I/O In Java

I/O in Java

- Flussi di dati (Stream)
- Buffer
- File
- Lettura e scrittura
- Interazione con il file system
- StringTokenizer/Scanner
- Serializzazione

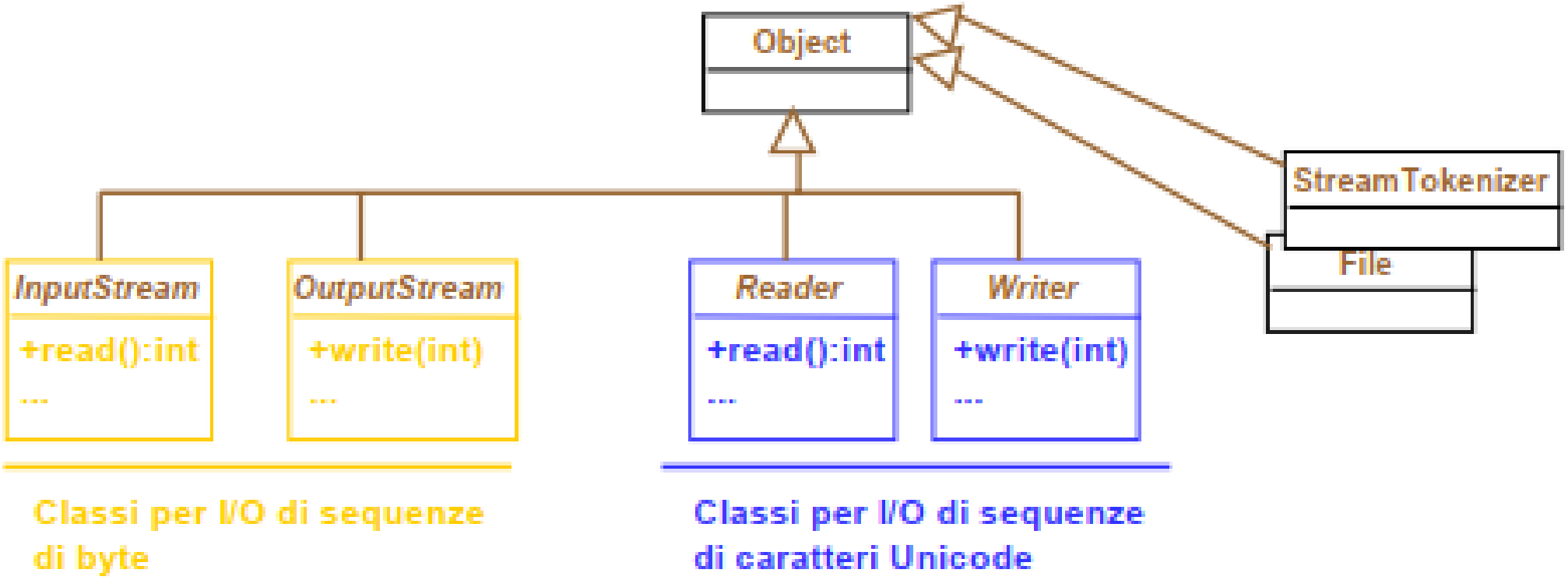
Stream

- Tutte le operazioni di I/O si riferiscono all'astrazione dello STREAM (flusso di byte)
- Uno stream può essere:
  - un file su disco
  - gli standard input, output, error
  - una connessione di rete
  - un flusso di dati associato a qualunque periferica
- Le operazioni di I/O funzionano nello stesso modo su **tutti** i tipi di stream

