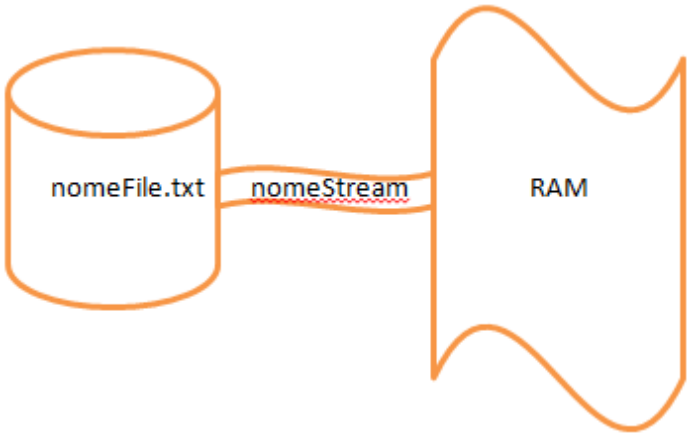
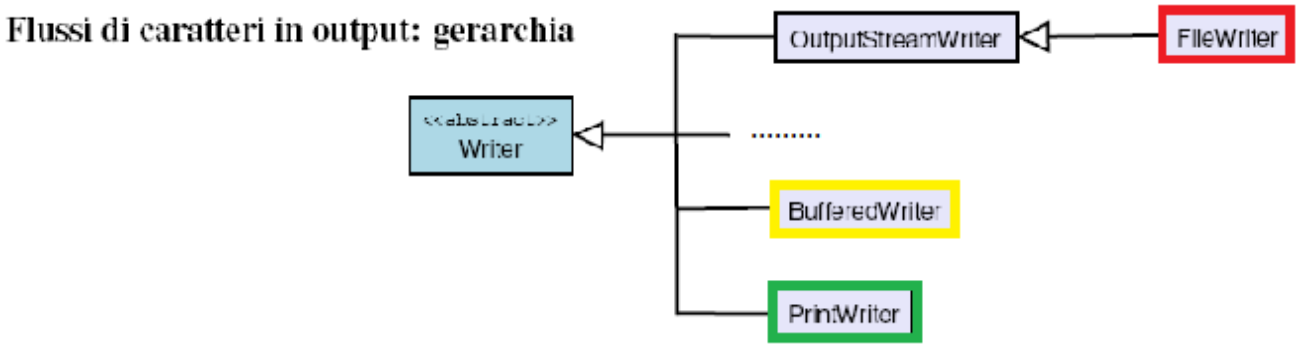
Accesso a file di testo: uso di stream o flusso di caratteri[1]



Scrittura su file di testo: package java.io

FileWriter nomeStream = new FileWriter (“nome_file.txt“); // **sovrascrive**

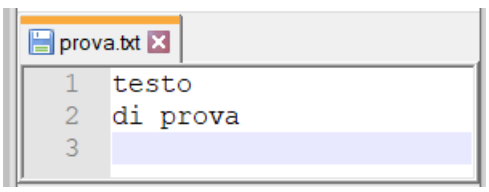
FileWriter nomeStream = new FileWriter(“nome_file.txt[2]“, true); // **aggiunge**



Flussi di **caratteri** di uscita (*writer* o scrittori), da utilizzare per scrivere un file di testo

FileWriter (String nome) **apre il file** nome **in scrittura** (lo crea se non esiste, lo azzera se esiste cioè sovrascrive) collegandolo ad uno scrittore, questo scrittore possiede il metodo **write**(x) dove x può essere un intero, una stringa o una sola parte, un'array di char o una sola parte. Per scrivere un singolo carattere alla volta all’interno del file

```
try {    // codice che può sollevare (cioè è soggetto a) eccezioni

        FileWriter cstamp = new FileWriter ("prova.txt");

        cstamp.write ("testo\ndi prova\n");  // gestione “a capo”

        cstamp.close();   // solo dopo la chiusura salva su file
}
catch (IOException e) { …[3] }
```



