DIETI UNINA UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE
Scuola Politecnica e delle Scienze di Base
Università degli Studi di Napoli Federico II

Corso di
Ingegneria del Software

Gestione I/O in Java

Sommario

- Input/Output e Filtri in Java

Riferimenti

Thinking in Java – 3° edizione:
Capitoli 12

Input/Output in Java

::.. Il concetto di Stream (1/2)

Java gestisce le operazioni di I/O servendosi dell'astrazione di flusso o stream e di due classi fondamentali: OutputStream e InputStream.

Stream: sequenza di byte che viaggiano da un'origine a una destinazione lungo un canale monodirezionale:

OutputStream: punto di accesso,

InputStream: punto di arrivo.

I dati scritti nell' output stream possono essere letti dallo input stream.

::.. Il concetto di Stream (2/2)

Java offre molteplici classi per la gestione dell'IO (che compongono la libreria java.io), tutte basate sul concetto fondamentale di flusso o stream.

OutputStream, classe astratta. Rappresenta la sorgente di un flusso con metodi write() per scrivere sul flusso.

::.. Il concetto di Stream (2/2)

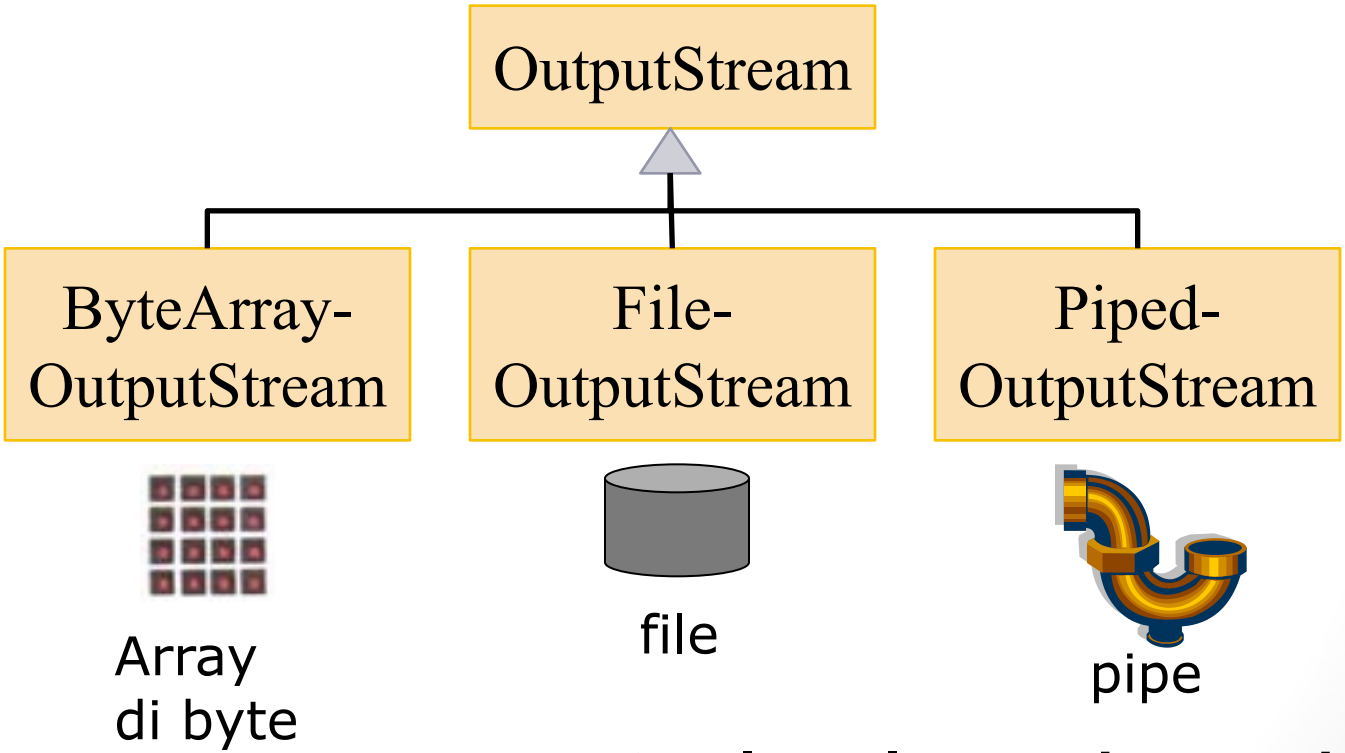
Java offre molteplici classi per la gestione dell'IO (che compongono la libreria java.io), tutte basate sul concetto fondamentale di flusso o stream.

OutputStream, classe astratta. Rappresenta la sorgente di un flusso con metodi write() per scrivere sul flusso.

InputStream, classe astratta. Rappresenta la destinazione di un flusso con metodi read() per leggere dal flusso.

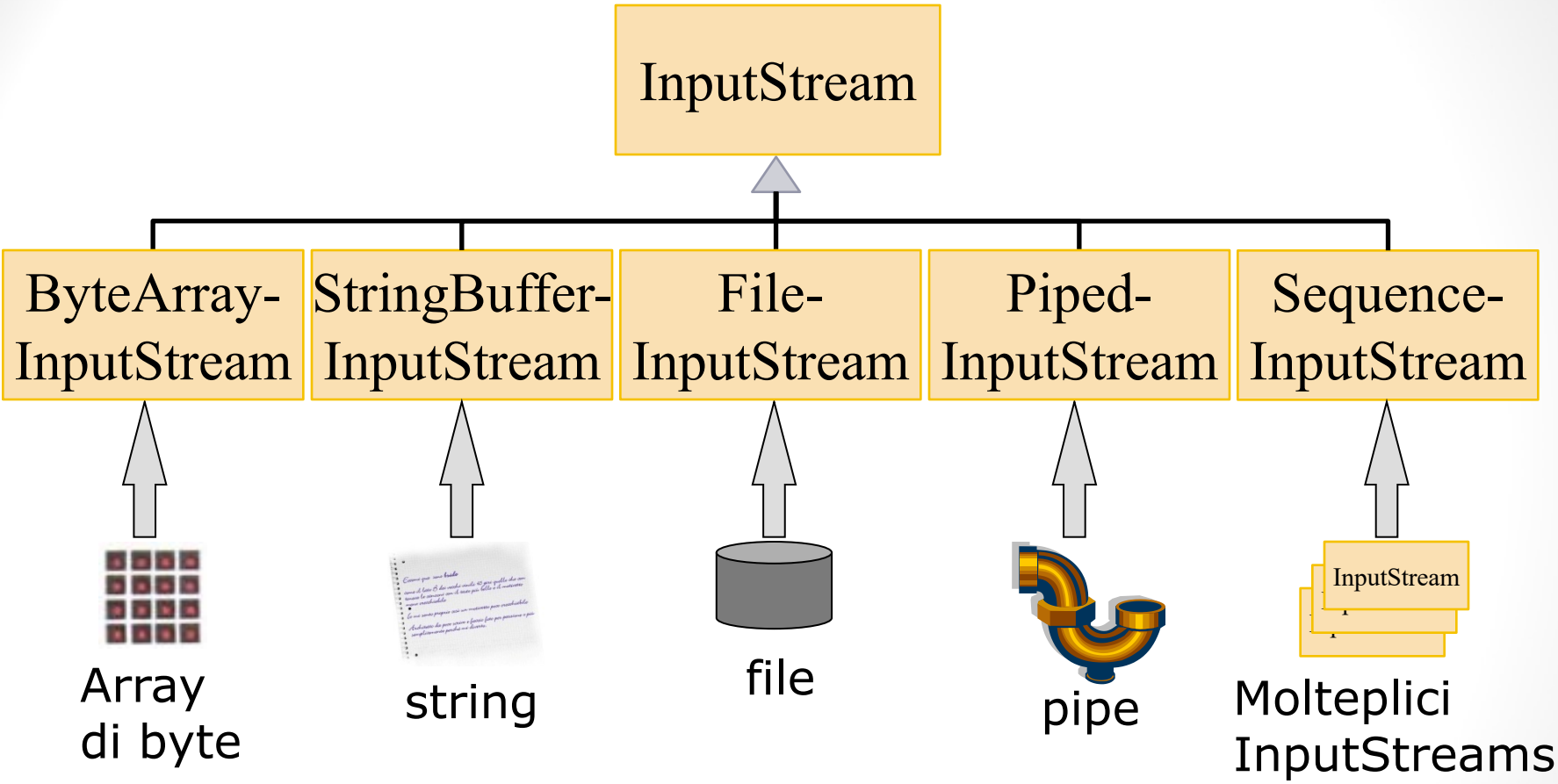
::.. I/O in Java 1.0 (1/4)

Java offre molteplici classi per la gestione dell'IO (che compongono la libreria java.io), tutte basate sul concetto fondamentale di flusso o stream.