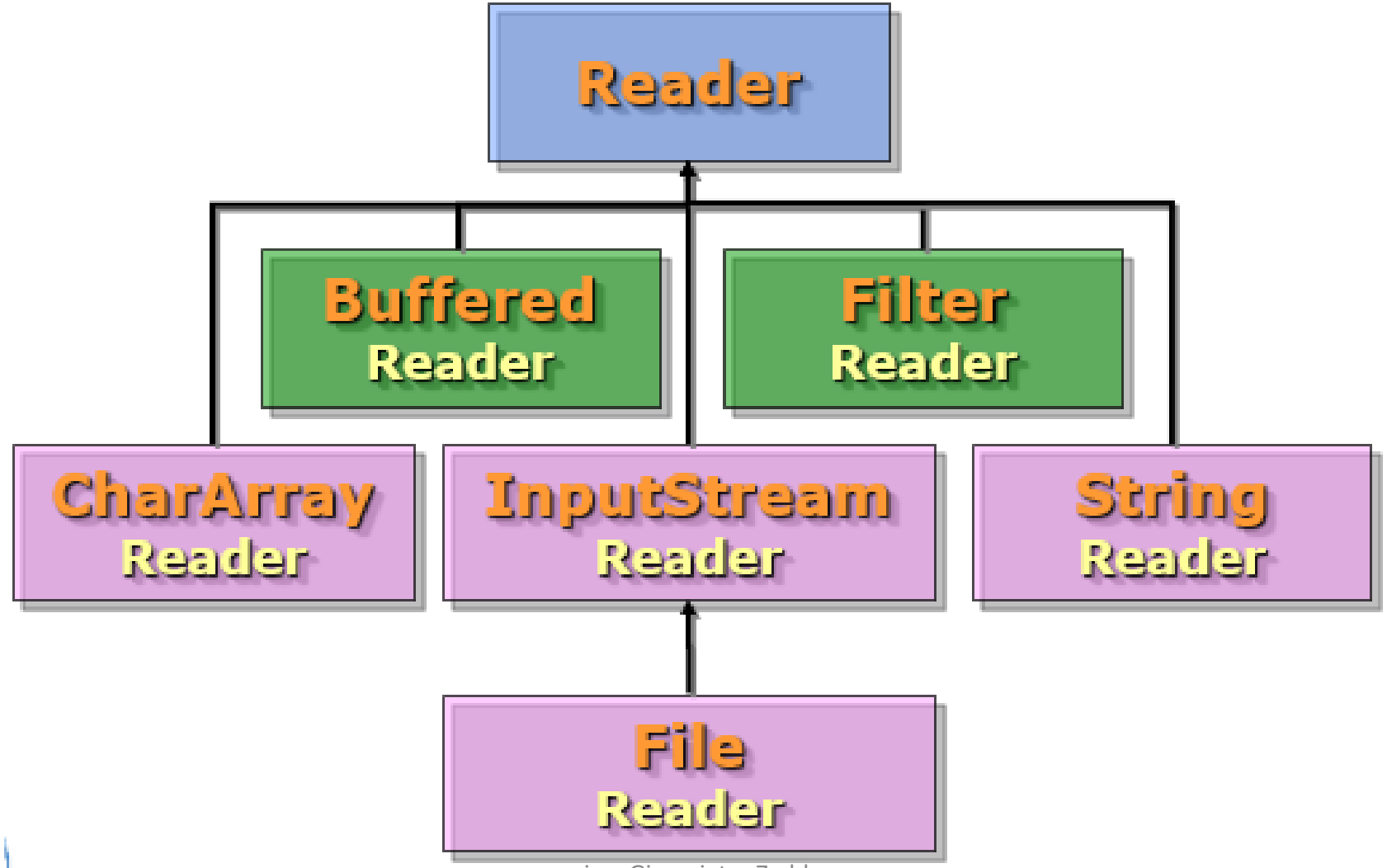
Stream

- Reader Writer
  - stream di char (caratteri Unicode 16 bit)
    - qualsiasi carattere
- InputStream OutputStream
  - stream di byte (8 bit)
    - dati binari, suoni, immagini
- package java.io
- tutte le exception relative sono sottoclassi di IOException

Classi di Primo Livello - java.io



Gerarchia di ereditarietà (char)

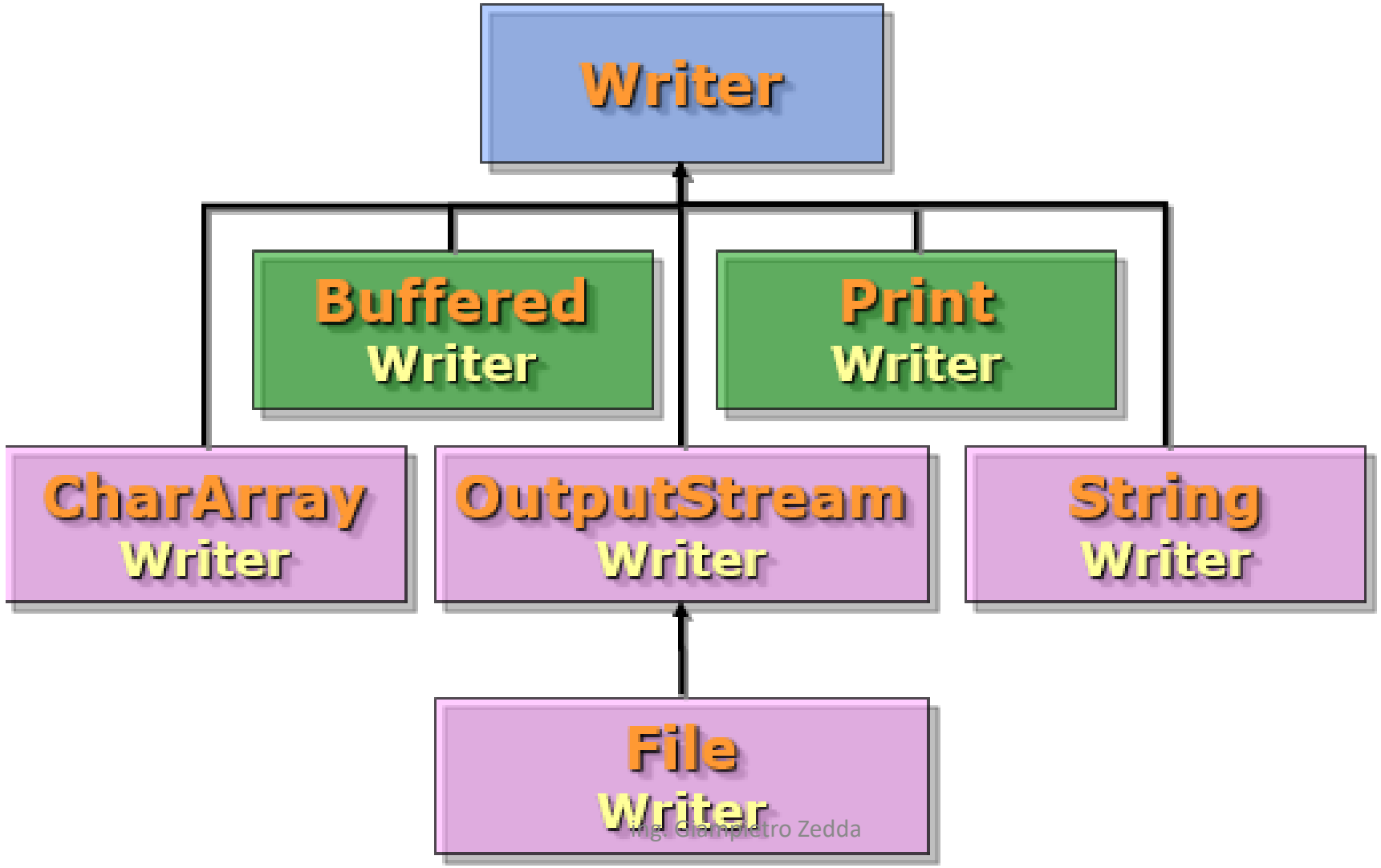


Reader (abstract)