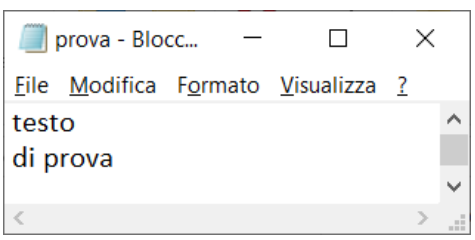
[1] Da manualetto introduttivo (pg.33); altri esempi (pg.46 creando una “pagina web” e pg.47 nell’uso di *finestre di dialogo*

[2] Un qualsiasi file di testo con estensione *.txt* potrà essere *importato* in un *foglio elettronico* seguendo i passaggi guidati di conversione

[3] Eccezione che può essere scatenata se non esiste il percorso in cui si vorrebbe creare il file

Codice dell'applicazione

```
/**
 * ScriviFile
 *
 * @author 3INF
 * @version 1
 */
import java.io.*;

public class ScriviFile {

   /** nome del file di testo */
   private String s;

   /**
    * Costruttore di default  degli oggetti di classe ScriviFile
    */
   public ScriviFile(){

      s = "prova.txt";   // si provi ad impostare il percorso ./testi/prova.txt
                         // senza aver creato la sottodirectory testi nella stessa cartella "di lavoro"
                         // cioè la sottocartella dove sono archiviati i file.class (bytecode)
   }

   /**
    *  metodo che scrive su file di testo
    */
   public void scrivi(){
      try {     // inizio del codice che può sollevare  (cioè è soggetto a) eccezioni

         FileWriter cstamp = new FileWriter (s);

         cstamp.write ("testo\ndi prova\n"); // gestione "a capo"

         cstamp.close();  // solo dopo la chiusura, scrive su file
       } catch (IOException e) {
                                        System.out.println("errore tentando di scrivere su file "+e);
                                    }
     }

   public static void main(String [] args){

      ScriviFile sf = new ScriviFile();
      sf.scrivi();
   }
}
```



I/O bufferizzato e formattato

La classe FileWriter fornisce un *metodo basico* per scrivere caratteri su file. Non è conveniente usarla direttamente nei programmi perché:

- rende un programma inefficiente, visto che ogni operazione di I/O (***write***) richiede un accesso al file;
- non permette di scrivere dati più complessi come stringhe e numeri.

Altre classi Java permettono di creare oggetti ***wrappers***: incapsulano gli oggetti della classe **FileWriter** estendendone le funzionalità.

➢ Un oggetto di tipo **PrintWriter** permette l'accesso a file di testo per **scrivere con formato**.

PrintWriter (String *nome*) - aperto il file *nome* in scrittura - lo collega ad uno scrittore che possiede anche i metodi void **print**(x) e **println**(x) ove x può essere un boolean, char, int, long, float, double, Object o String; l'argomento x verrà convertito in una stringa di caratteri che verrà poi trasferita in uscita; println() si comporta come il corrispondente print() ma aggiunge un *fine riga* al termine della stringa con corretto 'a capo'.

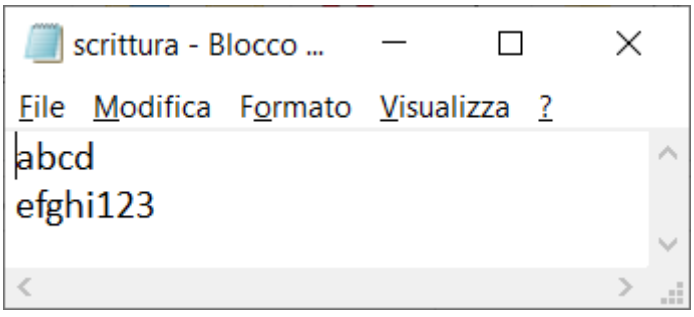
➢ Un oggetto di tipo FileWriter si può inglobare in un oggetto di tipo **BufferedWriter** (*con memoria tampone*) per scrivere intere stringhe **riducendo gli accessi** ed **aumentando l'efficienza**.

