Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita
Gli stream di I/O

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

Sommario

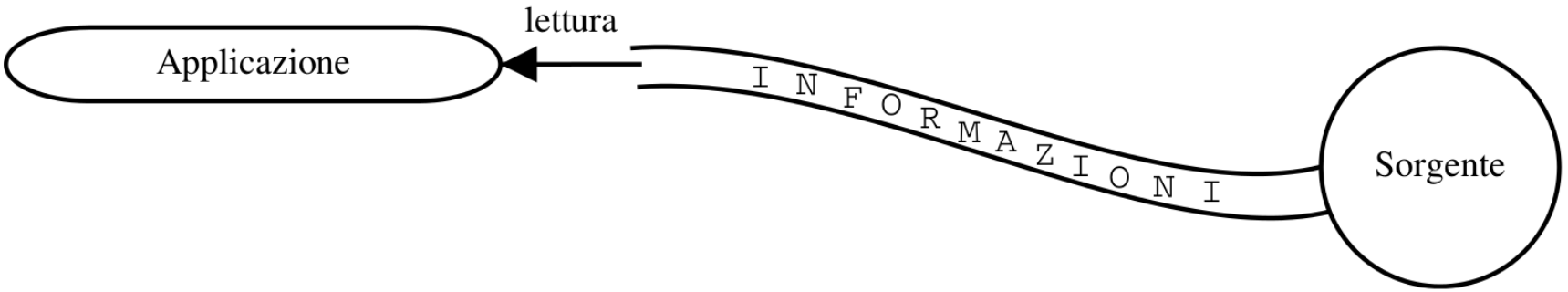
- Introduzione agli stream
  - Flussi di byte e di caratteri
  - Esempio: la classe FileReader
  - Le classi filtro
- Stream di caratteri
- La classe File
- Flussi di byte
  - La classe PrintStream

Gli stream di ingresso e uscita

- Il pacchetto **`java.io`** definisce le operazioni di ingresso e uscita in termini di *stream* (*flussi*).
- Gli stream sono sequenze ordinate di dati che hanno una *sorgente* (stream di ingresso) o una *destinazione* (stream di uscita).
- Le classi di **`java.io`** nascondono i dettagli del sistema operativo sottostante e dei dispositivi di ingresso e uscita coinvolti dalle operazioni di ingresso e uscita.

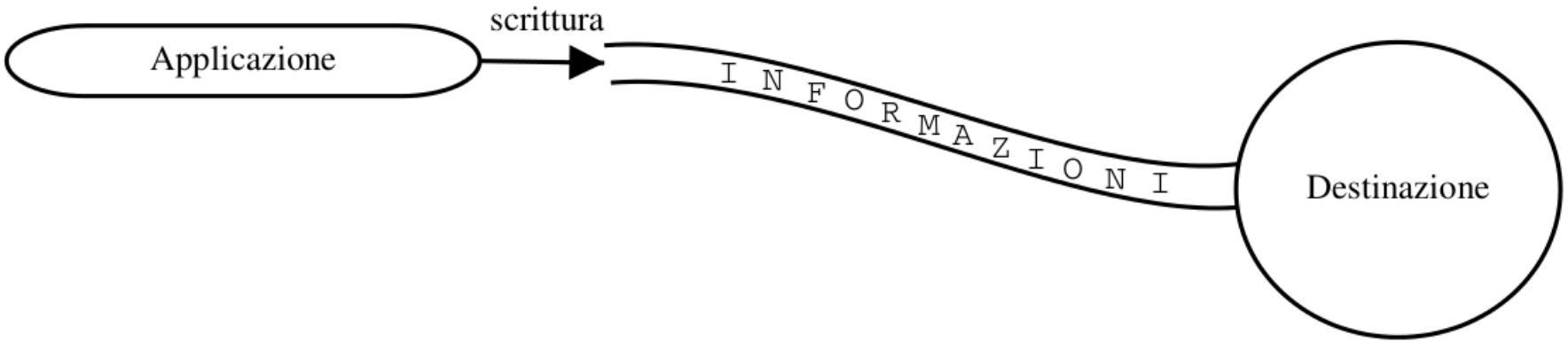
Flussi di input

- Per ricevere in ingresso dei dati, un'applicazione apre uno **stream** collegato a una sorgente di informazioni da cui **legge sequenzialmente** le informazioni.
  - La sorgente può essere in memoria, su disco, o anche remota



