Azzolini Riccardo

Stream

1 Stream di input e output

La libreria standard di Java mette a disposizione il package `java.io` per la gesione di input e output. Tale package è basato sul concetto di **stream** (*flusso*): una sequenza ordinata di dati che ha una **sorgente** e una **destinazione**.

- Per prelevare dati da una sorgente, l'applicazione deve aprire uno stream collegato a tale sorgente e leggere sequenzialmente (una dopo l'altra) le informazioni in esso contenute.
- Per inviare dati a una destinazione, l'applicazione deve aprire uno stream collegato alla destinazione e scrivere su di esso le informazioni.

2 Classificazione degli stream

Il package `java.io` contiene una gerarchia di classi per la gestione di due tipi di stream:

stream di caratteri: l'unità minima di informazione è un carattere

stream di byte: l'unità di informazione è un singolo byte

Il tipo di stream gestito e la funzione (input o output) delle varie classi si possono riconoscere in base ai loro nomi:

Tipo	Input	Output
caratteri	`*Reader`	`*Writer`
byte	`*InputStream`	`*OutputStream`

3 `Reader`: lettura di stream di caratteri

La classe astratta `Reader` definisce i metodi di base per la gestione di un qualsiasi stream di lettura di caratteri. I principali sono:

- `public int read() throws IOException`
  Legge e restituisce un singolo carattere, oppure restituisce -1 se si è raggiunta la fine dello stream (per questo il tipo restituito è `int` e non `char`).

- `public int read(char[] buf) throws IOException`
  Legge un numero di caratteri pari alla lunghezza dell'array passato come argomento e li memorizza in esso.
- `public abstract void close() throws IOException`
  Chiude lo stream, rilasciando la risorsa (il file collegato allo stream). In seguito, eventuali invocazioni di metodi che operano sullo stream sollevano un'eccezione `IOException`.

Le principali sottoclassi di `Reader` sono:

```
Reader
  InputStreamReader
    FileReader          (sorgente)
  BufferedReader
    LineNumberReader
  CharArrayReader       (sorgente)
  StringReader          (sorgente)
  FilterReader
    PushbacReader
  PipedReader           (sorgente)
```

Alcune di queste (`FileReader`, `CharArrayReader`, `StringReader` e `PipedReader`) leggono direttamente caratteri da una sorgente, mentre le altre forniscono meccanismi per manipolare le informazioni lette.

3.1 `InputStreamReader` e `FileReader`

`InputStreamReader` permette di tradurre uno stream di byte in uno stream di caratteri.

La sua sottoclasse `FileReader` consente quindi di leggere, carattere per carattere, i contenuti di un file. In altre parole, le sue istanze rappresentano stream di caratteri in lettura collegati a sorgenti di tipo file.

Il principale costruttore di `FileReader` è:

```
public FileReader(String nomeFile) throws FileNotFoundException
```

Esso crea un oggetto stream collegato al file di cui è specificato il nome. Se non esiste un file con tale nome, viene sollevata un'eccezione `FileNotFoundException`.

3.2 `BufferedReader`

Le istanze di questa classe non dati direttamente da una sorgente specifica, bensì da un altro stream di caratteri: il costruttore richiede infatti un argomento di tipo `Reader`.

Un'istanza di `BufferedReader` legge un certo numero di caratteri alla volta dallo stream sottostante e li conserva in una memoria interna (*buffer*). Ciò ha due vantaggi:

- rende più efficiente il processo di lettura
- consente di organizzare le sequenze di caratteri in strutture più complesse

Oltre ai metodi di `Reader`, questa classe mette a disposizione anche

```
public String readLine() throws IOException
```

che legge e restituisce una riga di testo, oppure restituisce `null` se è stata raggiunta la fine dello stream. Le righe sono considerate concluse da `\r`, `\n` o `\r\n`.

3.3 `CharArrayReader` e `StringReader`

Queste due classi permettono di leggere caratteri da sorgenti di dati in memoria: rispettivamente, da un array di caratteri e da una stringa.

4 `Writer`: scrittura di stream di caratteri

I principali metodi per gestire gli stream di scrittura di caratteri, definiti nella classe astratta `Writer`, sono:

- `public void write(int c) throws IOException`
  Scrive `c` sullo stream, sotto forma di carattere (vengono presi in considerazione solo i 16 bit meno significativi).
- `public abstract void write(char[] buf, int offset, int c) throws IOException`
  Scrive sullo stream i `c` caratteri contenuti nell'array `buf` a partire dalla posizione `offset`, cioè da `buf[offset]` a `buf[offset + c - 1]`.
- `public void write(char[] buf) throws IOException`
  Scrive tutti i caratteri contenuti nell'array `buf`. Corrisponde a `write(buf, 0, buf.length)`.
- `public void write(String s, int offset, int c) throws IOException`
  Scrive i `c` caratteri contenuti nella stringa `s` alle posizioni da `offset` a `offset + c - 1`.