- void close()
  - Close the stream.
- void mark(int readAheadLimit)
  - Mark the present position in the stream.
- boolean markSupported()
  - Tell whether this stream supports the mark() operation.
- int read()
  - Read a single character -1 when end of stream
  - will block until char is available, I/O error, end of stream
- int read(char[] cbuf)
  - Read characters into an array.
- abstract int read(char[] cbuf, int off, int len)
  - Read characters into a portion of an array.

- boolean ready()
  - Tell whether this stream is ready to be read.
- void reset()
  - Reset the stream.
- long skip(long n)
  - Skip characters.

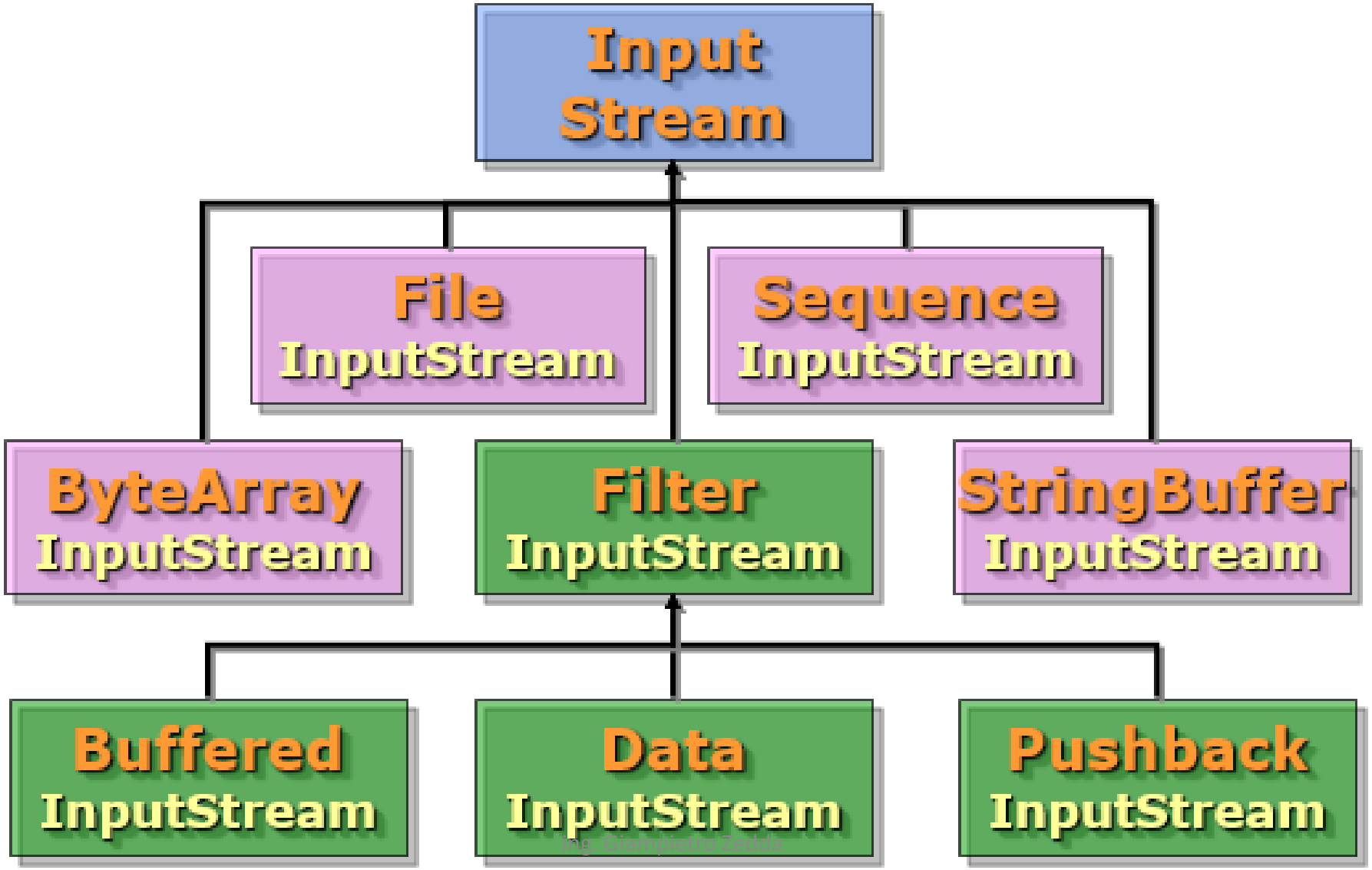
Gerarchia di ereditarietà (char)



Writer (abstract)

- close()
  - close the stream, flushing it first.
- abstract void flush()
  - Flush the stream.
- Void write(char[] cbuf)
  - Write an array of characters.
- Abstract void write(char[] cbuf, int off, int len)
  - Write a portion of an array of characters.
- Void write(int c)
  - Write a single character.
- Void write(String str)
  - Write a string.
- Void write(String str, int off, int len)
  - Write a portion of a string.