`OutputStream` rappresenta la classe base della gerarchia di classi delle destinazioni dei dati nel flusso.

::.. I/O in Java 1.0 (2/4)



`InputStream` è la base della gerarchia di classi che decidono da dove acquisire i dati attraverso il flusso.

::.. I/O in Java 1.0 (3/4)

Consideriamo come esempio di gestione dell'IO in Java un'applicazione che:

1.scrive su un `ByteArrayOutputStream` una stringa di caratteri;

2.Successivamente acquisisce la stringa attraverso un `ByteArrayInputStream`.

::.. I/O in Java 1.0 (4/4)

```
public class ByteArrayIOApp{
  public static void main(String args[]) throws IOException {
    ByteArrayOutputStream outStream = new ByteArrayOutputStream();
    String s = "Questo è un test …";
    for(int i = 0; i < s.length(); i++)
      outStream.write(s.charAt(i));
    System.out.println("Flusso di output "+outStream);
    System.out.println("Dimensione: "+outStream.size());
    ByteArrayInputStream inStream;
    inStream = new ByteArrayInputStream(outStream.toByteArray());
    int inBytes = inStream.available();
    byte inBuf[] = new byte[inBytes];
    int bytesRead = inStream.read(inBuf, 0, inBytes);
    System.out.println(bytesRead+" byte letti");
    System.out.println("Sono: "+new String(inBuf)); }
}
```

::.. I/O in Java 1.0 (4/4)

```
public class ByteArrayIOApp{
  public static void main(String args[]) throws IOException {
    ByteArrayOutputStream outStream = new By        rOutputStream();
    String s
    for(int
      outStr
    System.out.println("Flusso di output "+outStream);
    System.out.println("Dimensione: "+outStream.size());
    ByteArrayInputStream inStream;
    inStream = new ByteArrayInputStream(outStream.toByteArray());
    int inBytes = inStream.available();
    byte inBuf[] = new byte[inBytes];
    int bytesRead = inStream.read(inBuf, 0, inBytes);
    System.out.println(bytesRead+" byte letti");
    System.out.println("Sono: "+new String(inBuf)); }
}
```

L'invocazione di metodi che realizzano operazioni di I/O può sollevare eccezioni di tipo IOException, che vanno gestite.

::.. I/O in Java 1.0 (4/4)

```
public class ByteArrayIOApp{
  public static void main(String args[]) throws IOException {
    ByteArrayOutputStream outStream = new ByteArrayOutputStream();
    String s = "Questo è un test …";
    for(int i = 0; i < s.length(); i++)
      outStream.write(s.charAt(i));
    System.out.println("Flusso di output
```

Si istanzia un oggetto di tipo ByteArrayOutputStream e la stringa da utilizzare. Successivamente si realizza l'operazione di scrittura carattere per carattere.

```
    byte inBuf[] = new byte[inBytes];
    int bytesRead = inStream.read(inBuf, 0, inBytes);
    System.out.println(bytesRead+" byte letti");
    System.out.println("Sono: "+new String(inBuf)); }
}
```

::.. I/O in Java 1.0 (4/4)

```
public class ByteArrayIOApp{
  public static void main(String args[]) throws IOException {
    ByteArrayOutputStream outStream = new ByteArrayOutputStream();
    String s = "Questo è un test …";
    for(int i = 0; i < s.length(); i++)
      outStream.write(s.charAt(i));
    System.out.println("Flusso di output "+outStream);
    System.out.println("Dimensione: "+outStream.size());
    ByteArrayInputStream inStream;
    inStream = new ByteArrayInputStream(outStream.toByteArray());
    int inBytes = inStream.available();
}
```

Si istanzia un oggetto di tipo ByteArrayInputStream, trasmettendo come parametro di ingresso al costruttore l'array di outStream, che rappresenta il buffer da cui estrarre i byte.

::.. I/O in Java 1.0 (4/4)

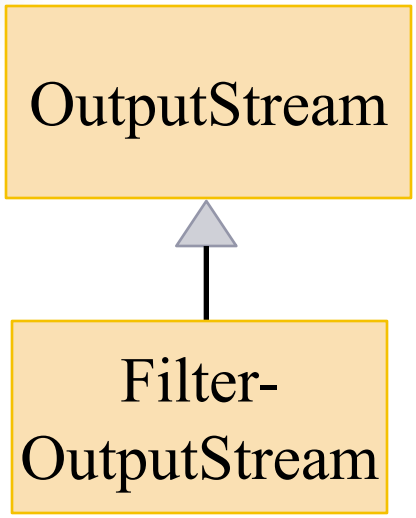
```
public class ByteArrayIOApp{
  public static void main(String args[]) throws IOException {
    ByteArrayOutputStream outStream = new ByteArrayOutputStream();
    String s = "Questo è un test …";
    ByteArrayInputStream inStream;
    inStream = new ByteArrayInputStream(      eam.toByteArray());
    int inBytes = inStream.available();
    byte inBuf[] = new byte[inBytes];
    int bytesRead = inStream.read(inBuf, 0, inBytes);
    System.out.println(bytesRead+" byte letti");
    System.out.println("Sono: "+new String(inBuf)); }
}
```

Si utilizza available() per conoscere quanti caratteri si devono leggere e dimensionare opportunamente un array di byte, che verrà usato nell'operazione di read.

::.. I/O filtrato: generalità (1/2)

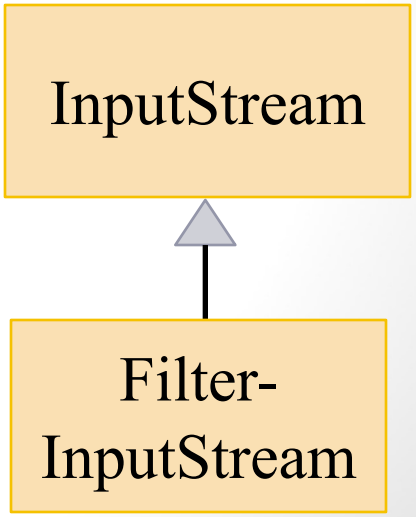
Tra le classi derivate da OutputStream e InputStream troviamo due classi particolari che consentono di implementare la tecnica dell' "IO filtrato": è possibile aggiungere dei filtri ai flussi, per adattarli a specifiche esigenze di programmazione.

::.. I/O filtrato: generalità (2/2)