Flussi di output

- Per esportare informazioni, un'applicazione apre uno **stream** verso una destinazione (file, memoria, socket,...) e vi **scrive sequenzialmente** le informazioni.
  - La destinazione può essere in memoria, su disco, o anche remota



Lettura e scrittura

- *Lettura*

```
creazione dello stream
while (ci sono informazioni da leggere)
  leggi
chiusura dello stream
```

- *Scrittura*

```
creazione dello stream
while (ci sono informazioni da scrivere)
  scrivi
chiusura dello stream
```

Il package java.io

- Il pacchetto `java.io` è composto da due sezioni principali, riguardanti:
  - ***Flussi di byte (byte stream)***
    - L'unità di informazione gestita è il byte (8-bit).
    - I/O binario: immagini, dati in formato binario. . .
    - le classi che realizzano gli stream di byte sono indicate come
      - stream *di ingresso (input stream)*
      - stream *di uscita (output stream)*
  - ***Flussi di caratteri (character stream)***
    - L'unità di informazione gestita sono caratteri Unicode (16-bit).
    - I/O testuale
    - le classi che realizzano gli stream di caratteri sono indicate come
      - *lettori (reader)*
      - *scrittori (writer)*

Byte e caratteri

- Si utilizzano ***byte stream*** quando si vuole elaborare informazione di tipo binario, oppure quando si vuole elaborare, un carattere alla volta, file composti da caratteri rappresentati con un solo byte ciascuno secondo la codifica ASCII.
- Si utilizzano ***reader*** e ***writer*** quando si vuole elaborare informazione di tipo testuale, codificata in Unicode.

Errori

- È sempre possibile che un'operazione di ingresso o uscita non vada a buon fine.
  - Ad esempio si può cercare di aprire un file che non esiste oppure cercare di scrivere su di un file protetto in scrittura o leggere un file protetto in lettura.
- Gli errori di ingresso e uscita sono segnalati in due modi:
  - in qualche caso l'errore è segnalato *cambiando lo stato* dello stream,
  - più frequentemente l'errore è segnalato lanciando *un'eccezione* di tipo `IOException`

Ingresso da file binari

- Per leggere un file in formato binario si utilizza un oggetto della classe `FileInputStream`
- Un file di cui si conosca il nome si apre invocando il costruttore:
  - `FileInputStream in = new FileInputStream(nome);`
- Per leggere **un** byte si utilizza il metodo `read()` che restituisce un `int` con il byte letto inserito negli 8 bit meno significativi
- All'**EOF** il metodo `read()` restituisce il valore `-1`
- Al termine delle operazioni di ingresso il file deve essere chiuso invocando il metodo `close()`

Scrittura su file binari

- Per scrivere un file in formato binario si utilizza un oggetto della classe `FileOutputStream`
- Il file in uscita viene aperto invocando il costruttore:
  `FileOutputStream out = new FileOutputStream(nome);`
  che crea il file individuato dalla stringa nome se non esiste o riscrive il file se esiste
- Per scrivere **un** byte si utilizza il metodo `write(int c)`.
  Il metodo usa un solo parametro `int` del quale verranno trasferiti in usciti gli 8 bit meno significativi
- Al termine delle operazioni di uscita il file deve essere chiuso, per assicurarsi che il contenuto dei buffer di sistema sia effettivamente trasferito sul file, invocando il metodo `close()`

Esempio: copia di file

```
import java.io.*;
public class CopiaBin {
  public static void main (String[] arg) throws IOException {
    int c = 0;
    FileInputStream in = new FileInputStream(arg[0]);
    FileOutputStream out = new FileOutputStream(arg[1]);
    while ((c = in.read()) != -1) {
      out.write(c);
    }
    out.close();
    in.close();
  }
}
```

NB: questo programma copia un file nell'altro byte per byte, qualunque significato e codifica sia attribuita ai byte. Quindi funziona anche per file di caratteri.