Gerarchia di ereditarietà (byte)

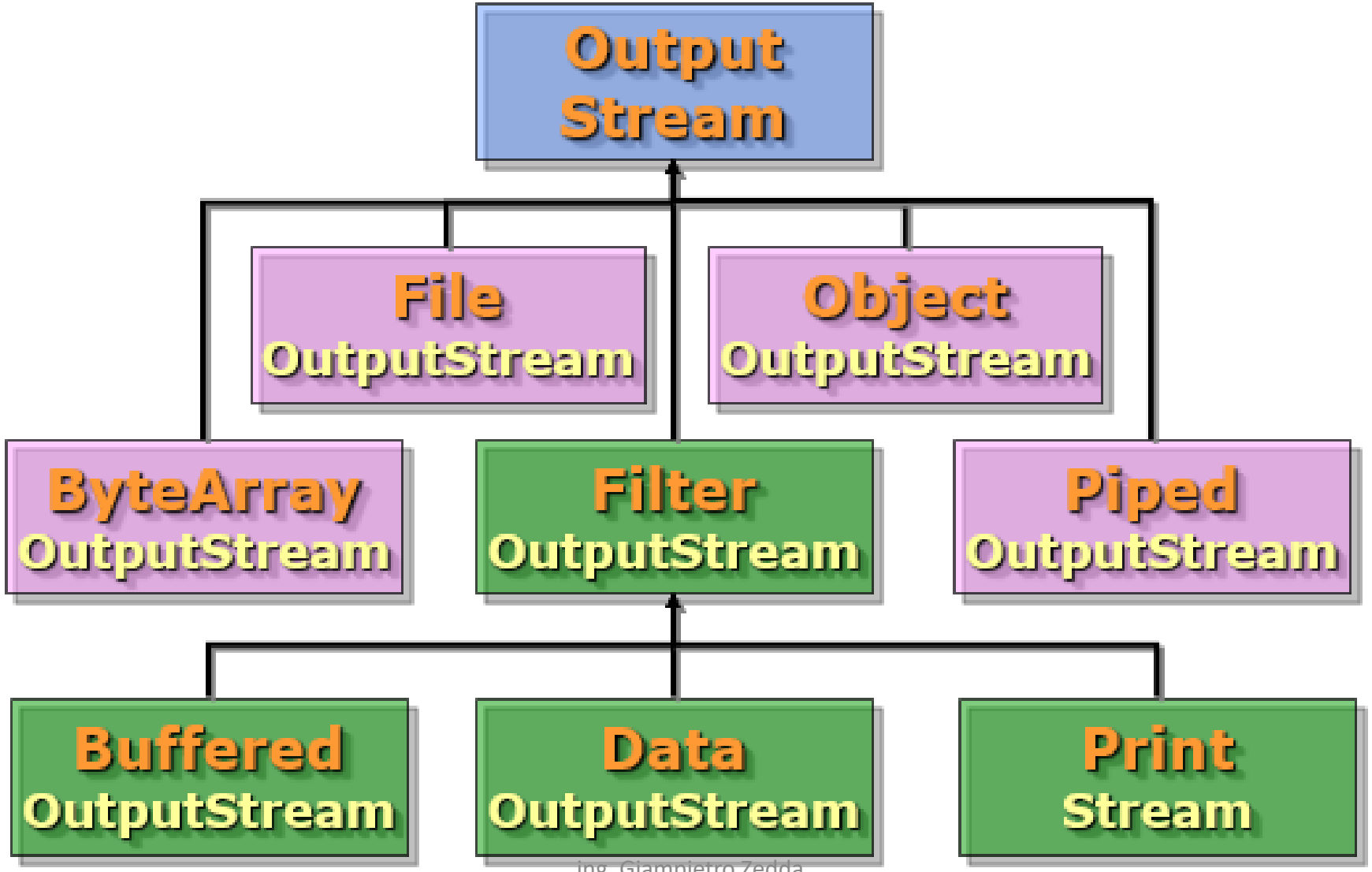


InputStream

- int available()
  - Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
- Void close()
  - Closes this input stream and releases any system resources associated with the stream.
- Void mark(int readlimit)
  - Marks the current position in this input stream.
- Boolean markSupported()
  - Tests if this input stream supports the mark and reset methods.
- Abstract int read()
  - Reads the next byte of data from the input stream.

- Int read(byte[] b)
  - Reads some number of bytes from the input stream and stores them into the buffer array b.
- int read(byte[] b, int off, int len)
  - Reads up to len bytes of data from the input stream into an array of bytes.
- Void reset()
  - Repositions this stream to the position at the time the mark method was last called on this input stream.
- long skip(long n)
  - Skips over and discards n bytes of data from this input stream.

Gerarchia di ereditarietà (byte)



OutputStream

- void close()
  - Closes this output stream and releases any system resources associated with this stream.
- void flush()
  - Flushes this output stream and forces any buffered output bytes to be written out.
- Void write(byte[] b)
  - Writes b.length bytes from the specified byte array to this output stream.
- Void write(byte[] b, int off, int len)
  - Writes len bytes from the specified byte array starting at offset off to this output stream.
- abstract void write(int b)
  - Writes the specified byte to this output stream.

in out

- System definisce lo stream in e out

```
class System {
  static InputStream in;
  static PrintStream out;  // PrintStream definita
                           // dopo
}
```

Specializzazioni di stream

- Memory
- pipe
- file
- buffered
- printed
- interpreted

Conversione byte - char

- InputStreamReader
- byte --> char
- OutputStreamWriter
- char --> byte

RW in memory

- CharArrayReader
- CharArrayWriter

- ByteArrayInputStream
- ByteArrayOutputStream

- RW di char o byte da/a array in memoria

RW in memory

- StringReader
- StringWriter
- RW di char da/a String
- StringBufferInputString
- legge bytes da StringBuffer