(esempio seguente ed altri)

// crea flusso di output bufferizzato

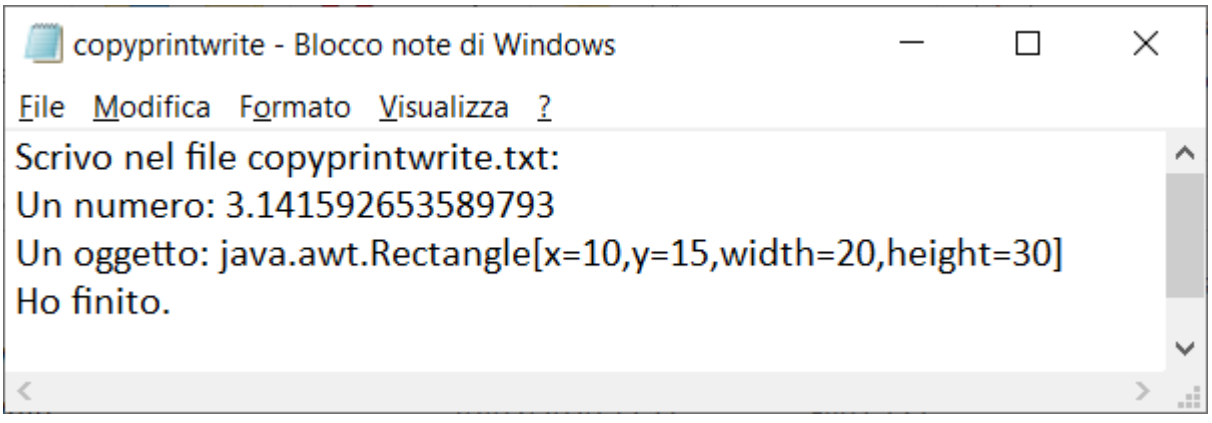
FileWriter fw = new FileWriter ("*nome_file*.txt");
BufferedWriter out = new BufferedWriter (fw); **/* per scrivere intere stringhe e non singoli caratteri utilizza il metodo write() */**

```
import java.io.*;

class ScriviUna {
  public static void main(String args[])
  throws IOException {                    // rilancia l'eccezione di tipo IOException
    FileWriter w;
    w=new FileWriter("scrittura.txt");

    BufferedWriter b;
    b=new BufferedWriter (w);

    b.write("abcd\nefghi");

    b.write("123");

    b.flush();  // solo dopo aver svuotato il flusso, scrive su file
  }
}
```



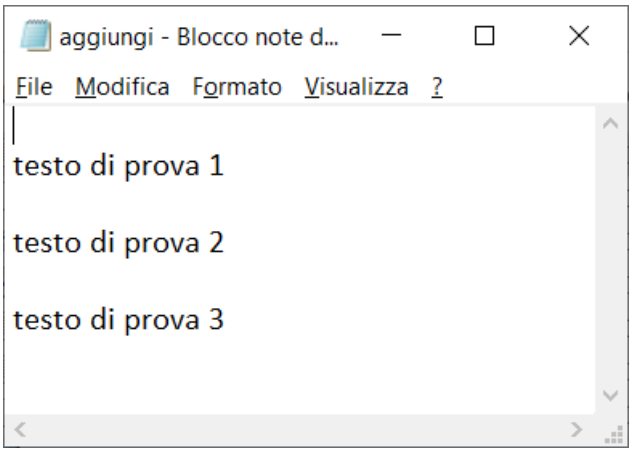
da esempio online

```
import java.io.*;

public class CopyPrintWrite {

  public static void main(String[] args)
  throws IOException {                    // rilancia l'eccezione di tipo IOException

                // creo un oggetto FileWriter...
   FileWriter fileout = new FileWriter("copyprintwrite.txt");

                // ... che incapsulo in un BufferedWriter...
   BufferedWriter filebuf = new BufferedWriter(fileout);

                // ... che incapsulo in un PrintWriter
   PrintWriter printout = new PrintWriter(filebuf);

   printout.println("Scrivo nel file copyprintwrite.txt:");
   printout.print("Un numero: ");
   printout.println(Math.PI);
   printout.print("Un oggetto: ");
   printout.println(new java.awt.Rectangle(10,15,20,30));
   printout.println("Ho finito.");
   printout.close();
  }
}
```