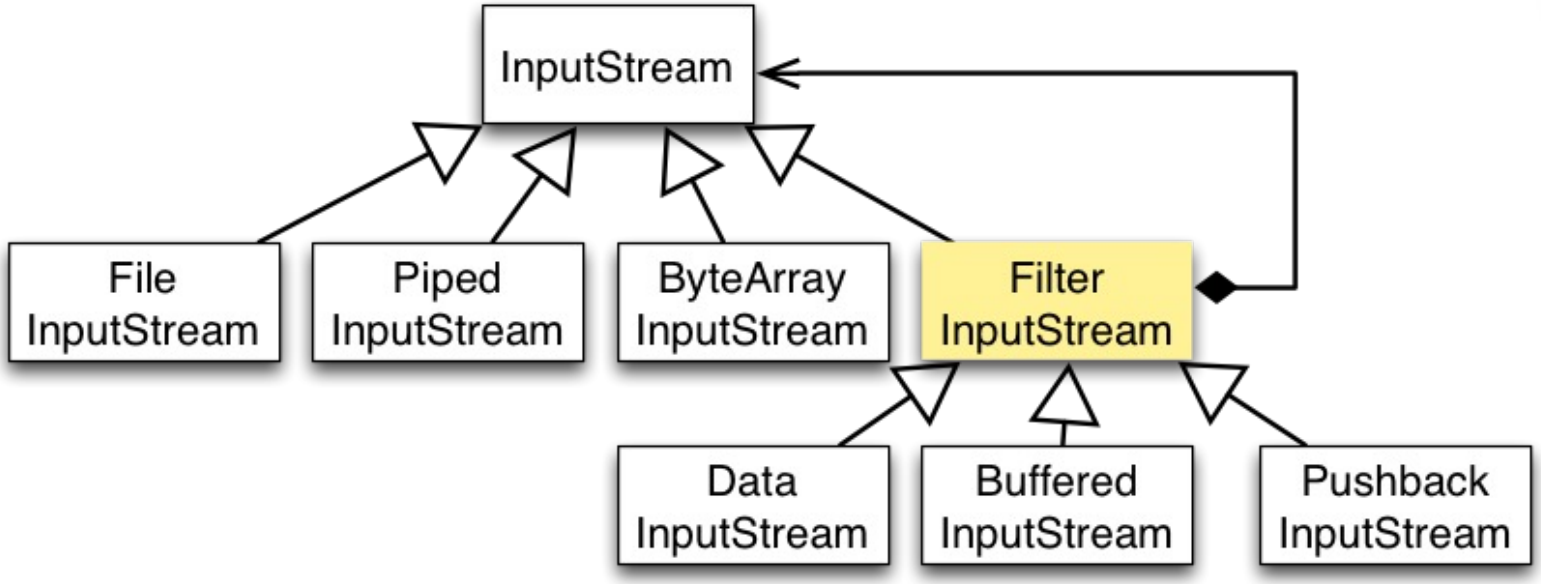


- **FilterOutputStream: classe astratta base di tutte le classi di flussi di output filtrati,**
  - **offre la possibilità di creare un flusso a partire da un altro.**
  - **È possibile definire filtri multipli concatenati usando più livelli di annidamento.**

FilterInputStream è complementare a FilterOutputStream per i flussi di Input filtrati.



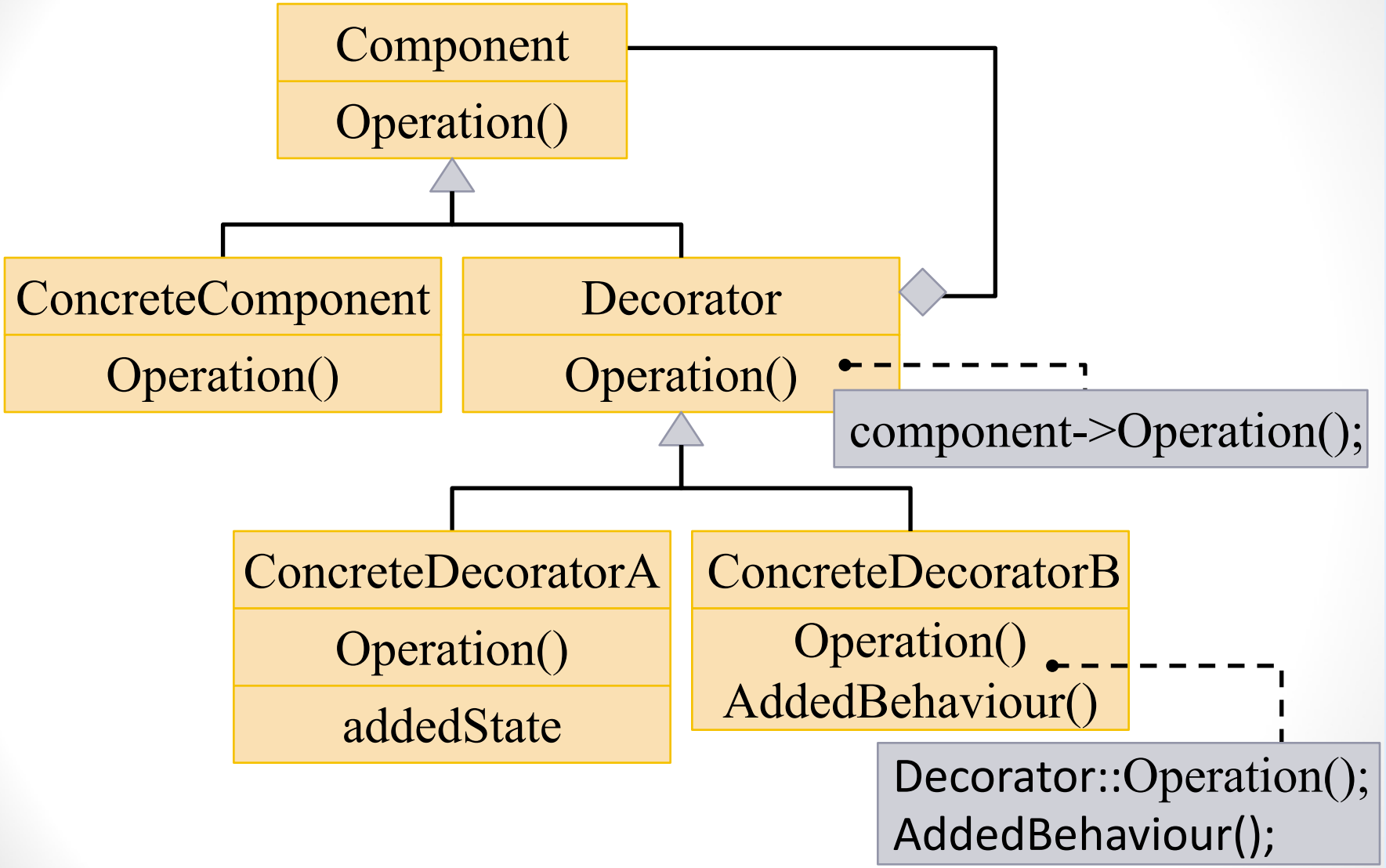
::.. Decorator per l'IO filtrato



::.. Pattern Decorator

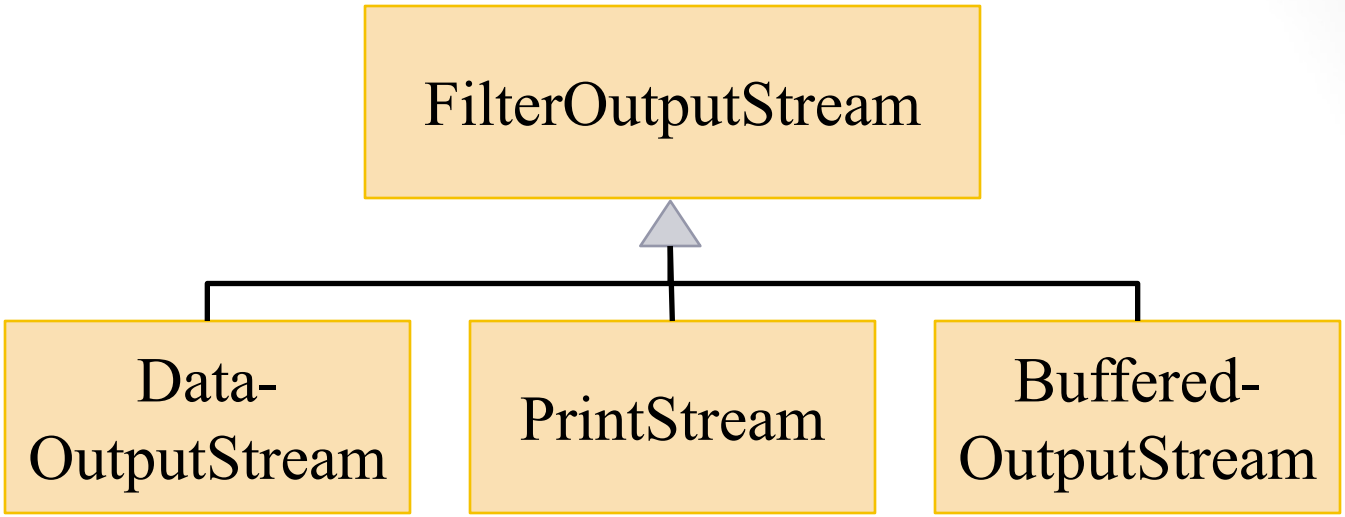
- **I filtri di flussi sono realizzati secondo un pattern che prende il nome di decorator: uso di oggetti annidati per aggiungere in modo dinamico e trasparente responsabilità a singoli oggetti.**
- **Il pattern decorator è molto simile alla composizione: la classe di livello superiore ha come attributo un oggetto della classe di livello inferiore.**
- **La differenza è che la classe di livello superiore deve avere la stessa interfaccia della classe dell'oggetto contenuto, così da poter far uso delle due classi in maniere trasparente al programmatore.**

