Corso di Programmazione
Anno Accademico 2013–2014
Prof. Giovanni Pighizzini

Note ed esercizi aggiuntivi

14. Stream di I/O (continuazione)

Esempio. Visualizza sul monitor il contenuto di un file di caratteri.

```
import java.io.FileReader;
import java.io.IOException;

class VisualizzaFile {

  public static void main(String[] args) throws IOException {
    //costruzione dello stream di caratteri
    FileReader frd = new FileReader(args[0]);

    int i;
    //lettura e visualizzazione
    while ((i = frd.read()) != -1)
      System.out.print((char)i);

    //chiusura dello stream
    frd.close();
  }
}
```

Note

- Il costruttore di `FileReader` può sollevare una `FileNotFoundException` (sottoclasse di `IOException`). Il metodo `read` e il metodo `close` possono sollevare una `IOException`. Poiché si tratta di *eccezioni controllate*, `main` deve intercettarle con un costrutto `try-catch` oppure delegarle esplicitamente al chiamante. Qui si è scelta la seconda alternativa: la delega esplicita al chiamante è espressa scrivendo `throws IOException` alla fine dell'intestazione di `main`. Accertatevi di avere compreso il significato di "eccezione controllata" e la differenza con "eccezione non controllata". Ricordatevi che il termine "controllata" non si riferisce a ciò che fa il programma in presenza di un'eccezione, ma ai controlli che effettua il compilatore relativamente al codice che potrebbe sollevare queste eccezioni.
- In questo esempio il file viene letto carattere per carattere. La lettura a caratteri è decisamente rudimentale e legata alla rappresentazione dei file, che varia in base al sistema operativo utilizzato. Provate a sostituire l'istruzione all'interno del ciclo `while` con

  ```
  System.out.print(i + " " + (char)i);
  ```

In questo modo ogni carattere sarà preceduto dal relativo codice Unicode. In alcuni sistemi la fine di una riga viene rappresentata mediante un solo carattere (di codice 10 oppure 13), in altri mediante due caratteri (di codice 13 e 10). In base alla codifica utilizzata dal vostro sistema otterrete risultati differenti.

Sebbene i file di testo siano effettivamente sequenze di caratteri, è bene vederli in maniera più astratta, come sequenze di righe, dove ciascuna riga è una sequenza di caratteri (ritorno a capo escluso). Per la lettura è bene utilizzare una metodologia che mantenga questa astrazione, leggendo righe e non singoli caratteri, astraendo così rispetto ai dettagli relativi alla rappresentazione. A tale scopo si può utilizzare la classe `BufferedReader`.

Esempio. Visualizza sul monitor il contenuto di un file di testo.

```
import java.io.FileReader;
import java.io.IOException;
import java.io.BufferedReader;

class VisualizzaFile {

  public static void main(String[] args) throws IOException {
    //costruzione dello stream di caratteri
    FileReader frd = new FileReader(args[0]);
    BufferedReader bfr = new BufferedReader(frd);

    String str;
    //lettura e visualizzazione
    while ((str = bfr.readLine()) != null)
      System.out.println(str);

    //chiusura dello stream
    bfr.close();
    frd.close();
  }
}
```

Note
La classe `BufferedReader` permette di vedere un flusso di caratteri organizzato per righe. Il metodo `readLine` permette di leggere una riga alla volta (il valore `null` indica il raggiungimento della fine del file).

Esercizio 14.1
Scrivete un'applicazione che legga un file di testo e calcoli la media delle lunghezze delle righe presenti. L'implementazione è abbastanza semplice se si utilizza la classe `BufferedReader`. Risulta invece decisamente complicata (e dipendente dal sistema sottostante per il riconoscimento della fine delle righe) nel caso si utilizzi solo `FileReader`.