- `public void write(String s) throws IOException`
  Scrive tutti i caratteri della stringa `s`. Equivale a `write(s, 0, s.length())`.
- `public abstract void flush() throws IOException`
  Forza la scrittura effettiva sullo stream di eventuali caratteri che, per motivi di efficienza, sono stati solo memorizzati in un buffer.
- `public abstract void close() throws IOException`

Le principali sottoclassi di `Writer` sono:

```
Writer
  OutputStreamWriter
    FileWriter        (destinazione)
  BufferedWriter
  CharArrayWriter     (destinazione)
  StringWriter        (destinazione)
  PrintWriter
  FilterWriter
  PipedWriter         (destinazione)
```

Come nella gerarchia di `Reader`, alcune classi scrivono direttamente su una destinazione (`FileWriter`, `CharArrayWriter`, `StringWriter` e `PipedWriter`), mentre le altre forniscono meccanismi per manipolare i dati.

4.1 `FileWriter`

La classe `FileWriter` consente di scrivere su file di caratteri.

I suoi principali costruttori sono:

- `public FileWriter(String fileName, boolean append) throws IOException`
  Crea un oggetto per la scrittura sul file di caratteri con il nome specificato. Se il file non esiste viene creato. Se invece esiste:
  - se `append` è `false`, viene sovrascritto
  - se `append` è `true`, i caratteri scritti sullo stream vengono accodati al contenuto attuale

  Infine, se il nome fornito come argomento corrisponde a un file che non può essere scritto o a una cartella viene sollevata un'eccezione `IOException`.
- `public FileWriter(String fileName) throws IOException`
  Equivale a `FileWriter(fileName, false)`, cioè se il file esiste lo sovrascrive.

4.2 `BufferedWriter`

La classe `BufferedWriter` fornisce un meccanismo di bufferizzazione che rende più efficienti le operazioni di scrittura su un altro stream (passato come argomento al costruttore).

Oltre ai metodi specificati da `Writer`, questa classe ha anche un metodo

```
public void newLine() throws IOException
```

che termina la riga corrente, scrivendo sullo stream un separatore di riga.

4.3 `PrintWriter`

Questa classe mette a disposizione metodi per scrivere tipi primitivi e oggetti in forma testuale su un altro stream, passato come argomento al costruttore.

Per ogni tipo primitivo (e per `String` e `Object`), `PrintWriter` mette a disposizione un metodo `print`, che riceve come argomento un valore di tale tipo e ne stampa la rappresentazione testuale, e il corrispondente metodo `println`, che dopo aver stampato il valore scrive anche un separatore di riga.

5 La classe `File`

Le sue istanze forniscono una rappresentazione astratta di file e directory e mettono a disposizione vari metodi utili per la loro gestione.

In particolare, gli oggetti di `File` rappresentano i *percorsi* (*pathname*) dei file e delle directory, in modo indipendente dal sistema operativo. Sono disponibili alcuni campi statici contenenti informazioni sulla rappresentazione dei percorsi nel sistema operativo corrente, tra i quali ad esempio

```
public static final char separator
```

che contiene il carattere utilizzato per separare le componenti del percorso (come `/` o \).

Tutte le classi che rappresentano stream collegati a file hanno un costruttore che riceve come argomento un oggetto di tipo `File`.

Il principale costruttore di `File` è:

```
public File(String pathname)
```

`File` definisce numerosi metodi per ottenere informazioni sul file rappresentato dall'oggetto, tra cui:

- `public boolean exists()`
  Verifica se il file esiste.
- `public boolean isDirectory()`
  Verifica se il file esiste ed è una directory.
- `public boolean isFile()`
  Verifica se il file è un file *normale*, cioè non è una directory e soddisfa altre proprietà che dipendono dal sistema operativo.
- `public String getName()`
  Restituisce il nome del file (l'ultima componente del percorso completo).
- `public String getAbsolutePath()`
  Restituisce il percorso completo del file.
- `public long lastModified()`
  Restituisce la data dell'ultima modifica del file, espressa come numero di millisecondi trascorsi dalle ore 00:00:00 GMT del primo gennaio 1970.
- `public long length()`
  Restituisce la dimensione (in byte) del file.
- `public boolean canWrite()`
  Controlla se il file può essere scritto.
- `public boolean canRead()`
  Controlla se il file può essere letto.
- `public String[] list()`
  Restituisce un array di stringhe contenente i nomi di file e directory presenti nella directory rappresentata dall'oggetto. Se l'oggetto non rappresenta una directory o si verifica un errore di input/output, restituisce `null`.

Inoltre, `File` fornisce anche alcuni metodi per operare sul file (o directory) rappresentato:

- `public boolean createNewFile() throws IOException`
  Crea il file rappresentato dall'oggetto, se non esiste già. Restituisce `true` se il file non esisteva ed è stato creato con successo, mentre restituisce `false` se esisteva già.
- `public boolean mkdir()`
  Crea la directory rappresentata, restituendo `true` se e solo se l'operazione ha successo.
- `public boolean delete()`
  Cancella il file rappresentato, o la directory rappresentata purché sia vuota. Restituisce `true` se e solo se l'operazione ha successo.

- `public void deleteOnExit()`
  Richiede che il file/directory rappresentato sia cancellato dalla JVM prima che quest'ultima termini. Non è possibile annullare la richiesta di cancellazione.