Leggete la documentazione!

https://docs.oracle.com/en/java/javase/11/
docs/api/java.base/java/io/FileInputStream.html

https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html

Most Visited Getting Started

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.io

Class FileInputStream

java.lang.Object
  java.io.InputStream
    java.io.FileInputStream

All Implemented Interfaces:

Closeable, AutoCloseable

```
public class FileInputStream
extends InputStream
```

A `FileInputStream` obtains input bytes from a file in a file system. What files are available depends on the host environment.

`FileInputStream` is meant for reading streams of raw bytes such as image data. For reading streams of characters, consider using `FileReader`.

Lettura di un file di testo: la classe `FileReader`

- Per implementare i reader, il package `java.io` fornisce la classe `FileReader`

  ```
  public class FileReader extends InputStreamReader
  ```

  - Convenience class for reading character files.
  - `FileReader` is meant for reading streams of characters. For reading streams of raw bytes, consider using a `FileInputStream`.

Lettura di un file di testo: la classe `FileReader`

- Costruttore (uno dei tanti):

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

  - Creates a new `FileReader`, given the name of the file to read from.
  - Parameters: `fileName` - the name of the file to read from
  - Throws: `FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

Metodi della classe `FileReader`

`public int read() throws IOException`

Legge un singolo carattere dallo stream che esegue il metodo e ne restituisce il codice sotto forma di intero compreso tra 0 e 65535 (i numeri interi rappresentabili con 16 bit). Se si è raggiunta la fine del file, il metodo restituisce –1.

`public int read(char[] buf) throws IOException`

Legge dallo stream che esegue il metodo una sequenza di caratteri della stessa lunghezza dell'array specificato come argomento e li memorizza nelle posizioni successive dell'array.

`public void close() throws IOException`

Chiude lo stream che esegue il metodo. Dev'essere invocato per rilasciare la risorsa, cioè il file cui è collegato lo stream. Una volta che lo stream è stato chiuso, eventuali invocazioni di un metodo **`read`** (o di altri metodi che eseguano operazioni sullo stream) sollevano l'eccezione **`IOException`**.

Esempio: visualizzazione di un file di caratteri

```
import java.io.*;

public class ViewCharFile {
  public static void main(String[] args) throws IOException {
    FileReader frd = new FileReader(args[0]);
    int i=0;
    while ((i = frd.read()) != -1)
      System.out.print((char)i);
    frd.close();
  }
}
```

Esempio: scrittura su file di testo

```
import java.io.*;
public class FileWriterWriteIntExample {
  public static void main(String[] args) {
    FileWriter fileWriter = null;
    try {
      fileWriter = new FileWriter("file.txt");
      fileWriter.write('C');
      fileWriter.write('i');
      fileWriter.write('a');
      fileWriter.write('o');
    } catch (Exception e) { e.printStackTrace(); }
      finally {
        try {
          if(fileWriter != null) {
            fileWriter.flush();
            fileWriter.close();
          }
        } catch (IOException e) { e.printStackTrace(); }
}}}
```

Leggere e scrivere una riga alla volta

- Leggere e scrivere un carattere alla volta è scomodo.
- I testi di caratteri sono generalmente organizzati in righe.
- Vogliamo leggere e scrivere una riga alla volta
  - Più semplice ed efficiente

Leggere e scrivere una riga alla volta

- Un file di testo è un file composto da caratteri
- Il file è diviso in righe cioè in sequenze di caratteri terminati dalla stringa EndOfLine (fine riga)
- EndOfLine è codificata diversamente nei vari sistemi operativi
  - “\r\n” Windows
  - “\n” Unix – Linux
  - “\r” MAC OS
- Dove:
  - \n = new line o linefeed
  - \r = carriage return

