INPUT – OUTPUT

Il package `java.io` include classi, interfacce ed eccezioni per la gestione dell'I/O.
Si può distinguere tra due classi di funzionalità:

- per leggere e scrivere dati (anche in maniera indipendente dal tipo di sorgente/destinazione)
- per accedere ad informazioni relative a file e directory (ossia per "gestire" iil file system). In questo caso, la classe fondamentale per accedere a tali informazioni è la classe **`File`**.

Il meccanismo di lettura/scrittura sequenziale dei dati è basato sul concetto di stream (oggetto su cui è possibile leggere e scrivere dati sequenzialmente).

Stream di byte

Input sequenziale

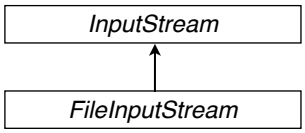
La classe astratta **`InputStream`** racchiude tutti i metodi per la lettura da uno stream di byte:

```
public abstract int read()
public int read(byte b[])
public int read(byte b[], int offset, int length)
```

tutte bloccanti; ritornano -1 se si è raggiunta la fine dello stream). Fornisce inoltre metodi di utilità quali:

```
public void close()
```

La classe **`FileInputStream`** è usata per l'accesso sequenziale a file di byte (ridefinisce i metodi della classe `InputStream` fornendone un'implementazione per i file).



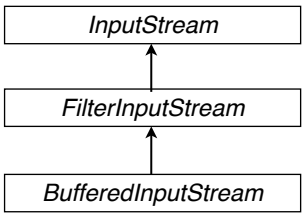
Costruttori:

```
public FileInputStream(String name)
public FileInputStream(File f)
public FileInputStream(FileDescriptor fd)
```

Ed aggiunge il metodo `public final FileDescriptor getFD()`

La classe **`FilterInputStream`** è la superclasse di tutte le classi che forniscono funzionalità di filtro; si appoggia ad un `InputStream` esistente passato al momento della costruzione.

La classe **`BufferedInputStream`** legge una sequenza di byte che mette in un buffer interno (ottimizza la lettura).



La classe **`DataInputStream`** fornisce metodi per leggere i diversi tipi base di Java da un `InputStream` esistente passato al momento della costruzione: `readBoolean`, `readByte`, `readInteger`, `readShort`, `readLong`, `readFloat`, `readDouble`, `readChar`, `readUnsignedByte`, `readUnsignedShort`. Es di utilizzo:

DataInputStream dis = new DataInputStream(new BufferedInputStream (new FileInputStream("prova.txt")));

Output Sequenziale

La classe astratta **`OutputStream`** racchiude tutti i metodi per la scrittura da uno stream di byte:

```
public abstract void write(byte b)
public void write(byte b[])
public void write(byte b[], int offset, int length)
```

Fornisce inoltre metodi di utilità quali:

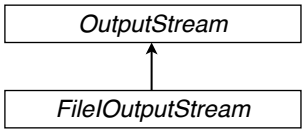
```
public void close()
public void flush()
```

La classe **`FileInputStream`** è usata per scrivere file sequenziali di byte (ridefinisce i metodi della classe `OutputStream` fornendone un'implementazione per i file).

Costruttori:

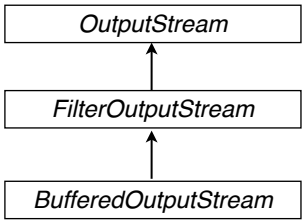
```
public FileOutputStream(String name)
public FileOutputStream(String name, boolean app)
public FileOutputStream(File f)
public FileOutputStream(FileDescriptor fd)
```

Ed aggiunge il metodo `public final FileDescriptor getFD()`



La classe **`FilterOutputStream`** è la superclasse di tutte le classi che forniscono funzionalità di filtro; si appoggia ad un `OutputStream` esistente passato al momento della costruzione.

La classe **`BufferedOutputStream`** sfrutta un buffer interno per ottimizzare la scrittura (che avviene effettivamente solo nel momento in cui tale buffer risulta pieno (o viene useguita una *flush*)).