RW of Pipe

- PipedReader
- PipedWriter
  - RW di char da pipe
- PipedInputStream
- PipedOutputStream
  - RW di byte da pipe
  - pipe servono per la comunicazione tra thread

RW of File

- FileReader
- FileWriter
  - RW di char da file
- FileInputStream
- FileOutputStream
  - RW di byte da file
- File
  - gestione di filename e pathname

File

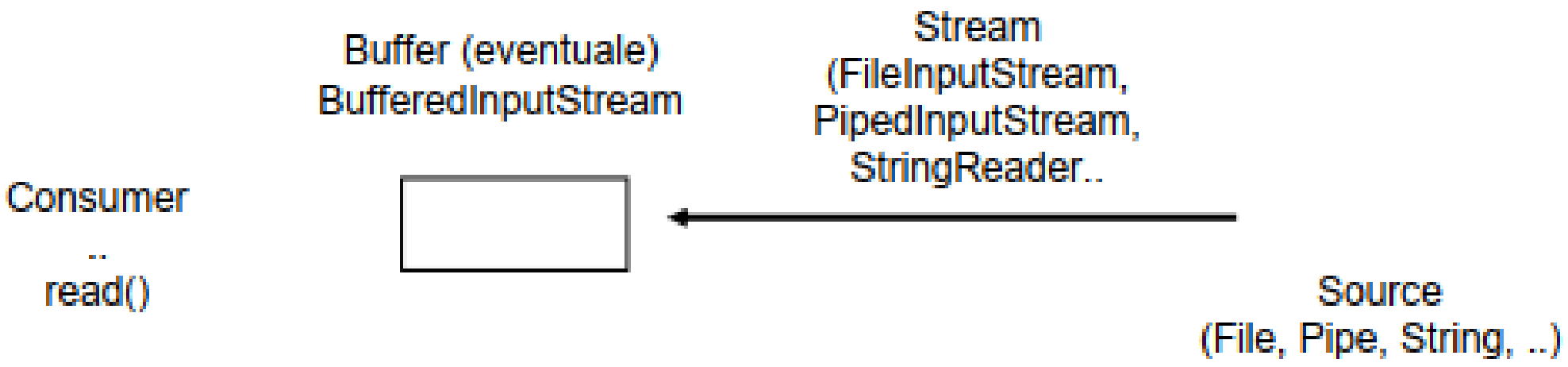
- abstract pathname
  - directory, file, fileseparator
  - absolute, relative
- converte abstract pathname <--> string
- metodi:
  - create() delete() exists() , mkdir()
  - getName() getAbsolutePath(), getPath(), getParent(), isFile(), isDirectory()
  - isHidden(), length()
  - listFiles(), renameTo()

Copia di file (char)

```java
import java.io.File;

public class FileCopiaChar {
    static final int EOF = -1;
    static final char CR = '\r';  // x'0D'
    static final char LF = '\n';  // x'0A'

    public static void main(String[] args) {
        File inputFile = new File("data.txt");
        File outputFile = new File("dataCopied.txt");
        try {
            FileReader in = new FileReader(inputFile);
            FileWriter out = new FileWriter(outputFile);
            int c;
            while ((c = in.read()) != EOF) {
                out.write(c);
            }

            // Inserimento riga vuota a fine file
            out.write(CR);   // CR Carriage return x'0D' (13)
            out.write(LF);   // LF Line feed       x'0A' (10)

            in.close();
            out.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("File copiato con successo");
        System.out.println("Input File: "       + inputFile.getName() + "\n"
                         + "Path Input File: "   + inputFile.getAbsolutePath() + "\n"
                         + "Ouput File: "        + inputFile.getName() + "\n"
                         + "Path Input File: "   + outputFile.getAbsolutePath() + "\n"
                         + "Length Input File: " + inputFile.length() + "\n"
                         + "Length Ouput File: " + outputFile.length() + "\n"
                );
    }
}
```

buffered

- BufferedInputStream
- BufferedOutputStream
- BufferedReader
  - readLine()
- BufferedWriter
- bufferizzano l'IO
  - BufferedInputStream(InputStream i)
  - BufferedInputStream(InputStream i, int size)




```
File in = new File("in.txt");
BufferedInputStream b = new BufferedInputStream (new FileInputStream( in));
.. while (b.read != -1 )  b.read();
```

printed

- PrintStream(OutputStream o)
  - print() println() per tipi primitivi e String
  - non throw IOException
    con checkError()

- System definisce out e err

```
class System {
  static PrintStream out, err;
}
```

interpreted

- Trasformano tipi primitivi Java in rappresentazione standard (UTF-8) su file
- DataInputStream(InputStream i)
  - readByte(), readChar(), readDouble(), readFloat(), readInt(), readLong(), readShort(), ..
  - readLine() deprecated (usare BufferedReader)
- DataOutputStream(OutputStream o)
  - analoghi write()

ReadLine da BufferedReader

- Per leggere una linea per volta da uno stream si utilizza la classe BufferedReader che fornisce il metodo

```
String readLine();
```

- BufferedReader NON può essere costruito direttamente a partire da un InputStream (come ad es. System.in), bisogna passare attraverso InputStreamReader:

```
BufferedReader in =
      new BufferedReader(
         new InputStreamReader(System.in));
```