::.. Decorator per l'IO filtrato



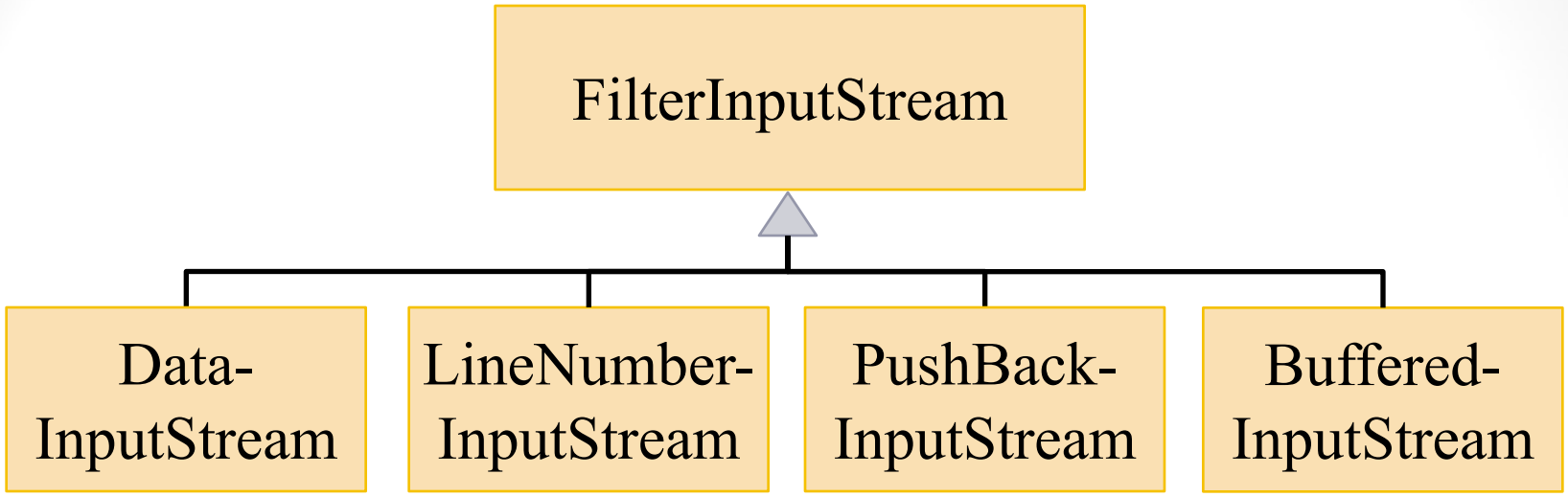
* E. Gamma et al. "Design Pattern"

::.. I/O filtrato (1/5)



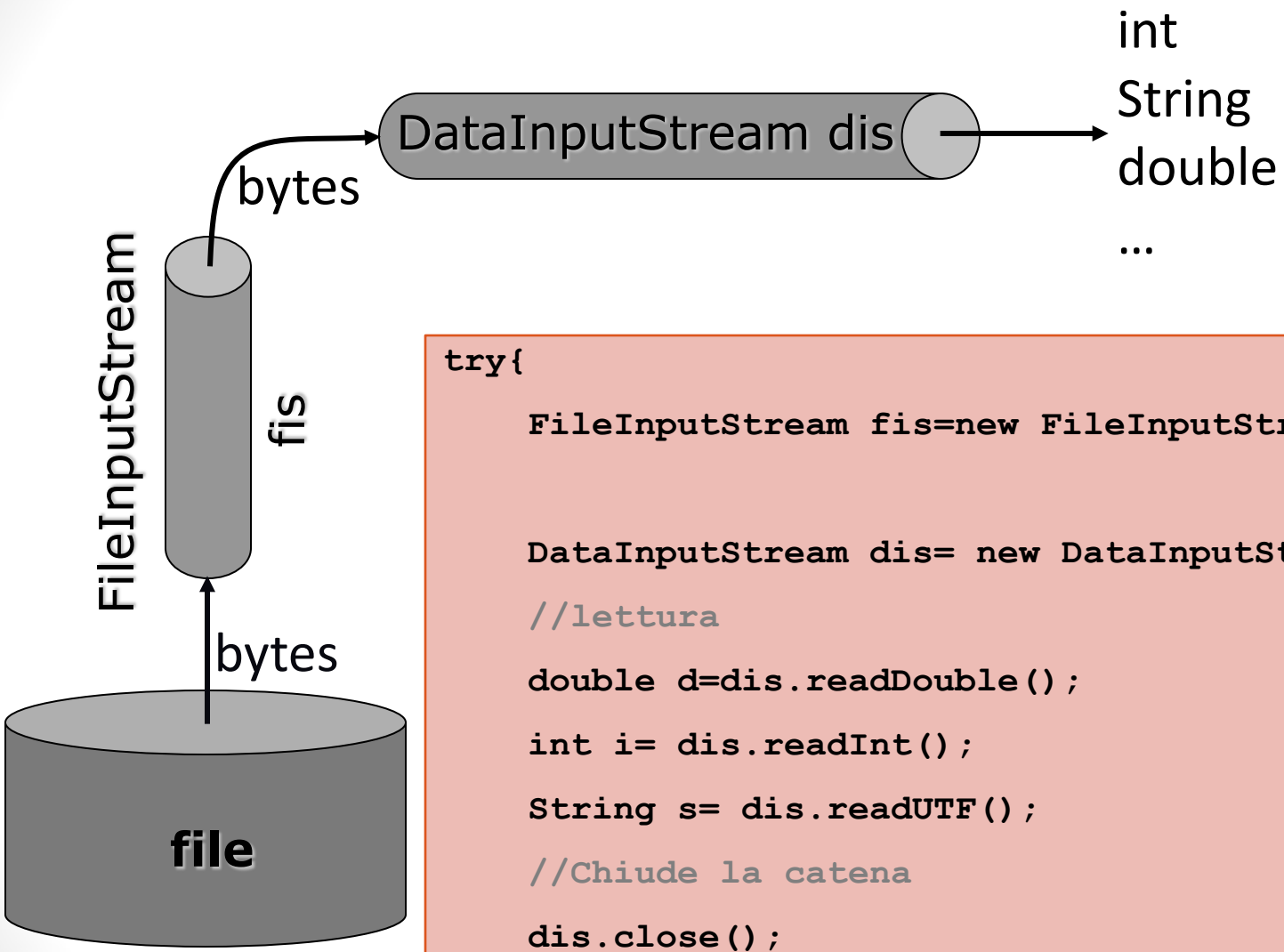
- **DataOutputStream consente di inserire istanze di tipi primitivi (ad es. int) e String in uno stream di Output.**
- **PrintStream produce un output formattato.**
- **BufferedOutputStream viene usato per prevenire una scrittura fisica ogni volta che si scrive su un flusso, i dati in scrittura sono posti su un buffer e l'utente sollecita una scrittura sul flusso con flush().**

::.. I/O filtrato (2/5)



- **DataInputStream e BufferedInputStream, complementari di DataOutputStream e BufferedOutputStream per l' Input.**
- **LineNumberInputStream registra i numeri di riga nello stream di Input.**
- **PushBackInputStream dispone di un buffer Push-Back della dimensione di un byte, cosicché l'utente può operare un inserimento nel buffer dell'ultimo byte letto.**

