LETTURA E SCRITTURA DI FILE

LA CLASSE FILE

- La classe File fornisce una rappresentazione astratta (non è una classe astratta!) del percorso (pathname) di una cartella (directory) o di un documento (file)

- Sistemi operativi e interfacce utilizzano stringhe dipendenti dal sistema in uso. Questa classe mette a disposizione un'astrazione comunque gerarchica ma indipendente dal sistema

- Esempio: nei percorsi si usa il separatore "/" in Unix, ma Windows ammette anche "\"

PERCORSI ASSOLUTI E RELATIVI

- Un percorso può essere assoluto o relativo. Quando è assoluto, contiene intrinsecamente tutte le informazioni per localizzare un file. Se invece è relativo, deve essere interpretato sulla base di altre informazioni

- Di default, le classi nel package java.io risolvono i percorsi relativi rispetto alla directory corrente. Tipicamente, si tratta della cartella in cui si invoca la Java Virtual Machine

- Gli attributi statici File.pathSeparator e File.separator sono costanti che indicano i separatori usati nel sistema in uso (Es. ":" e "/")
  - pathSeparator è usato per separare vari percorsi in una lista (es. nella variabile di sistema PATH)
  - separator è usato per separare le varie parti che formano un percorso

LA CLASSE FILE

- Costruttore principale: File(String pathname)

- boolean canExecute()
- boolean canRead()
- boolean canWrite()

  testano i permessi da parte dell'applicazione

- boolean delete()

  cancella il file o la cartella

VERIFICHE SUI FILE

- Visto che la stessa classe può essere usata per effettuare operazioni su file e su directory, spesso occorre controllare il tipo di oggetto su cui si sta operando

- Il modo più semplice è invocare il metodo boolean isFile(), oppure boolean isDirectory()

- In aggiunta, il metodo boolean exists() verifica se il file o la directory esistono

- boolean isHidden() verifica se si tratta di file nascosto

ALTRI METODI DELLA CLASSE FILE

- La classe File mette a disposizione un gran numero di metodi per gestire documenti e cartelle. Ad esempio:
  - String getName() restituisce il nome del file o directory
  - String getParent() restituisce il nome della cartella padre
  - String getPath() converte il nome astratto di un percorso a stringa
  - long lastModified() restituisce l'istante di ultima modifica
  - long length() restituisce la dimensione del file
  - String[] list() restituisce l'elenco di file e cartelle contenute nella cartella denotata dal path corrente (con numerose varianti…)
  - …

LA CLASSE FILEREADER

- FileReader è pensata per leggere dei file testuali
- Costruttori: FileReader(File file), FileReader(String fileName), …
- Metodi (è supportata la lettura sequenziale):
  - int read(): legge un carattere, restituito in uscita (-1 se si è giunti alla fine del file)
  - int read(char[] buffer, int offset, int length): legge length caratteri inserendoli alla posizione offset dell'array di caratteri buffer (restituisce il numero di caratteri effettivamente letti)
  - long skip(long n): avanza di n caratteri (restituisce l'avanzamento realmente effettuato)
- Esempio:

```
FileReader fr = new FileReader("a.txt");
char[] cbuf = new char[30];
fr.read(cbuf, 2, 3);
fr.read(cbuf, 10, 7);
System.out.println(cbuf);
```

a.txt:

contenuto1

contenuto2

contenuto3

Output:

␣␣con␣␣␣␣␣tenuto1␣␣␣␣␣␣␣␣␣␣␣␣␣␣

LA CLASSE BUFFEREDREADER

- Un BufferedReader legge testo da un flusso di caratteri in ingresso, in modo da rendere efficiente la lettura sequenziale di caratteri e righe

- Sarebbe possibile specificare anche la dimensione del buffer, ma quella di default è sufficientemente capiente per la maggior parte delle applicazioni

- Tra i principali metodi: int read() legge un singolo carattere, String readLine() legge una riga di testo e long skip(long n) salta nella lettura un certo numero di caratteri

ESEMPIO: LETTURA RIGA PER RIGA

```
File file = new File("miofile.txt");
if (file.isFile()) {
  FileReader fileReader = new FileReader(file);
  BufferedReader bufReader = new BufferedReader(fileReader);
  try {
    String line;
    // lettura del file di testo riga per riga
    while ((line = bufReader.readLine()) != null) {
       ; // operazioni su line
    }
  } finally {
    // rilascio delle risorse
    bufReader.close();
    fileReader.close();
  }
}
```