Lettura da keybord

```
final static int EOF = -1;
StringBuffer buffer=new StringBuffer();
int input;
try{
 while((input=System.in.read())!='\n'){
   if(input == EOF) break;
   if(input != '\r'){
      char ch = (char)input;
      buffer.append(ch);
   }
}catch(Exception e){
  System.err.println("Error");
}
```

Termina la lettura quando incontra la fine dello stream

A capo:
in Unix: "\n"
in Win: "\n" + "\r"

Considera solo il primo byte di input

Lettura da keyboard con read()

```java
package esempiIO;

import java.io.IOException;

public class FileReadKeyboard {
    static final int EOF = -1;
    static final char CR = '\r';  // x'0D'
    static final char LF = '\n';  // x'0A'


    public static void main(String[] args) {

        StringBuilder buf = new StringBuilder();
//      char lineSep = System.getProperty("line.separator").charAt(0); // CR \r x'0D'
        int input;


        try {
            // Read from keyboard
            while ((input = System.in.read()) != LF) {
                if (input == EOF) {break;};
                if (input != CR) {
                    char ch = (char) input;
                    buf.append(ch);
                }
            }

            System.out.println("Stringa digitata: " + buf.toString());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

Lettura da keyboard con readLine ()

```java
package esempiIO;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class FileReadKeyboardReadline {

    public static void main(String[] args) {

        System.out.println("STOP per terminare");

        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        String line = "";
        while (!line.equals("STOP")) {
            try {
                line = in.readLine();
                if (line.equals("STOP")) {
                    break;
                }
                if (line.isEmpty()) {
                    continue;
                }
                System.out.println("Line: " + line);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        System.out.println("STOP digitato: fine elaborazione");
    }
}
```

Leggere da file

```
//analogamente

    BufferedReader br = new BufferedReader (new FileReader ("MioFile.txt" ));
    try{
        while ( (c = br.read()) != -1)  // c disponibile
     } catch (IOException e) {}


    // oppure procedere a righe
    BufferedReader br = new BufferedReader (new FileReader ("MioFile.txt" ));
    String s;
    try{
        while ( s = br.readLine() ) != null)  // s disponibile
     } catch (IOException e) {}
```

Lettura da file di testo con read()

```java
package esempiIO;

import java.io.BufferedReader;

public class FileReadBufferedRead {
    static final int EOF = -1;
    static final char CR = '\r';  // x'0D'
    static final char LF = '\n';  // x'0A'

    public static void main(String[] args) throws EOFException {

        File file = new File("data.txt");
        int c = 0;
        try {
            FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr);

            // Get char by char
            while ((c = br.read()) != EOF) {
                // Bypass control characters
                if (c == CR) {continue;}
                if (c == LF) {continue;}
//              System.out.print(c);
                System.out.print((char)c);
            }
            br.close();

        } catch (FileNotFoundException e) {
            System.out.println("File "+file.toString() + " Non trovato");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Lettura da file di testo con readLine()

```java
package esempiIO;

import java.io.BufferedReader;

public class FileReadBufferedReadline {

    public static void main(String[] args) throws EOFException {

        File file = new File("data.txt");
        try {
            FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr);

            // Get in loop lines
            while (true) {
                String line = br.readLine();
                if (line == null) {break;}
                System.out.println("Line: " +line);
            }
            br.close();

        } catch (FileNotFoundException e) {
            System.out.println("File "+file.toString() + " Non trovato");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Copia file con FileReader e FileWriter

```java
import java.io.File;

public class FileCopiaChar {
    static final int EOF = -1;
    static final char CR = '\r';  // x'0D'
    static final char LF = '\n';  // x'0A'

    public static void main(String[] args) {
        File inputFile = new File("data.txt");
        File outputFile = new File("dataCopied.txt");
        try {
            FileReader in = new FileReader(inputFile);
            FileWriter out = new FileWriter(outputFile);
            int c;
            while ((c = in.read()) != EOF) {
                out.write(c);
            }

            // Inserimento riga vuota a fine file
            out.write(CR);   // CR Carriage return x'0D' (13)
            out.write(LF);   // LF Line feed       x'0A' (10)

            in.close();
            out.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("File copiato con successo");
        System.out.println("Input File: "       + inputFile.getName() + "\n"
                         + "Path Input File: "   + inputFile.getAbsolutePath() + "\n"
                         + "Ouput File: "        + inputFile.getName() + "\n"
                         + "Path Input File: "   + outputFile.getAbsolutePath() + "\n"
                         + "Length Input File: " + inputFile.length() + "\n"
                         + "Length Ouput File: " + outputFile.length() + "\n"
                );
    }
}
```

StringTokenizer

- Permette di suddividere una stringa in base a dei separatori
- Costruttore:

```
StringTokenizer(String str, String delim)
```

- Metodi di iterazione:

```
boolean hasMoreTokens()
String nextToken()
```

Interpretare una riga

```
// riga letta disponibile in s
// il numero di token e' noto, ad es nome, eta', reddito

    String s;
    StringTokenizer st = new StringTokenizer(s);

    String nome = st.nextToken();
    int eta = Integer.parseInt(st.nextToken());
    double reddito = Double.parseDouble(st.nextToken());
```

Tokenizers

- StringTokenizer
  - lavora su String
  - set di delimiter (default: blank, “,”, \t, \n, \r, \f)
  - spezza la String in token (separati dai delimiter), rende token
  - hasMoreTokens(), nextToken()
  - non distingue identificatori, numeri, commenti, quoted strings

Tokenizers

- StreamTokenizer
  - lavora su Stream (Reader)
  - piu sofisticato, riconosce identifiers, commenti, quoted string, numeri
  - usa tabella di simboli e flag
  - nextToken(), TT_EOF se fine

Interpretare una riga