La classe **`DataInputStream`** fornisce metodi per scrivere i diversi tipi base di Java da un `OutputStream` esistente passato al momento della costruzione: `writeBoolean`, `writeByte`, `writeInteger`, `writeShort`, `writeLong`, `writeFloat`, `writeDouble`, `writeChar`, `writeUnsignedByte`, `writeUnsignedShort`.

Stream di caratteri

La lettura e la scrittura di caratteri vengono effettuate tramite le classi `Reader` e `Writer` che inglobano le funzionalitù per leggere stream sequenziali di caratteri.

Reader

La classe astratta Reader racchiude tutti i metodi per la lettura da uno stream di caratteri:

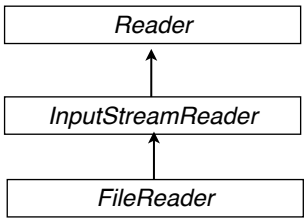
```
public abstract int read()
public int read(char c[])
public int read(char c[], int offset, int length)
```

tutte bloccanti; ritornano -1 se si è raggiunta la fine dello stream). Fornisce inoltre metodi di utilità quali:

```
public void close()
```

La classe astratta **`InputStreamReader`** costruisce un ponte tra gli stream di byte e queli di caratteri (effettua la conversione da byte a caratteri). Costruttore:

```
public InputStreamReader(InputStream is)
```



La classe **`FileReader`** è usata per l'accesso sequenziale a file di caratteri (ridefinisce i metodi della classe `InputStreamReader` fornendone un'implementazione per i file).
Costruttori:

```
public FileReader(String name)
public FileReader(File f)
public FileReader(FileDescriptor fd)
```

La classe **`FilterReader`** è la superclasse di tutte le classi che forniscono funzionalità di filtro; si appoggia ad un `Reader` esistente passato al momento della costruzione.

La classe **`BufferedReader`** legge una sequenza di caratteri che mette in un buffer interno (ottimizza la lettura). Si appoggia ad un `Reader` esistente (passato al momento della costruzione) ed aggiunge il metodo:

```
public String readLine()
```

Writer

La classe astratta **`Writer`** racchiude tutti i metodi per la scrittura da uno stream di caratteri:

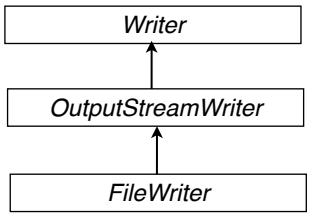
```
public abstract void write(char c)
public void write(char c[])
public void write(char c[], int offset, int length)
```

Fornisce inoltre metodi di utilità quali:

```
public void close()
public void flush()
```

La classe **`OutputStreamWriter`** costruisce un ponte tra gli stream di byte e queli di caratteri (effettua la conversione da byte a caratteri). Costruttore:

```
public OutputStreamReader(OutputStream is)
```



La classe **`FileWriter`** è usata per l'accesso sequenziale a file di caratteri (ridefinisce i metodi della classe `OutputStreamReader` fornendone un'implementazione per i file).

Costruttori:

```
public FileWriter(String name)
public FileWriter(File f)
public FileWriter(FileDescriptor fd)
```

La classe **`FilterWriter`** è la superclasse di tutte le classi che forniscono funzionalità di filtro; si appoggia ad un `Writer` esistente passato al momento della costruzione.

La classe **`BufferedWriter`** sfrutta un buffer interno per ottimizzare la scrittura (che avviene effettivamente solo nel momento in cui tale buffer risulta pieno (o viene useguita una *flush*)). Si appoggia ad un `Writer` esistente (passato al momento della costruzione) ed aggiunge il metodo: `public void newLine()`

StreamTokenizer

La classe **`StreamTokenizer`** fornisce funzionalitò per estrearre token da uno stream di caratteri.
Il costruttore prende come parametro un `Reader`.
Il metodo `int nextToken()` estra un token alla volta dal Reader:

- quando raggiunge la fine dello stream il metodo restituisce la costante statica `StreamTokenizer.TT_EOF`
- l'attributo `ttype` permette di identificare il tipo del token letto:
  - `StreamTokenizer.TT_WORD`
  - `StreamTokenizer.TT_NUMBER`
- il valore letto viene recuperato tramite l'attributo:
  - `String sval`, per le stringhe
  - `double nval`, per i numeri
  - `int ttype`, negli altri casi