Classe BufferedReader

- Costruttore:

```
public BufferedReader(Reader in)
```

Prende un **`Reader`** e restituisce un **`BufferedReader`**

- Metodi:

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

- Legge una riga di testo.
  - Una linea è considerata conclusa dai caratteri \n (linefeed), \r (carriage return) o da un carriage return seguito da un linefeed.
- Se invece al momento dell'invocazione è stata raggiunta la fine dello stream, il metodo restituisce **`null`**.
- Altrimenti, il metodo restituisce la stringa contenente la linea di caratteri letta (senza il carattere di terminazione della riga).

Esempio: visualizzazione di un file di caratteri

```
import java.io.*;

public class ViewCharFile2 {
  public static void main(String[] args) throws IOException {
    FileReader frd = new FileReader(args[0]);
    BufferedReader bfr = new BufferedReader(frd);
    String str;
    while ((str = bfr.readLine()) != null)
      System.out.println(str);
    bfr.close();
    frd.close();
  }
}
```

Classe PrintWriter

- È l'analogo di BufferedReader per scrivere
- Costruttore:

```
public PrintWriter(Writer out)
```

- Metodi

  `print(arg)` e `println(arg)`

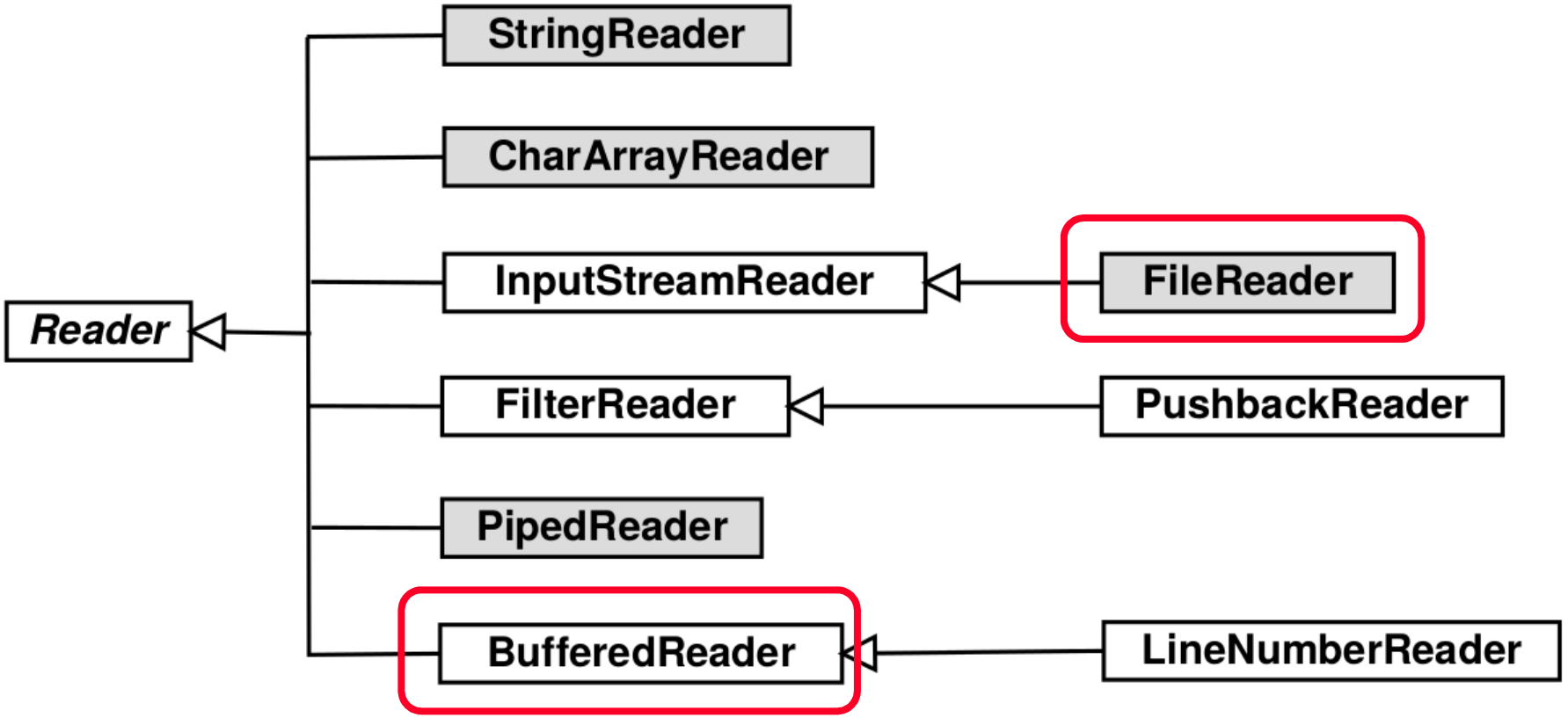
  - convertono l'argomento in una stringa e la trasferiscono in uscita (`println` con l'aggiunta di un `EndOfLine`)