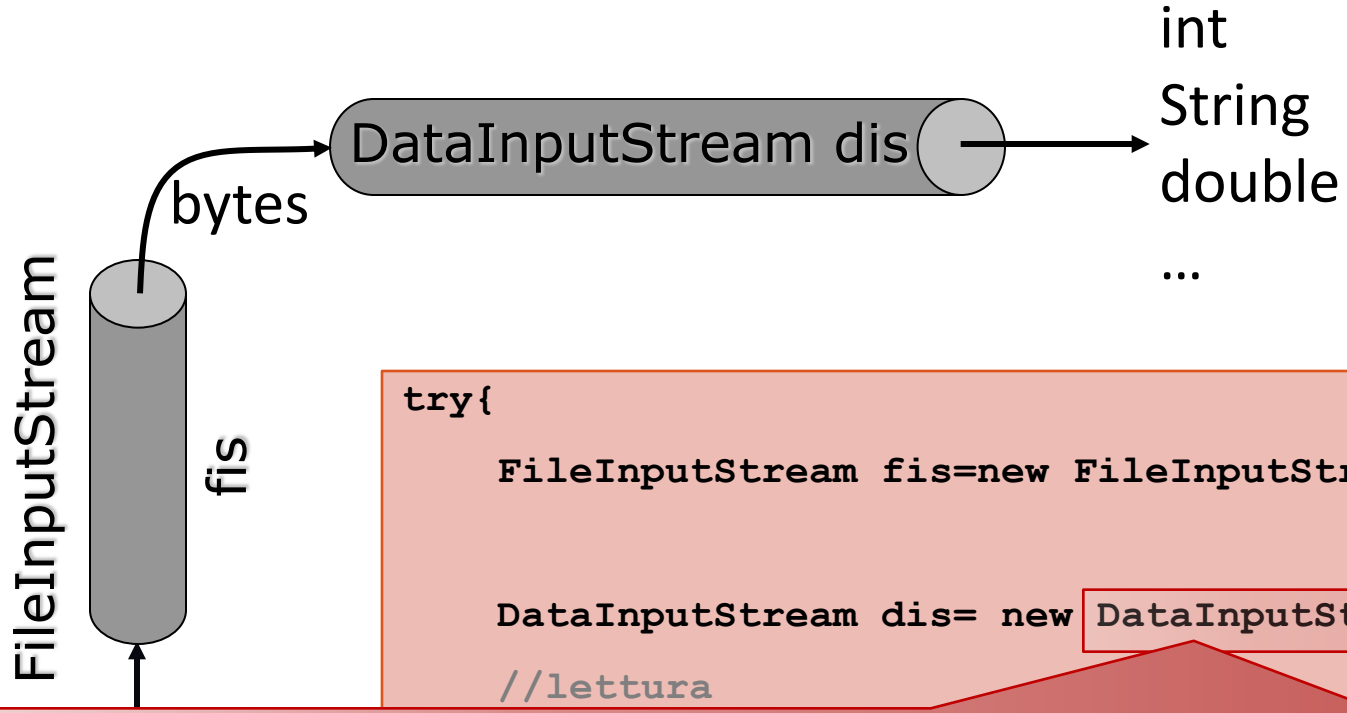
::.. I/O filtrato (3/5)




```
try{
    FileInputStream fis=new FileInputStream("text.txt");

    DataInputStream dis= new DataInputStream(fis);
    //lettura
    double d=dis.readDouble();
    int i= dis.readInt();
    String s= dis.readUTF();
    //Chiude la catena
    dis.close();
    fis.close();
}catch (IOExceptions e){}
```

::.. I/O filtrato (3/5)



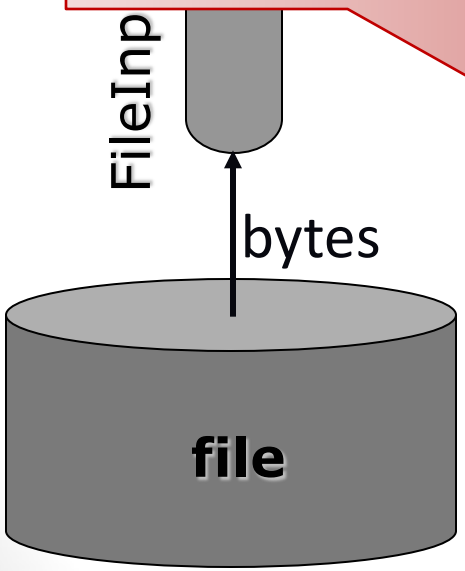

```
try{
    FileInputStream fis=new FileInputStream("text.txt");

    DataInputStream dis= new DataInputStream(fis);
    //lettura
}catch (IOExceptions e){}
```

Il costruttore del filtro richiede in ingresso il riferimento di un altro flusso o filtro a cui connettersi. Questo mette in luce la natura dei filtri: sebbene derivate da InputStream o OutputStream, le classi-filtro non sono dei veri e propri stream, infatti hanno come end-point degli stream anziché dei device.

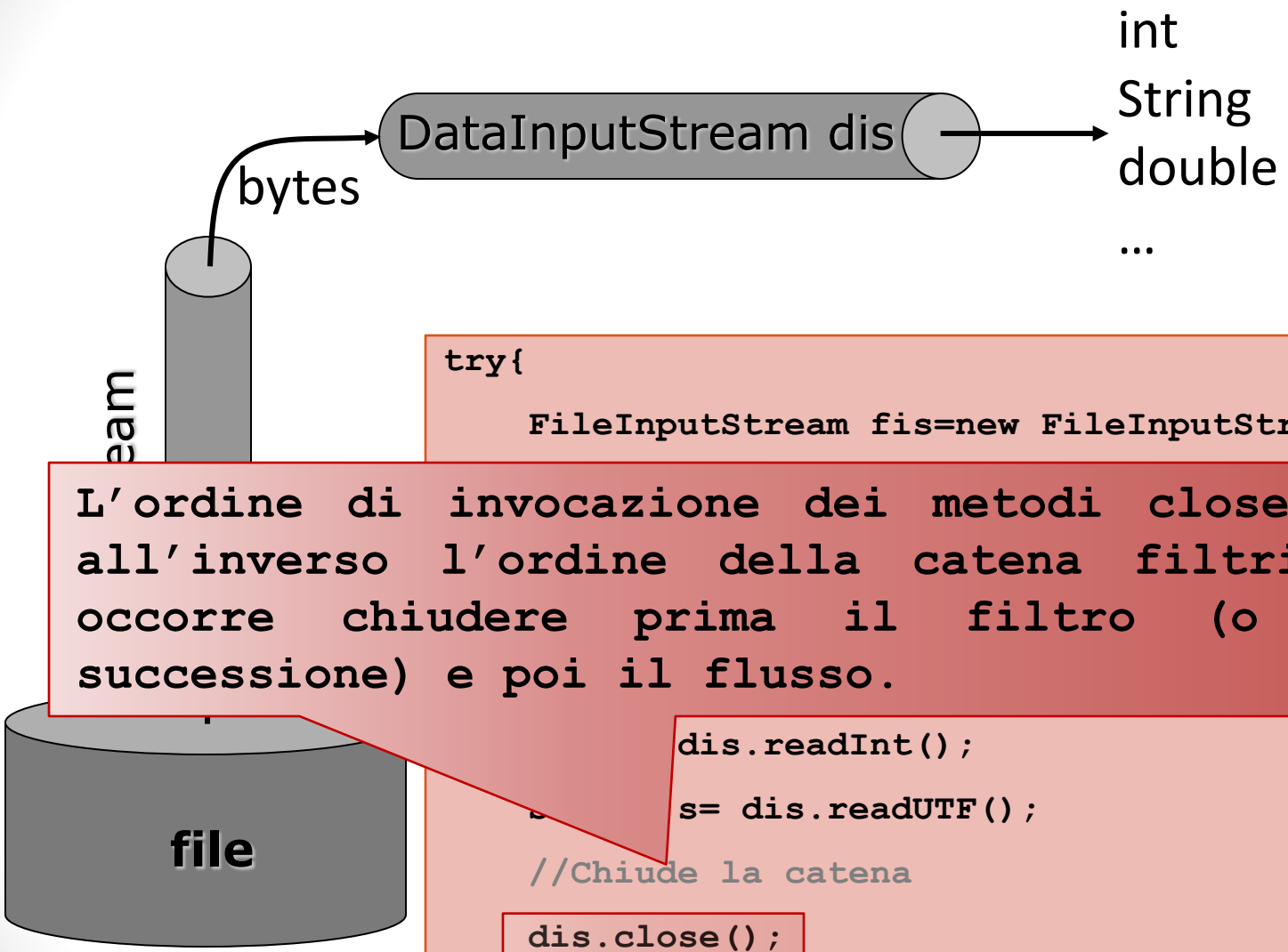
::.. I/O filtrato (3/5)

Posso fare operazioni di lettura senza l'obbligo di usare array di tipo byte come nel caso precedente:
byte inBuf[] = new byte[inBytes];
int bytesRead = inStream.read(inBuf, 0, inBytes);
Ma il filtro consente di ottenere un accesso tipizzato al canale del flusso.




```
                tStream dis= new DataInputStream(fis);
        //le      a
        double d=dis.readDouble();
        int i= dis.readInt();
        String s= dis.readUTF();
        //Chiude la catena
        dis.close();
        fis.close();
}catch (IOExceptions e){}
```

::.. I/O filtrato (3/5)




```
try{
    FileInputStream fis=new FileInputStream("text.txt");
    dis.readInt();
    s= dis.readUTF();
    //Chiude la catena
    dis.close();
    fis.close();
}catch (IOExceptions e){}
```

L'ordine di invocazione dei metodi close() segue all'inverso l'ordine della catena filtri/flussi: occorre chiudere prima il filtro (o la sua successione) e poi il flusso.