```
// riga letta disponibile in s
// il numero di token non e' noto


    String s;
    StringTokenizer st = new StringTokenizer(s);
    while (st.hasMoreTokens()){
      // trattare st.nextToken()
    }
```

Lettura righe strutturate

- Leggere i prodotti della cassa da un file di testo con le linee strutturate nel modo seguente

  `<codice> ; <Nome> ; <Prezzo>`

- Ogni linea letta dal file deve essere suddivisa in corrispondeza dei ";" (separatori)

Scrivere file

```
// FileWriter scrive caratteri su file
// BufferedWriter bufferizza per migliorare efficenza, solo metodo write
// PrintWriter offre println

    try {
    PrintWriter out = new PrintWriter( new BufferedWriter( new FileWriter( "File.txt")));

    out.println( );

    } catch(IOException e){}
```

Scrivere file con OutputStream (Byte)

```java
package esempi10;

import java.io.FileNotFoundException;

public class FileWriteByte {

    public static void main(String[] args) {

        String s = "This is a development software course";
        byte buf[] = s.getBytes();
        try {

            // Write byte by byte
            OutputStream os1 = new FileOutputStream("datawrite1.txt");
            for (byte b : buf) {
                os1.write(b);
            }
            os1.close();


            // Write a whole buffer
            OutputStream os2 = new FileOutputStream("datawrite2.txt");
            os2.write(buf);
            os2.close();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("Generated datawrite1.txt datawrite2.txt");
    }
}
```

Scrivere file con FileWriter (Char)

```java
package esempiIO;
import java.io.FileWriter;


public class FileWriteFileWriter {

    public static void main(String[] args) throws IOException {

        String s = "Test FileWriter\nto write characters";
        char bufChar [] = new char[s.length()];
        s.getChars(0, s.length(), bufChar, 0);  // Copy from s to bufChar

        // Write byte by byte
        FileWriter f1 = new FileWriter("filewriter1.txt");
        for (char c : bufChar) {
            f1.write(c);
        }
        f1.close();

        // Write all buffer
        FileWriter f2 = new FileWriter("filewriter2.txt");
        f2.write(bufChar);
        f2.close();

        // Write all buffer
        FileWriter f3 = new FileWriter("filewriter3.txt");
        f3.write(bufChar, bufChar.length - bufChar.length/4, bufChar.length/4);
        f3.close();

        System.out.println("Generated filewriter1.txt byte by byte");
        System.out.println("Generated filewriter2.txt byte by whole buffer");
        System.out.println("Generated filewriter3.txt from 3/4 for 1/4");

    }
}
```

Scrivere file con PrintWriter (Char)

```java
package esempiIO;
import java.io.IOException;
import java.io.PrintWriter;


public class FileWritePrintWriter {

    public static void main(String[] args) throws IOException {

        String r1 = "13 ITEM1 24";
        String r2 = "27 ITEM2 100";

        // Write byte by byte
        PrintWriter f1 = new PrintWriter("printwriter1.txt");
        f1.println(r1);
        f1.println(r2);
        f1.print("200");
        f1.print("ITEM3");
        f1.print("2000");
        f1.println();      // Inserisce il line separator
        f1.close();
    }
}
```

Scrivere file con BufferedWriter (Char)

```java
package esempiIO;
import java.io.BufferedWriter;


public class FileWriteBufferedPrintWriter {

    static final char CR = '\r';  // x'0D'
    static final char LF = '\n';  // x'0A'

    public static void main(String[] args) throws IOException {


        String r1 = "13 ITEM1 24";
        String r2 = "27 ITEM2 100";
        char bufChar1 [] = new char[r1.length()];
        char bufChar2 [] = new char[r2.length()];
        r1.getChars(0, r1.length(), bufChar1, 0);  // Copy from r1 to bufChar1
        r2.getChars(0, r2.length(), bufChar2, 0);  // Copy from r2 to bufChar2

        // Write byte by byte
        BufferedWriter f1 = new BufferedWriter(new PrintWriter("bufferedwriter1.txt"));
        // Write string
        f1.write(r1);
        f1.write(CR);
        f1.write(LF);
        f1.write(r2);
        f1.write(CR);
        f1.write(LF);

        // Write char
        f1.write(bufChar1);
        f1.write(CR);
        f1.write(LF);
        f1.write(bufChar2);
        f1.write(CR);
        f1.write(LF);

        f1.flush();    // ??
        f1.close();
```

Creazione file con DataOutputStream

```java
package esempiIO;

import java.io.DataOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCreaDataOutputStream {

    public static void main(String[] args) {
        char lineSep = System.getProperty("line.separator").charAt(0);

        try {
            DataOutputStream out = new DataOutputStream(
                    new FileOutputStream("fatturaNonChar.txt"));
            int[] units = {12, 8, 13, 29,50};
            double[] prices = {19.99, 9.99, 15.99, 3.99, 4.99};
            String[] descs = {"T-shirt", "Mug", "Duke", "Pin", "Key Chain"};
            for (int i = 0; i < prices.length; i++) {
                out.writeDouble(prices[i]);
                out.writeChar('\t');
                out.writeInt(units[i]);
                out.writeChar('\t');
                out.writeChars(descs[i]);
//              out.writeChar('\n');
                out.writeChar(lineSep);   // \r CR Carriage return x'0D' (13)
            }

