *str*))	Crea *str* come stringtokenizer vuoto
StringTokenizer (**String** *str*, **String** *delim*)	Crea *str* stringtokenizer con delimitatore *delim*
int countTokens ( )	numero di token ancora da analizzare
boolean hasMoreTokens ( )	**true** se esistono ancora token da analizzare
String nextToken ( )	restituisce il token successivo

Lettura
- uso StringTokenizer

In questo modo il metodo *leggiRec*():

- crea un oggetto *rec* di classe **StringBuffer** di 100 caratteri;
- legge una stringa dal file mediante il metodo **readLine**() e la appende in *rec*;
- trasforma *rec* in un oggetto *rec1* di classe **StringTokenizer**;
- estrae, uno ad uno, i *token* da *rec1* con **nextToken()** e ricostruisce gli attributi dell'oggetto *l* di classe Libro;

Lettura
- uso StringTokenizer

```
public Libro leggiRec (RandomAccessFile raf)
{   StringBuffer rec = new StringBuffer (100);
    try   {  rec.append (raf.readLine());                // legge stringa
             StringTokenizer rec1=new StringTokenizer (rec.toString());
             c=Integer.parseInt (rec1.nextToken(";"));   // estrae token
             aut=rec1.nextToken(";");
             estrae i token successivi;
             Libro l = new Libro(c, aut, altri campi);    // crea oggetto
             return l;          }
    catch (IOException e)
    {       System.out.println("Fine file");  return null;
    }
    catch (NumberFormatException nfe)
    {       System.out.println("Fine file");  return null;
    }
}
```

A fine file dà questa eccezione

Ricerca

Siamo arrivati al metodo di ricerca con accesso diretto che, grazie alle considerazioni precedenti, diventa di semplicissima implementazione.

```
public Libro ricerca (int cod)
{   Libro l=null;
    try   {  RandomAccessFile raf = new RandomAccessFile (filename, "r");
             raf.seek((long)(cod-1)*100);
             l=leggiRec(raf);
             return l;
          }
    catch(IOException e)
    {       System.out.println("Fine file");
            return null;
    }
}
```


Eliminazione

L'eliminazione pone 0 nel campo codice del record desiderato, accedendolo con accesso diretto.

```
public void eliminazione (int cod)              // cancellazione logica
    {   StringBuffer rec=new StringBuffer (100);
        try { Libro l = null;
              apre il file, si posiziona in posizione cod e legge il record;
              l.setCodice(0);                   // ... pone a 0 il codice
              si riporta in posizione cod;
              ricostruisce la stringa con gli spazi;
              raf.writeBytes(rec.toString());   //... scrive string buffer su disco
            }
            catch (IOException e)
            {  System.out.println("Fine file");  }
    }
```

Argomenti

- Utilizzo di RandomAccessFile
  - inserimento
  - stampa
  - ricerca
- Specifiche dell'applicazione
  - ipotesi di lavoro
  - cancellazione logica
  - classi impiegate
- La classe **RandomAccessFile**
- Inserimento
- Stampa
- Scrittura e lettura
- Scrittura
  - uso StringBuffer
- Lettura
  - uso StringTokenizer
- Ricerca
- Eliminazione

Altre fonti di informazione

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