::.. I/O filtrato (4/5)

Analogamente, possiamo usare nel seguente modo la combinazione del filtro DataOutputStream e del flusso FileOutputStream:

```
try{
    FileOutputStream fis = new FileOutputStream("text.txt");
    DataOutputStream dis = new DataOutputStream(fis);
    //Scrittura
    dis.writeDouble(1.3456);
    dis.writeInt(44);
    dis.writeUTF("ciao mondo");
    //Chiude la catena
    dis.close(); //chiude prima il filtro
    fis.close();
}catch (IOExceptions e){}
```

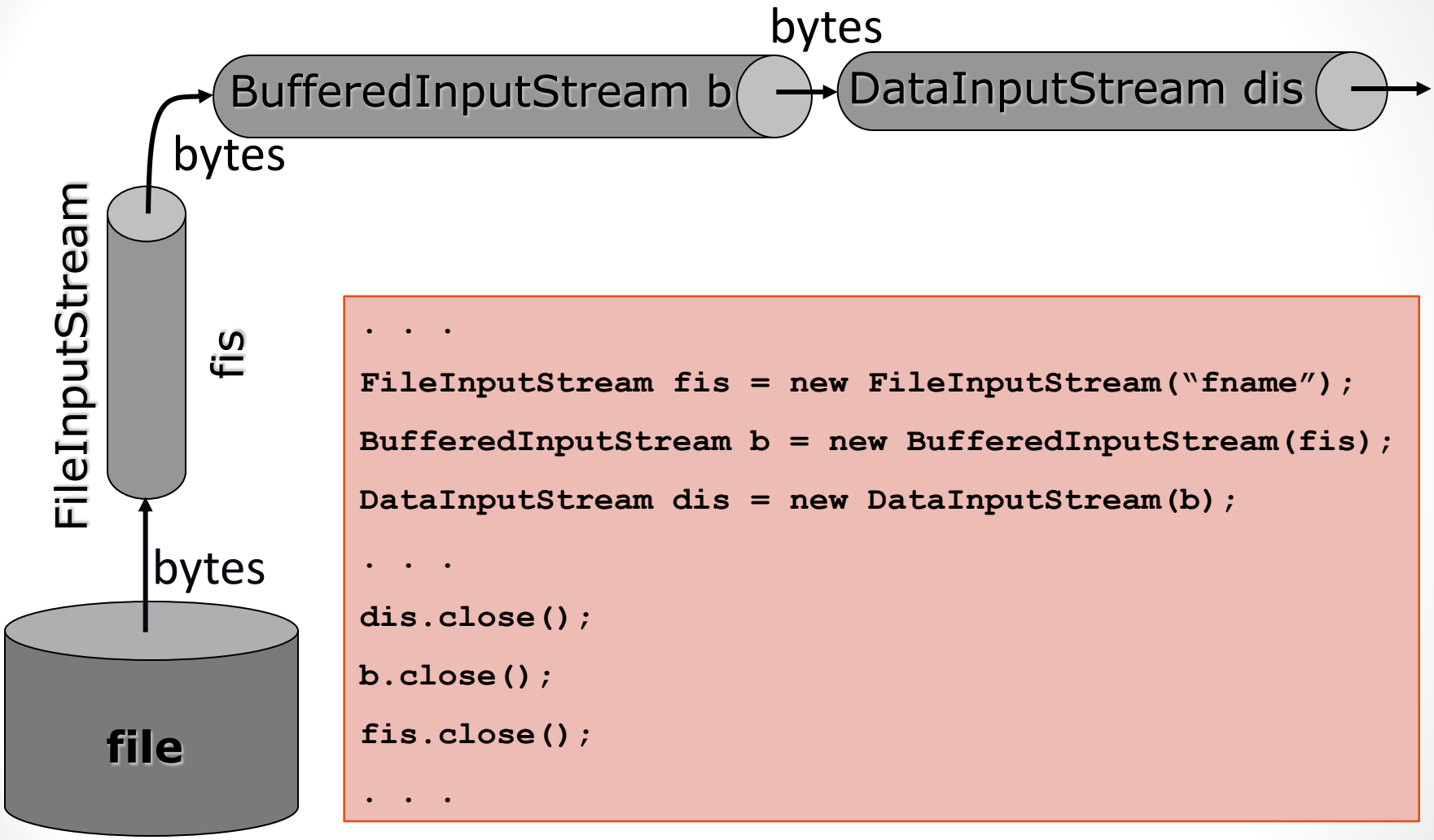
::.. I/O filtrato (4/5)

Analogamente, possiamo usare nel seguente modo la combinazione del filtro DataOutputStream e del flusso

Notiamo come in scrittura non siamo limitati solo a scrivere un byte o un array di byte come nel caso precedente:
outStream.write(s.charAt(i));
possiamo scrivere valori di tipi primitivi e String.

```
    //Scrittura
    dis.writeDouble(1.3456);
    dis.writeInt(44);
    dis.writeUTF("ciao mondo");
    //Chiude la catena
    dis.close(); //chiude prima il filtro
    fis.close();
}catch (IOExceptions e){}
```

::.. I/O filtrato (5/5)




```
. . .
FileInputStream fis = new FileInputStream("fname");
BufferedInputStream b = new BufferedInputStream(fis);
DataInputStream dis = new DataInputStream(b);
. . .
dis.close();
b.close();
fis.close();
. . .
```

::.. I/O da stdout/stdin

- Per stampare a schermo si possono utilizzare i metodi dell'oggetto statico **System.out** collegato allo stdout

```
System.out.print(…);
System.out.println(…);
System.out.format("Sintassi printf %d", …);
```

- Per leggere **stringhe** da stdin occorre usare un oggetto BufferedReader collegato su **System.in**

```
BufferedReader bufferedReader = new BufferedReader(
                                new InputStreamReader(System.in));
String input = bufferedReader.readLine();
intnumber = Integer.parseInt(input);
double d = Double.parseDouble(input);
```

::.. I/O su File (1/2)

- **L'IO su file è realizzabile per mezzo della combinazione di FileOutputStream e FileInputStream. Java dispone di un'altra classe che prende il nome di RandomAccessFile:**
- **Consente di effettuare operazioni di input/output direttamente in specifiche porzioni di un file.**
- **Non è derivata dalle classi basi InputStream/OutputStream.**
- **Il nome della classe deriva dal fatto che i dati possono essere letti e/o scritti su porzioni casuali all'interno di un file, invece di costituire un flusso continuo di informazioni.**

::.. I/O su File (2/2)

- **Per facilitare la gestione dell'IO su file, Java offre anche un'ulteriore classe: File.**
  - **Questa classe consente di accedere a uno o più file o directory, e utilizza le convenzioni di assegnazione dei nomi del sistema operativo host.**
- **Dalla versione 7, è offerta una ulteriore classe per la gestione dell'I/O su file: Path**

::.. Esempio di I/O su File (1/2)

```
…
File file = new File(“myFile.txt”);
try {
    file.createNewFile();
} catch (IOException x) {
    System.err.format("createFile error: %s%n", x.getMessage());}

//Operazione di lettura
InputStream in = null;
try {
    in = new FileInputStream(file);
    BufferedReader reader = new BufferedReader(
                                new InputStreamReader(in));
    String line = null;
    while ((line = reader.readLine()) != null)
        System.out.println(line);
} catch (IOException x) {
    System.err.println(x);
} finally {
    if (in != null) in.close();}
…
```

::.. Esempio di I/O su File (1/2)

```
…
File file = new File("myFile.txt");
try {
    file.createNewFile();
} catch (IOException x) {
    System.err.format("createFile error: %s%n", x.getMessage());}

//Operazione di lettura
Inpu
try
    BufferedReader reader = new BufferedReader(
                            new InputStreamReader(in));
    String line = null;
    while ((line = reader.readLine()) != null)
        System.out.println(line);
} catch (IOException x) {
    System.err.println(x);
} finally {
    if (in != null) in.close();}
…
```

Creazione di un file di nome "myFile.txt".