Esempio: copia di file di caratteri

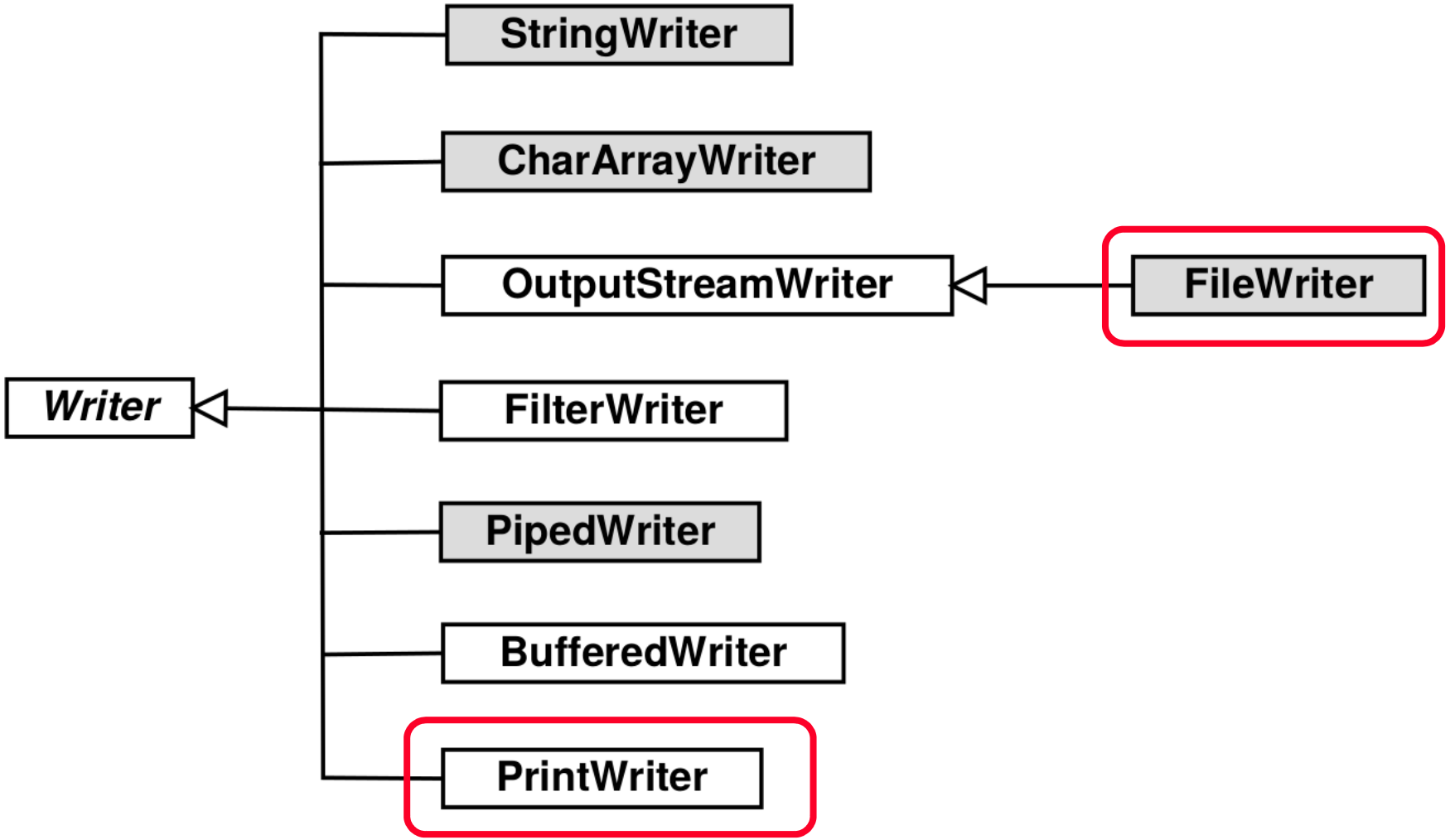
```
import java.io.*;

public class CopyChar {
  public static void main (String[] arg) throws IOException {
    String str = "";
    FileReader frd = new FileReader(arg[0]);
    BufferedReader bfr = new BufferedReader(frd);
    FileWriter fwr = new FileWriter(arg[1]);
    PrintWriter bwr = new PrintWriter(fwr);
    while ((str = bfr.readLine()) != null)
      bwr.println(str);
    bfr.close();
    frd.close();
    bwr.close();
    fwr.close();
  }
}
```