TRY-WITH-RESOURCES

- In generale tutte le risorse allocate con i metodi di lettura/scrittura di file devono essere rilasciate invocando il metodo close()
- Nell'esempio precedente, vengono gestite correttamente con l'utilizzo di try…finally le eccezioni sulla lettura del file, ma potrebbe succedere che venga generato un errore proprio nell'utilizzo del metodo close() e questa eccezione non viene gestita
- Da Java 7 è preferibile usare il costrutto try-with-resources, che automaticamente chiude il flusso di input/output
- Anche in questo caso è possibile aggiungere le clausole catch e finally, però quando si giunge al blocco corrispondente lo stream è già stato chiuso

ESEMPIO: LETTURA RIGA PER RIGA (2)

```
File file = new File("miofile.txt");
if (file.isFile()) {
  try (
    FileReader fileReader = new FileReader(file);
    BufferedReader bufReader = new BufferedReader(fileReader)
  ) {
    String line;
    // lettura del file di testo riga per riga
    while ((line = bufReader.readLine()) != null) {
       ; // operazioni su line
    }
  }
}
```

ESERCIZIO

(EasyScore.java)

- Scrivere un software che interpreta una rappresentazione testuale di una partitura, in cui ogni riga corrisponde a uno strumento (monodico) e ogni gruppo di caratteri separato da virgole a una pulsazione di valore ritmico predefinito (ad esempio da un ottavo)
- Le altezze sono scritte in notazione anglosassone (ad es. C4 rappresenta il Do della quarta ottava, e Eb5 il Mi bemolle della quinta). Si considerino solo alterazioni singole (# e b)
- Le pause sono contrassegnate come note «degeneri» tramite la lettera R
- Le durate si ricavano dal numero di virgole che separano le stringhe: ogni stringa mancante tra virgole costituisce l'allungamento della nota (o della pausa) precedente
- Ad esempio, l'incipit di Fra' Martino sarebbe:

```
C4,D4,E4,C4,C4,D4,E4,C4,E4,F4,G4,,E4,F4,G4,,G4,,G4
R,,,,,,,,C4,D4,E4,C4,C4,D4,E4,C4,E4,F4,G4
```

- Hint: stringa.split("char") restituisce un array di String separando la stringa principale per il carattere specificato (più precisamente il metodo accetta espressioni regolari)

LA CLASSE FILEINPUTSTREAM

- FileInputStream è pensata per leggere dei file binari (es.: file audio, immagini, ecc.)

- Costruttori: FileInputStream(File file), FileInputStream(String fileName), …

- Metodi (è supportata la lettura sequenziale):
  - int read(): legge un byte, restituito in uscita (-1 se si è giunti alla fine del file)
  - int read(byte[] buffer): legge buffer.length byte inserendoli nell'array di byte buffer (restituisce il numero di caratteri effettivamente letti)
  - int read(byte[] buffer, int offset, int length): legge length byte inserendoli alla posizione offset dell'array di byte buffer (restituisce il numero di caratteri effettivamente letti)
  - long skip(n): avanza di n byte (restituisce l'avanzamento realmente effettuato)

SCRITTURA DI FILE

- Si può scrivere su file usando le classi: FileWriter, BufferedWriter, e FileOutputStream
- FileWriter scrive file di testo immediatamente su disco
- BufferedWriter scrive file di testo solo a buffer pieno, ottimizzando l'accesso al disco (in più offre il metodo newLine() per andare a capo)
- FileOutputStream è usato per scrivere file binari
- I costruttori/metodi sono simili a quelli visti in lettura (read > write)

ESEMPIO DI SCRITTURA FILE

```
try (FileWriter w = new FileWriter("pitches.txt");
  BufferedWriter bw = new BufferedWriter(w)) {
  bw.write('A');
  bw.write('4');
  bw.newLine();
  bw.write("G4");
} catch (IOException ex) {
}
```

ESERCIZIO

(EasyScore.java)

- Modificare il software precedente per scrivere su file il contenuto della struttura dati score
- Ogni riga rappresenta uno strumento
- Utilizzare il metodo toString() già esistente per le singole note
- Il contenuto del file deve risultare incolonnato sulle singole note, considerando 12 caratteri per ogni granulo (nel codice: l'ottavo)
- Ad esempio, l'incipit di Fra' Martino sarebbe:

```
C4 [1/8]    D4 [1/8]    E4 [1/8]    C4 [1/8]    C4 [1/8]    D4 [1/8]    E4 [1/8]    C4 [1/8]↵
E4 [1/8]    F4 [1/8]    G4 [2/8]                E4 [1/8]    F4 [1/8]    G4 [2/8]

Rest [8/8]                                                                              ↵
C4 [1/8]    D4 [1/8]    E4 [1/8]    C4 [1/8]    C4 [1/8]    D4 [1/8]    E4 [1/8]    C4 [1/8]
```