::.. Esempio di I/O su File (1/2)

Realizzazione operazione di lettura: 1) istanziazione InputStream sul file, 2) istanziazione di un BufferedReader, 3) lettura riga per riga del file, e 4) gestione eventuali eccezioni e 5) chiusura flusso.

```
…
File
try

} ca

//Operazione di
InputStream in = nul
try {
    in = new FileInputStream(file);  (1)
    BufferedReader reader = new BufferedReader(  (2)
                                new InputStreamReader(in));

    String line = null;  (3)
    while ((line = reader.readLine()) != null)
        System.out.println(line);
} catch (IOException x) {
    System.err.println(x);  (4)
} finally {
    if (in != null) in.close();}  (5)
…
```

::.. Esempio di I/O su File (2/2)

```
//Operazione di scrittura
String s = "Testo di prova";
byte data[] = s.getBytes();
OutputStream out = null;
try {
    out = new BufferedOutputStream(newFileOutputStream(file));

    out.write(data, 0, data.length);
} catch (IOException x) {
    System.err.println(x);
} finally {
    if (out != null) {
        out.flush();
        out.close();
    }
}
```

::.. Esempio di I/O su File (2/2)

```
//Operazione di scrittura
String s = "Testo di prova";
byte data[] = s.getBytes();
OutputStream out = null;
try {
    out = new BufferedOutputStream(newFileOutputStream(file));

    out.write(data, 0, data.length);
} catch (IOException x) {
    System.err.println(x);
} finally {
    if (out != null) {
        out.flush();
        out.close();
    }
}
```

2 1 3 4 5

Realizzazione operazione di scrittura: 1) istanziazione OutputStream sul file, 2) istanziazione di un BufferedOutputStream, 3) scrittura dell'intero array di byte, 4) gestione di eventuali eccezioni e 5) chiusura flusso.

::.. I/O su File con RandomAccessFile

```
…
File file = new File("DemoRandomAccessFile.out");
RandomAccessFile raf = new RandomAccessFile(file, "rw");

//Lettura di un singolo carattere
byte ch = raf.readByte();
System.out.println("Read first character of file: " + (char)ch);

//Lettura di una riga
System.out.println("Read full line: " + raf.readLine());
raf.seek(file.length());

//Scrittura di una nuova riga a fine file
raf.write(0x0A);
raf.writeBytes("This will complete the Demo");
raf.close();
…
```

::.. Java 1.1 I/O streams (1/2)

La libreria java.io in Java 1.1 estende la libreria di I/O con nuove classi, organizzate in due gerarchie (una per l'input e una per l'output) con a capo rispettivamente le classi astratte Reader e Writer.

Non si deve erroneamente pensare che Reader e Writer abbiano soppiantato InputStream e OutputStream. Esistono casi, infatti, dove si possono usare congiuntamente le vecchie classi con le nuove, usando classi "ponte":

- **InputStreamReader converte un InputStream in un Reader;**
- **OutputStreamWriter converte un OutputStream in un Writer.**

::.. Java 1.1 I/O streams (2/2)

- **Reader e Writer sono state introdotte per la gestione di caratteri Unicode a 16 bit, mentre le vecchie classi gestivano solo caratteri a 8 bit. Inoltre, le nuove classi sono più performanti.**

::.. StreamTokenizer (1/3)

- **Java offre una classe di supporto per i flussi chiamata StreamTokenizer (esiste anche un corrispettivo per le stringhe StringTokenizer).**
  - **non è una derivata di InputStream o OutputStream, e lavora solo con oggetti InputStream.**
  - **Viene utilizzata per spezzare il flusso di Input in segmenti lessicali chiamati "token", usando metodi speciali per identificare i parametri di tokenizzazione, come caratteri normali, spazi bianchi etc.**

::.. StreamTokenizer (2/3)

- **StreamTokenizer definisce quattro valori costanti:**
- **TT_EOF: ovvero fine flusso,**
- **TT_EOL: ovvero fine linea,**
- **TT_NUMBER: il token è un numero,**
- **TT_WORD: il token è una parola.**

- **L'utente può modificare i parametri del tokenizzatore secondo apposite funzioni, si rimanda alla documentazione online della classe per maggiori dettagli.**

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.TT_EOF) {
      String s;
      switch(st.ttype) {
        case StreamTokenizer.TT_EOL:
          s = new String("EOL");
          break;
        case StreamTokenizer.TT_NUMBER:
          s = Double.toString(st.nval);
          break;
        case StreamTokenizer.TT_WORD:
          s = st.sval;
          break;
        default:
          s = String.valueOf((char)st.ttype); }
      System.out.println( "Token = "+s);}
      inStream.close();
      inData.close();}}
```

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.
      String s;
```

Istanzio un oggetto BufferedReader, ovvero un filtro che bufferizza un flusso di Input. In ingresso pongo la conversione in Reader di un InputStream, che è quello canonico dalla tastiera. System.out e System.in sono due flussi, rispettivamente di input e di output. Il flusso così ottenuto è passato al costruttore di StreamTokenizer.

```
        default:
          s = String.valueOf((char)st.ttype); }
      System.out.println( "Token = "+s);}
      inStream.close();
      inData.close();}}
```

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.TT_EOF) {
      String
      switch
        cas                          TT_EOL:
          break;
        case StreamTokenizer.TT_WORD:
          s = st.sval;
          break;
        default:
          s = String.valueOf((char)st.ttype); }
      System.out.println( "Token = "+s);}
      inStream.close();
      inData.close();}}
```

Segnalo al Tokenizzatore che i caratteri '.' e '-' non devono essere presi in esame durante l'analisi del flusso

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.TT_EOF) {
      String s;
      switch(s
```

Entro in un ciclo while in cui:

- **Determino il token corrente;**
- **Valuto che tale token è il carattere di fine flusso;**
- **se sono al fine flusso esco dal ciclo.**

```
          s = String.valueOf((char)st.ttype); }
      System.out.println( "Token = "+s);}
      inStream.close();
      inData.close();}}
```

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    while (st.n                 () != StreamTokenizer.TT_EOF) {
      String s;
      switch(st.ttype) {
        case StreamTokenizer.TT_EOL:
          s = new String("EOL");
          break;
        case StreamTokenizer.TT_NUMBER:
          s = Double.toString(st.nval);
          break;
        case StreamTokenizer.TT_WORD:
          s = st.sval;
          break;
        default:
          s = String.valueOf((char)st.ttype); }
      System.out.println( "Token = "+s);}
      inStream.close();
      inData.close();}}
```

Valuto nel ciclo switch il tipo di token: carattere di fine riga, numero, parola o ignoto.

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.TT_EOF) {
      String s;
      switch(st.ttype) {
        case StreamTokenizer.TT_EOL:
          s = new String("EOL");
          break;
        case StreamTokenizer.TT_NUMBER:
          s = Double.toString(st.nval);
          break;
        case St                    izer.TT_WORD:
          s =
      System.out.println( "Token = "+s);}
      inStream.close();
      inData.close();}}
```

Prendo dal Tokenizer una rappresentazione numerica del token e la assegno a s

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.TT_EOF) {
      String s;
      switch(st.ttype) {
        case StreamTokenizer.TT_EOL:
          s = new String("EOL");
          break;
        case StreamTokenizer.TT_NUMBER:
          s = Double.toString(st.nval);
          break;
        case StreamTokenizer.TT_WORD:
          s = st.sval;
          break;
        default
```

Prendo dal Tokenizer una rappresentazione a stringa del token e assegno a s.

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.TT_EOF) {
      String s;
      switch(st.ttype) {
        case StreamTokenizer.TT_EOL:
          s = new String("EOL");
          break;
        case StreamTokenizer.TT_NUMBER:
```

Prendo dal Tokenizer il valore di tipo di token, opero il casting in char, e lo assegno a un oggetto di tipo String.

```
        default:
          s = String.valueOf((char)st.ttype); }
      System.out.println( "Token = "+s);}
      inStream.close();
      inData.close();}}
```

::.. StreamTokenizer (3/3)

```
public class StreamTokenApp {
  public static void main (String args[]) throws IOException {
    BufferedReader inData = new BufferedReader(
                            new InputStreamReader(System.in));
    StreamTokenizer st = new StreamTokenizer(inData);
    st.ordinaryChar('.');
    st.ordinaryChar('-');
    while (st.nextToken() != StreamTokenizer.TT_EOF) {
      String s;
      switch(st.ttype) {
        case StreamTokenizer.TT_EOL:
          s = new String("EOL");
          break;
        case StreamTokenizer.TT_NUMBER:
          s = Double.toString(st.nval);
          break;
        case StreamTokenizer.TT_WORD:
          s = st.sval;
          break;
        default:
          s = String.valueOf((char)st.ttype); }
      System.out.println( "Token = "+s);}
    inStream.close();
    inData.close();}}
```

Chiudo i flussi.