Esempio. Visualizza sul monitor il contenuto di un file di testo. In caso di errore il programma fornisce un messaggio appropriato e termina.

```
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;
```

```
import java.io.BufferedReader;

class VisualizzaFile {

  public static void main(String[] args) {
    try {
      //costruzione dello stream di caratteri
      FileReader frd = new FileReader(args[0]);
      BufferedReader bfr = new BufferedReader(frd);

      String str;
      //lettura e visualizzazione
      while ((str = bfr.readLine()) != null)
        System.out.println(str);

      //chiusura dello stream
      bfr.close();
      frd.close();
    } catch (ArrayIndexOutOfBoundsException e) {
      System.err.println("Errore: manca il nome del file!");
    } catch (FileNotFoundException e) {
      System.err.println("Il file " + args[0] + " non esiste!");
    } catch (IOException e) {
      System.err.println("Errore durante la lettura da file: " + e);
    }
  }
}
```

Note

- Poiché il metodo `main` intercetta le eccezioni, non è necessario scrivere `throws IOException` nell'intestazione (anzi ciò sarebbe logicamente scorretto).
- `System.err` è un riferimento predefinito allo *standard error*, un canale di output, solitamente associato al monitor, utilizzato di solito per i messaggi d'errore.
- Si noti l'ordine con cui vengono intercettate le eccezioni, in particolare le ultime due. Provate a scambiare gli ultimi due blocchi `catch` e fate compilare il codice. Il compilatore segnala un errore. Spiegatene il motivo.

Esercizio 14.2
Il comando `wc` (Unix/Linux) conta il numero di righe, parole e caratteri presenti in un file di testo il cui nome viene specificato sulla linea di comando (se non viene specificato il file il comando legge da tastiera). Ad esempio se il file contiene

```
cane
gatto pipistrello
topo
```

il risultato prodotto è[1]

```
       3       4      28
```

[1] Nel conteggio dei caratteri il comando conta anche l'*end-of-line* (fine riga).

Scrivete un'applicazione che abbia lo stesso comportamento.

Esempio. Crea una copia di un file di caratteri. Il nome del file da cui leggere e il nome del file in cui scrivere sono forniti come argomenti, nell'ordine, sulla riga di comando.

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

class CopiaFile {

  public static void main(String[] args) throws IOException {
    //costruzione degli stream di caratteri
    FileReader sorg = new FileReader(args[0]);
    FileWriter dest = new FileWriter(args[1]);

    int i;
    //lettura e visualizzazione
    while ((i = sorg.read()) != -1) {
      dest.write(i);
    }

    //chiusura dello stream
    sorg.close();
    dest.close();
  }
}
```

Esempio. Crea una copia di un file di testo. Il nome del file da cui leggere e il nome del file in cui scrivere sono forniti come argomenti, nell'ordine, sulla riga di comando.

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;

class CopiaFile {

  public static void main(String[] args) throws IOException {
    //costruzione degli stream di caratteri
    FileReader frd = new FileReader(args[0]);
    BufferedReader sorg = new BufferedReader(frd);
    FileWriter fwt = new FileWriter(args[1]);
    BufferedWriter dest = new BufferedWriter(fwt);

    String str;
    //lettura e visualizzazione
    while ((str = sorg.readLine()) != null) {
      dest.write(str);
      dest.newLine();
    }
```

```
    //chiusura dello stream
    sorg.close();
    frd.close();
    dest.close();
    fwt.close();
  }
}
```

Esercizio 14.3
L'applicazione dell'esempio precedente non prevede il trattamento delle eccezioni, rinviandole quindi alla Java Virtual Machine. Modificatela in modo che le eccezioni siano trattate.

Esercizio 14.4
Modificate l'applicazione dell'esercizio 14.3 in modo che nel caso il file destinazione esista già chieda all'utente se vuole cancellarne il contenuto, se vuole copiare il contenuto del file sorgente in coda al file destinazione o se vuole rinunciare ad effettuare la copia. Per stabilire se un file esiste si può utilizzare uno dei metodi forniti dalla classe `File` del package `java.io`. Tale classe fornisce vari metodi utili relativi ai file (per avere maggiori informazioni, consultate la documentazione della classe o il paragrafo 13.3 del libro di testo).