            out.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

Lettura file con DataInputStream

```java
package esempiIO;

import java.io.DataInputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class FileReadDataInputStream {
    static final int EOF = -1;

    public static void main(String[] args) throws EOFException {

        DataInputStream in = null;;
        StringBuffer desc = new StringBuffer(20);
        int unit;
        double price;
        double total = 0.0;
        char chr = 0;;
        char lineSep = System.getProperty("line.separator").charAt(0); // CR \r x'0D'

        try {
            in = new DataInputStream(
                    new FileInputStream("fatturaNonChar.txt"));

            // Scan all rows
            while (true) {
                  price = in.readDouble();
                  in.readChar();            // Skip tab \t
                  unit = in.readInt();
                  in.readChar();            // Skip tab \t
                  total += unit * price;

                  // Scan description of current row
                  while ((chr = in.readChar()) != lineSep && chr != EOF) {
                    desc.append(chr);
                  }
                  // Print result
                  System.out.println("Ordinate "
                              + unit+" di "+desc+" Al prezzo di " + price);
                  desc = new StringBuffer(20);

            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (EOFException e) {
            System.out.println("Per un totale di " + total);
        } catch (IOException e) {
            e.printStackTrace();
    }

}
```

Esempio

- Scrivere un programma in grado di
  - scrivere un array di interi su un file
  - rileggere gli interi nell'array

Esempio

- array di interi

```
class IntArray implements Serializable {
 private   int interi[];
 public IntArray(){
}
 public void scarica(PrintWriter out){
  // no change
}
public void carica(BufferedReader br){
  // no change
}
```

```
public static void main(String[] args) {

IntArray a = new IntArray();
 try {
   PrintWriter out =   new PrintWriter (new BufferedWriter( new FileWriter("prova.txt")));
   a.scarica(out);

   BufferedReader br = new BufferedReader (new FileReader ("prova.txt" ));
  a.carica(br);
 } catch (IOException e){}
}
// varianti: carica(String s) e scarica(String s) ricevono s = nome file
```

```
class IntArray {
 private   int interi[];
 public IntArray(){
    interi = new int[30];
      for (int i=0; i<interi.length; i++){
        interi[i] = i+1;
      }
 }
public void scarica(PrintWriter out){
      out.println("Array di interi ");
      for (int i=0; i<interi.length; i++){    out.print(interi[i] + " ");  // tutti su una riga
     };
     out.flush();
     out.close();
 }
```

```
public void carica(BufferedReader br){
    String s=null;
    try{
        s = br.readLine(); // skip prima riga
        s = br.readLine();
    } catch (IOException e) {}
    StringTokenizer st = new StringTokenizer(s);
    int i = 0;
    while (st.hasMoreTokens()){
            interi[i++] = Integer.parseInt(st.nextToken());
    }
 }
}
```

Writers

- I writer non sono equivalenti
  - Non tutti rappresentano il new-line in maniera corretta

```
FileWriter fw = new FileWriter("provaFW.txt");
fw.write("Hello!\n");
PrintWriter pw = new PrintWriter(fw);
pw.println("Hello!");
```

- Se non vengono chiusi il buffer si perde!
  - `fw.close()`

URLs

- Gli stream possono essere collegati, oltre che ai file, a degli URL

```
URL page = new URL(url);
InputStream in = page.openStream();
```

- Occorre fare attenzione al tipo di file che si scarica

Serializzazione

- Per leggere/scrivere un oggetto occorre:
  - leggere/scrivere gli attributi (ed eventualmente il tipo) dell'oggetto
  - separare correttamente i vari elementi
  - in lettura, creare l'oggetto e impostarne tutti gli attributi
- queste operazioni (serializzazione) sono automatizzate da
  - `ObjectInputStream`
  - `ObjectOutputStream`

Uso della Serializzazione

- I metodi per leggere/scrivere oggetti sono:

```
void writeObject(Object)
Object readObject()
```

- Possono essere serializzati solo gli oggetti che implementano l'interfaccia `Serializable`
  - questa interfaccia non contiene metodi
  - ⇒serve soltanto per evitare che possano essere serializzati oggetti senza il consenso di chi ha scritto la classe

```
public static void main(String[] args) {
// salvare
try {
    ObjectOutputStream os = new ObjectOutputStream(new
                                                FileOutputStream("provaserial.txt"));
     os.writeObject(a);
     } catch (FileNotFoundException e) {
     } catch (IOException e) {}
}
```

```
public static void main(String[] args) {
// caricare
try {
    ObjectInputStream is = new ObjectInputStream(new
                                                FileInputStream("provaserial.txt"));
      a = (IntArray) is.readObject();
    } catch (FileNotFoundException e) {
    } catch (IOException e) {
    } catch (ClassNotFoundException e) {
}
```

```
public class Studente implements Serializable {
  // no change
}
public class StudentSet implements Serializable {
  // no change
}
public static void main(String[] args) {      // salvataggio su file
        Studente s1, s2, s3;
        s1= new Studente("Mario", "Rossi", 1234);
        s2= new Studente("Gianni", "Bianchi", 1243);
        StudentSet ss = new StudentSet();
        ss.add(s1);
        ss.add(s2);
     try {
         ObjectOutputStream os = new ObjectOutputStream(new
                                        FileOutputStream("studenti.txt"));
         os.writeObject(ss);
         os.close();
         } catch (FileNotFoundException e) {
         } catch (IOException e)    }
}
```

```
public static void main(String[] args) {      // caricamento da file
        StudentSet ss =null;

     try {
          ObjectInputStream is = new ObjectInputStream(new FileInputStream("studenti.txt"));
          ss = (StudentSet) is.readObject();

     } catch (FileNotFoundException e) {
     } catch (IOException e) {
     } catch (ClassNotFoundException e) {
     }
    ss.print();
   }
```