::.. Esempio di StringTokenizer

```
import java.io.*;
public class StringTokenApp {

    public static void main(String[] args) {

        String content[] = new String[2];

        content[0] = "Roberto,INFORMATICA,36";
        content[1] = "Pietro,ELETTRONICA,30";

        int i;
        for(i=0; i<content.length; i++) {

            StringTokenizer tok = new StringTokenizer(content[i], ",");

            String nome = tok.nextToken();
            String corso = tok.nextToken();
            int eta = Integer.parseInt(tok.nextToken());

            // ....

    }
}
```

Ogni stringa contiene 3 token (es. comma-separated values). Il formato del file deve rispettare il formato atteso dal programma.

Classe Scanner

- Soluzione semplcie per realizzare I/O da uno stream di caratteri.

```
import java.util.Scanner;
public class Prova {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
                        //istanzia un oggetto lettore di tipo Scanner
        String s = in.nextLine();
                        //legge una riga di testo e
                        //la memorizza nella variabile s
        System.out.println("Ho letto la riga: " + s);
    }
}
```

Input di una riga da tastiera

Classe Scanner

- Soluzione semplcie per reali

Lettura di interi da tastiera

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class Main {
        public static void main(String[] argv) {
                int i, j;
                Scanner scanner = new Scanner(System.in);
                try {
                        System.out.print("Primo numero: ");
                        i = scanner.nextInt();
                        System.out.print("Secondo numero: ");
                        j = scanner.nextInt();
                        System.out.println(i + j);
                } catch(InputMismatchException ex) {
                        System.out.println("Errore, input non valido.);
                } finally {
                        scanner.close();
                }
        }
}
```

Classe Scanner

- Soluzione semplcie per realizzare I/O da uno stream di caratteri.

```
import java.util.Scanner;

public class Main {
        public static void main(String[] argv)  {
                String input = "aaa 1 xyz 56 78 .,- pp 1092 yasl 3.14 100";
                Scanner scanner = new Scanner(input);
                while(scanner.hasNext()) {
                   // se il prossimo token è un intero lo estrae e lo visualiz
                   if(scanner.hasNextInt())
                   System.out.println(scanner.nextInt());
                   // altrimenti lo salta
                   else
                      scanner.next();
                }
                scanner.close();
        }
}
```

Lettura di token di interi

Classe Scanner

- Soluzione semplcie per realizzare I/O da uno stream di caratteri.

```
public class Test {
    public static void main(String[] args) {
        String s = "3; A; 4; A; 5; A";

        //Prima soluzione: metodi split() e trim() della classe String
        String[] vs = s.split(";");
        for(int i=0; i<vs.length; i+=2){
            System.out.print(vs[i].trim());
            System.out.println(vs[i+1].trim());
        }

        System.out.println("");

        //Seconda soluzione: oggetto di classe StringTokenizer
        StringTokenizer st = new StringTokenizer(s, ";");
        while(st.hasMoreTokens()) {
            System.out.print(st.nextToken().trim());
            System.out.println(st.nextToken().trim());
        }
    }
}
```

Estrazione di token

::.. Serializable

- La object serialization è il processo di **convertire un oggetto** (in memoria, prodotto da "new MyClass(...)") in una **sequenza di bytes**, e **salvarli su I/O**
  - File, DB, network
  - Anche denominato "marshaling" o "deflation"
- La deserializzazione è il processo inverso
  - Anche denominato "unmarshaling" o "inflation"

::.. ObjectStreams

- Gli **ObjectStreams** supportano l'I/O di oggetti
  - l'oggetto deve essere **Serializable** oppure **Externalizable**
- Le classi che possono essere utilizzate sono **ObjectOutputStream** (serializzazione) e **ObjectInputStream** (deserializzazione)

::.. Interfaccia Serializable

```
public class Person implements Serializable {
        // Code for the Person class goes here
}
```

- Nessun metodo è definito in **Serializable**
- Il programmatore non deve quindi implementare alcun metodo aggiuntivo
- Java si prende carico di analizzare il contenuto della classe (tramite i meccanismi di **reflection**), e salvarne tutte le variabili membro (salvo diversa indicazione del programmatore)

::.. Interfaccia Externalizable

```
public class Person implements Externalizable {

    public void readExternal(ObjectInput in)
            throws IOException, ClassNotFoundException {
        // Write the logic to read the Person object fields from the stream
    }

    public void writeExternal(ObjectOutput out) throws IOException {
        // Write the logic to write Person object fields to the stream
    }
}
```

- Il programmatore può implementare i metodi di Externalizable per controllare il salvataggio dell'oggetto (mediante singole letture/scritture delle variabili membro sullo stream in ingresso ai metodi)

::.. Esempio Serializable

```
// Create an object output stream to write objects to a file
ObjectOutputStream oos = new ObjectOutputStream(
                                 new
FileOutputStream("person.ser"));

// Serializes the john object
oos.writeObject(john);

// Close the object output stream
oos.close();

// Create an object input stream to read objects from a file
ObjectInputStream ois = new ObjectInputStream(
                                 new
FileInputStream("person.ser"));

// Read an object from the stream
Object obj = ois.readObject();

// Close the object input stream
ois.close();
```

::.. Esempio Externalizable

```
public class PersonExt implements Externalizable {
       private String name   = "Unknown";
       private String gender = "Unknown" ;
       private double height = Double.NaN;
       // ...

       public void readExternal(ObjectInput in)
            throws IOException, ClassNotFoundException {
              // Read name and gender in the same order they
were written
              this.name   = in.readUTF();
              this.gender = in.readUTF();
       }

       public void writeExternal(ObjectOutput out) throws
IOException {
              // we write only the name and gender to the stream
              out.writeUTF(this.name);
              out.writeUTF(this.gender);
       }
}
```

::.. Versioning di classi serializzabili

- Una classe può essere modificata nel tempo da un programmatore
  - Aggiunta o rimozione di variabili membro e metodi
- La nuova versione della classe potrebbe diventare **incompatibile con oggetti serializzati da versioni precedenti** della stessa classe
- Per rilevare tali circostanze di incompatibilità, Java introduce un codice numerico all'interno di ogni classe (**serial version unique ID - SUID**)

::.. Serial version unique ID

```
public class MyClass {
        // Declare the SUID field.
        // The "L" in "801890L" denotes a long value
        private static final long serialVersionUID = 801890L;
}
```

- Lo SUID può essere definito in più modi
  - Omesso: Java crea automaticamente lo SUID calcolando un "hash" sulla definizione della classe
  - Definito (arbitrariamente) dal programmatore
  - Definito dal programmatore, scelto con l'utility "serialver":
    `serialver -classpath <class-path> <class-name>`

::.. Serial version unique ID

- Se si de-serializza un oggetto dopo aver modificato la definizione di una classe (**alterandone** lo SUID), Java genera una **java.io.InvalidClassException**
- In alcuni casi, questo comportamento è desiderabile (se i cambiamenti alla classe sono "incompatibili")
  - Cancellazione di variabili membro
  - Cambio alla gerarchia di classi
  - Cambio da non-static a static
  - Cambio da non-transient a transient
  - Cambio di tipo di una variabile primitiva

::.. Serial version unique ID

- In altri casi, questo comportamento non è quello voluto dal programmatore (se i cambiamenti alla classe sono "compatibili")
  - Aggiunta di variabili membro
  - Cambio da static a non-static
  - Cambio da transient a non-transient
  - Aggiunta di classi alla gerarchia
- In questi casi, il programmatore può indicare la compatibilità tra le classi **definendo manualmente lo SUID**, e **mantenendo lo stesso valore** tra le versioni
- È buona pratica in generale **definire lo SUID nelle classi serializzabili**, per responsabilizzare il programmatore nella gestione delle versioni