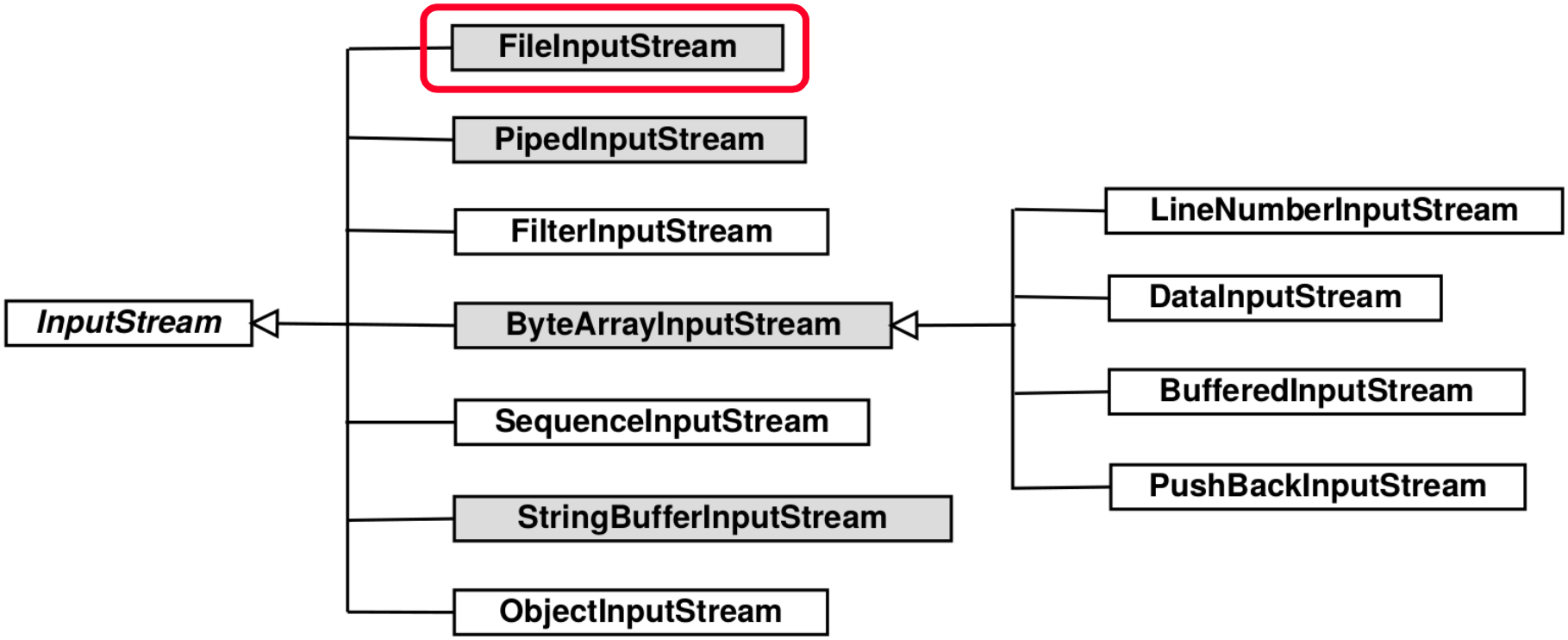
Classi che gestiscono flussi (stream) di caratteri in ingresso