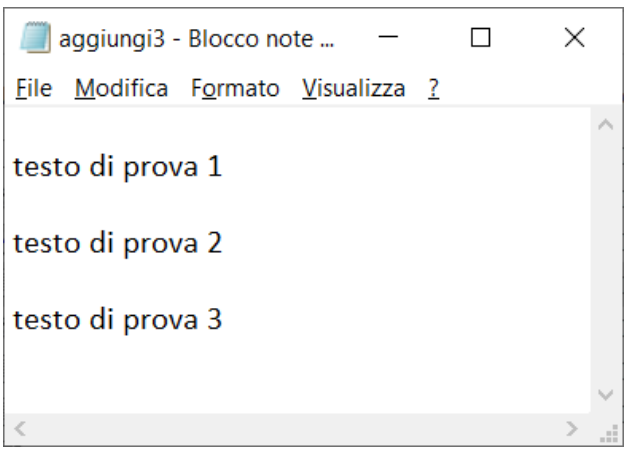
NB: con la clausola **throws** (lancia ... o meglio *ri-lancia*) nella dichiarazione del metodo, in pratica è come se avvertissimo il compilatore che siamo consapevoli che il metodo possa lanciare al runtime quell'eccezione, e di non "preoccuparsi", perché gestiremo in un'altra parte del codice l'eccezione. Tale meccanismo di **propagazione dell'eccezione** permette in realtà di non gestire le eccezioni (è possibile infatti che – rilancio dopo rilancio – lo stesso metodo main non gestisca)

```java
/**
 * ScriviOggetto
 *
 * @author 3INF
 * @version 1
 */
import java.io.*;

public class ScriviOggetto {

   /** nome del file di testo */
   private String s;

   /**
    * Costruttore di default degli oggetti di classe  ScriviFile
    */
   public ScriviOggetto(){

      s = "aggiungi.txt";
   }

   /**
    *  metodo che scrive su file di testo (aggiungendo) con formattazione
    *  @param n  numero da accodare al testo
    */
   public void scrivi(int n){

      try {     // inizio del codice che può sollevare  (cioè è soggetto a) eccezioni

          FileWriter fw = new FileWriter (s, true); // aggiunge

          BufferedWriter bw=new BufferedWriter (fw);

          PrintWriter pw = new PrintWriter(bw);

          pw.println("\ntesto di prova " + n);

          pw.close();       // solo dopo la chiusura salva su file
        }
      catch (IOException e) {   }
   }

   public static void main(String [] args){

      ScriviOggetto sf = new ScriviOggetto();
      int max = 3;  // numero di linee di testo da scrivere/aggiungendo
      for (int i = 1; i<=max; i++)
           sf.scrivi(i);
   }
}
```



Analogo effetto, senza formattazione:

```
/**
 *  metodo che scrive su file di testo "appendendo" senza formattazione
 *  @param n  numero da accodare al testo
 */
public void appendi (int n){

   try {     // inizio del codice che può sollevare  (cioè è soggetto a) eccezioni

        FileWriter fw = new FileWriter (s,true);

        BufferedWriter bw=new BufferedWriter (fw);

        bw.append ("\ntesto di prova " + n + "\n");


        bw.flush();  // solo dopo lo svuotamento
                     // del buffer salva su file

      }
    catch (IOException e) {   }
}
```



NB: *si noti che il metodo* **flush()** *consente la* ***scrittura su file***
ed in Java ***non sempre è necessario l'uso esplicito del*** ***metodo conclusivo … close()***