Infine, ci sono anche due metodi statici chiamati `createTempFile`, che creano dei file temporanei.

6 Stream di byte

Le gerarchie di classi per la lettura e scrittura di stream di byte sono parallele a quelle per gli stream di caratteri.

La classe astratta per la lettura è `InputStream`. I principali metodi che essa definisce sono:

- `public abstract int read() throws IOException`
  Legge e restituisce il valore di un byte, oppure restituisce -1 se si è raggiunta la fine dello stream.
- `public int read(byte[] b) throws IOException`
- `public void close() throws IOException`
- `public int available() throws IOException`
  Restituisce il numero di byte che possono essere letti dallo stream.

La gerarchia delle sottoclassi di `InputStream` è:

```
InputStream
  FileInputStream          (sorgente)
  PipedInputStream         (sorgente)
  FilterInputStream
    LineNumberInputStream
    DataInputStream
    BufferedInputStream
    PushBackInputStream
  ByteArrayInputStream     (sorgente)
  SequenceInputStream
  StringBufferInputStream  (sorgente)
  ObjectInputStream
```

La scrittura, invece, si basa sulla classe astratta `OutputStream`, che definisce:

- `public abstract void write(int b) throws IOException`
  Scrive un byte, fornito come `int` per non dover effettuare cast in seguito alle operazioni aritmetiche (che tra `byte` danno risultato `int`): vengono considerati solo gli 8 bit meno significativi.
- `public write(byte[] b) throws IOException`
- `public void flush() throws IOException`
- `public void close() throws IOException`

La gerarchia delle sottoclassi di `OutputStream` è:

```
OutputStream
  FileOutputStream        (destinazione)
  PipedOutputStream       (destinazione)
  FilterOutputStream
    DataOutputStream
    BufferedOutputStream
    PrintStream
  ByteArrayOutputStream   (destinazione)
  ObjectOutputStream
```

6.1 `DataInputStream` e `DataOutputStream`

Le classi `DataInputStream` e `DataOutputStream` consentono rispettivamente di scrivere e leggere valori di tipo primitivo su uno stream di byte.

Ciascuna delle due classi mette a disposizione un unico costruttore:

- `public DataInputStream(InputStream in)`
- `public DataOutputStream(OutputStream out)`

Queste classi mettono a disposizione vari metodi, tra cui quelli di lettura e scrittura per tutti i tipi primitivi:

DataInputStream	DataOutputStream
`boolean readBoolean()`	`void writeBoolean(boolean v)`
`char readChar()`	`void writeChar(int v)`
`byte readByte()`	`void writeByte(int v)`
`short readShort()`	`void writeShort(int v)`
`int readInt()`	`void writeInt(int v)`
`long readLong()`	`void writeLong(long v)`
`float readFloat()`	`void writeFloat(float v)`
`double readDouble()`	`void writeDouble(double v)`

6.2 `PrintStream`

La classe `PrintStream` fornisce metodi per scrivere rappresentazioni di tipi primitivi e oggetti su uno stream di output.

Essa mette a disposizione un metodo `print` e un metodo `println` per ogni tipo primitivo, per `String` e per `Object` (stampano la stringa restituita da `toString`).

Per renderne più agevole l'uso, questi metodi *non sollevano eccezioni controllate*: è invece disponibile il metodo

```
public boolean checkError()
```

che restituisce `true` se si è verificata un'`IOException` durante l'ultima operazione.

6.2.1 Il metodo `printf`

`PrintStream` (così come altri stream di output) definisce anche un metodo con un *numero variabile di argomenti* che consente di scrivere stringhe specificandone il formato:

```
public PrintStream printf(String format, Object... args)
```

La *stringa di formattazione* `format` specifica come rappresentare gli altri argomenti `args`: essa può contenere testo prefissato e un numero qualsiasi di *specificatori di formato*.

Uno specificatore di formato ha la struttura:

```
%[indice_argomento$][flags][ampiezza][.precisione]rappresentazione
```

- `indice_argomento` indica il numero dell'argomento da rappresentare. Gli argomenti sono numerati a partire da 1. Se viene omesso, la corrispondenza tra specificatori e argomenti è posizionale (il primo specificatore si riferisce al primo argomento, e così via).
- I `flags` specificano modifiche al comportamento standard dello specificatore (ad esempio, `-` indica l'allineamento a sinistra).
- `ampiezza` indica la lunghezza della stringa da costruire.
- `precisione` (solo per argomenti `float` o `double`) indica quante cifre decimali devono essere rappresentate.
- `rappresentazione` specifica come deve essere rappresentato l'argomento. Ad esempio:
  - `d` (tipi interi): intero in notazione decimale
  - `o` (tipi interi): notazione ottale

- `s` (qualsiasi tipo): stringa
- `f` (`float` o `double`): notazione decimale
- `e` (`float` o `double`): notazione scientifica

Se la stringa di formattazione contiene errori di sintassi o non corrisponde agli argomenti forniti, viene sollevata un'eccezione non controllata `IllegalFormatException`.