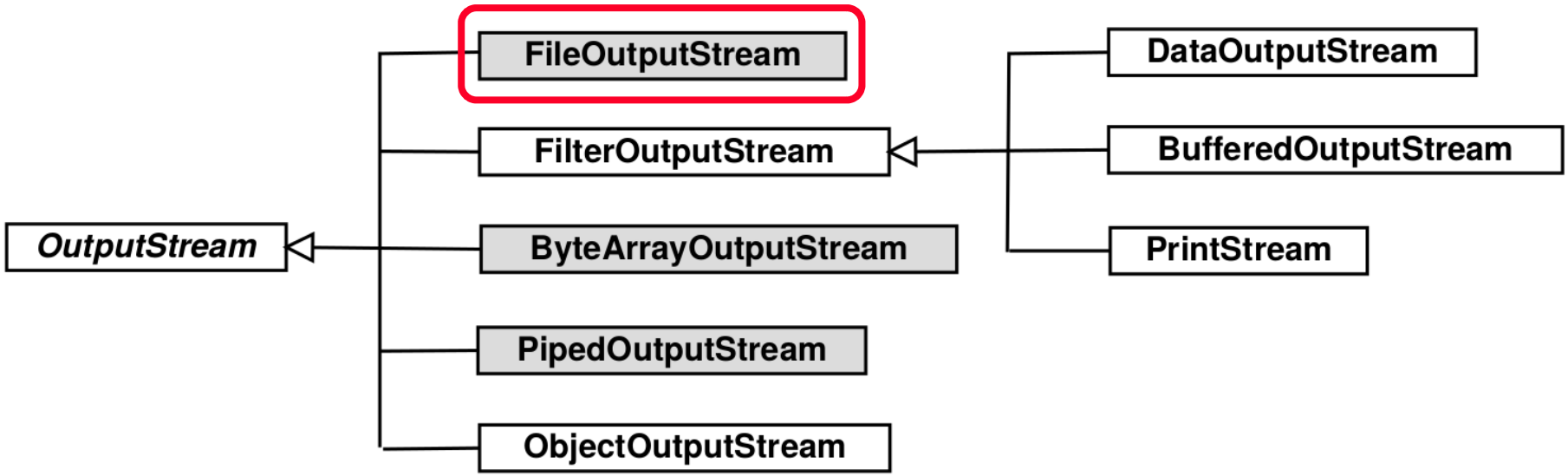
Classi che gestiscono flussi (stream) di caratteri in uscita



Classi che gestiscono flussi (stream) di byte in ingresso



Classi che gestiscono flussi (stream) di byte in uscita



Scrivere dati primitivi su stream di byte

- La classe **`PrintStream`** (estensione di **`OutputStream`**) consente di convertire dati primitivi in sequenze di byte.
- Costruttore

```
public PrintStream(OutputStream out)
```

  - Crea un print stream verso l'output stream specificato.
- Metodi

```
void print(boolean b)
void print(int i)
void print(long l)
void print(float f)
void print(double d)
void print(char c)
void print(char[] s)
void print(String s)
void print(Object obj)
```