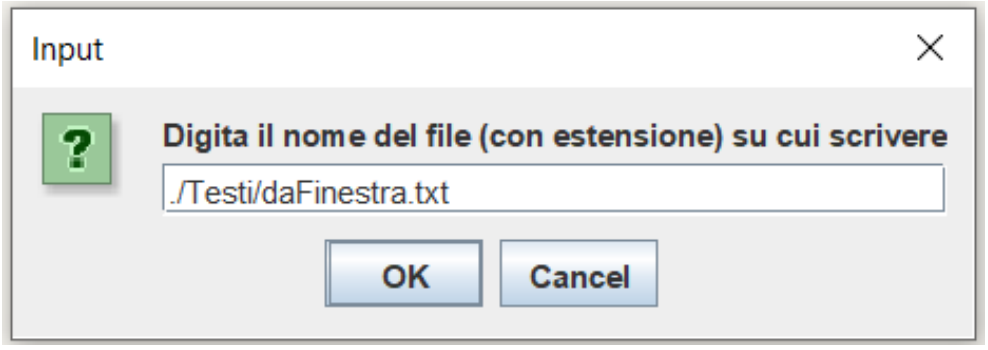
Quando **non esistono riferimenti ad un oggetto,**

il **meccanismo automatico** di ***garbage collection***

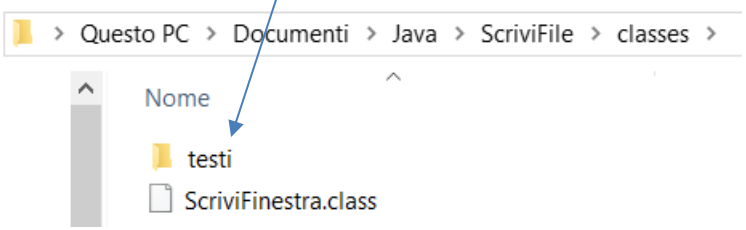
individua e libera tale area di *memoria inaccessibile*

Applicazione con uso di finestre di dialogo (gestendo eventi)

Si vuole creare un programma che generi una sequenza di stringhe e le scriva in un file di testo, il cui nome è letto da tastiera con richiesta all'utente mediante una *finestra di dialogo (dialog box)*



Il file viene creato potendo indicare il *percorso* a partire dalla cartella che contiene il programma stesso (nomi di file e sottocartelle *non sono case-sensitive*). Il numero di righe del testo è fisso (pari a 10).




```
import java.io.*;
import javax.swing.JOptionPane;
public class ScriviFinestra{
  public static void main(String[] args) {
  String s =JOptionPane.showInputDialog("Digita il nome del file (con estensione) su cui scrivere");
                                        // scoperta: se si preme (pg.2) Cancel o icona ×
  FileWriter fw;
  try {
        fw = new FileWriter (s);
  }
  catch (IOException e)   {
                System.out.println("errore in apertura file "+e);
                 fw=null;          // l'oggetto non è più  referenziato …. Il meccanismo di
                                   //  garbage  collection individua e libera l'area di memoria
          }
   try {
          for(int i=0; i<10; i++) {
                fw.write("questa e' la riga "+i+ "\n");
          }
          fw.close();
  }
  catch (IOException e) {System.out.println("errore in scrittura file "+e);  }
  } // fine main
} // fine applicazione
```

NB: scoperta …. problemi aperti …. **Come evitare eccezione** di tipo java.lang.**NullPointerException** sollevata quando, invece di premere sul pulsante **OK, l'utente** preme su **Cancel** o sull'**icona di chiusura**? × ….. **Suggerimento**: possibile domanda **if (s != null)** …..

Vedremo in seguito **Come evitare eccezione di tipo java.io.FileNotFoundException se non si scrive nella casella di testo …. quindi come scoprire se l'oggetto String s è uguale a " "**

>> Confrontare String (pg.10)