- Per ciascuno di questi esiste il corrispondente **`println`**

Esempio

```
File file = new File(nomefile);
FileOutputStream fos = new FileOutputStream(file);
PrintStream ps = new PrintStream(fos);
ps.println("Provo valori di vario tipo");
ps.println(100);
ps.println(3/4.0);
ps.println(true && false);
```

Collegamento con file

- I sistemi operativi utilizzano pathname per attribuire un nome ai file e alle directory
- La classe **`File`** fornisce una rappresentazione astratta ed *indipendente dal sistema* dei pathname gerarchici
  - Gli oggetti di tipo **`File`** consentono di creare un collegamento con il file fisico trattando nomi di file e directory (pathname gerarchici) in modo astratto e indipendente dal sistema

Gli stream (flussi) di I/O Standard

- I riferimenti a standard input, standard output e standard error sono disponibili mediante attributi statici della classe `java.lang.System`

```
public static final InputStream in
```

  - Standard input (tastiera)

```
public static final PrintStream out
```

  - Standard output (monitor)

```
public static final PrintStream err
```

  - Standard error (monitor)

Dump.java
esamina il contenuto di un file binario

```
import java.io.*;

public class Dump {
  public static void main (String[] arg) throws IOException {
    FileInputStream in = new FileInputStream(arg[0]);
    int n = Integer.parseInt(arg[1]);
    int c = 0;
    int i = 0;
    String str = "    ";
    System.out.print(str);
    String car = "    ";
```

Dump.java
esamina il contenuto di un file binario

```
while (((c = in.read()) != -1) && (i < n)) {
  str = Integer.toString(c);
  while (str.length() < 4) { str = " " + str; }
  System.out.print(str);
  i++;
  if (c < 31)
    car += '.'; // i car. di controllo diventano un punto
  else if (c < 128)
    car += (char)c; // caratteri ASCII
  else
    car += '*';     // caratteri > 127 diventano *
  if (i % 10 == 0) {    // ogni 10 caratteri un fineriga
    System.out.println(car);
    car = "    ";
    str = Integer.toString(i);
    while (str.length() < 4) { str = " " + str; }
    System.out.print(str);
  }
}
in.close();
} }
```

Esempi di applicazione

- Applicazione ad un file binario:

```
java Dump Dump.class 100
```

```
   202 254 186 190   0   0   0  55   0  94     ****...7.^
 10  10   0  18   0  32   7   0  33  10   0     .... ..!..
 20   2   0  34  10   0  35   0  36   8   0     ..".."#.$..
 30  37   9   0  38   0  39  10   0  40   0     %..&.'..(.
 40  41  10   0   2   0  42  10   0  35   0     ).....*..#.
 50  43  10   0  27   0  44  18   0   0   0     +....,....
 60  48  18   0   1   0  48  18   0   2   0     0....0....
 70  51  18   0   3   0  48  10   0  40   0     3....0..(.
 80  53  10   0   2   0  54   7   0  55   7     5....6..7.
 90   0  56   1   0   6  60 105 110 105 116     .8...<init
100
```

Esempi di applicazione

- Applicazione ad un file di testo

```
java Dump Dump.java 100
```

```
   105 109 112 111 114 116  32 106  97 118    import jav
10  97  46 105 111  46  42  59  10  10 112    a.io.*;..p
20 117  98 108 105  99  32  99 108  97 115    ublic clas
30 115  32  68 117 109 112  32 123  10  32    s Dump {.
40  32 112 117  98 108 105  99  32 115 116     public st
50  97 116 105  99  32 118 111 105 100  32    atic void
60 109  97 105 110  32  40  83 116 114 105    main (Stri
70 110 103  91  93  32  97 114 103  41  32    ng[] arg)
80 116 104 114 111 119 115  32  73  79  69    throws IOE
90 120  99 101 112 116 105 111 110  32 123    xception {
100
```