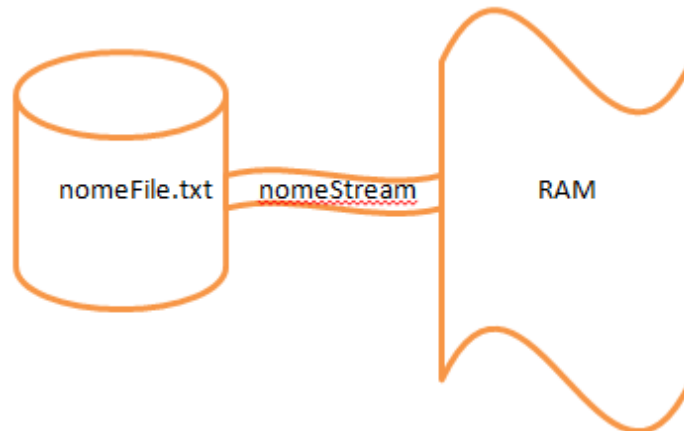


Accesso a file di testo: uso di stream o flusso di caratteri

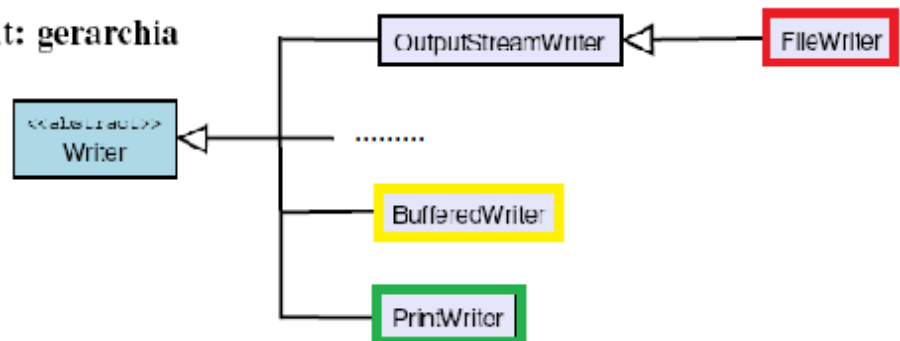


Scrittura su file di testo

```
FileWriter nomeStream = new FileWriter ( "nome_file.txt" ); // sovrascrive
```

```
FileWriter nomeStream = new FileWriter( "nome_file.txt", true ); // aggiunge
```

Flussi di caratteri in output: gerarchia



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 azzerava se esiste cioè sovrascrive) collegandolo ad uno scrittore, questo scrittore possiede il metodo **write(x)** ove 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 { // inizio del codice che può sollevare (cioè è soggetto a) eccezioni
```

```
    FileWriter cstamp = new FileWriter ("retta.txt");  
    cstamp.write ("ascisse\tordinate\n");
```

```
    for (int i = 0; i < N; i++ , x = x + dx) {  
        y = m*x + q;  
        cstamp.write( x + "\t" + y + "\n");  
    }
```

```
    cstamp.close(); // solo dopo la chiusura memorizza su file
```

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

I/O bufferizzato e formattato

Le classi `FileReader` e `FileWriter` forniscono i metodi basici per leggere o scrivere caratteri su file. Non è conveniente usarle direttamente nei programmi perché:

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

Altre classi Java permettono di creare oggetti *wrappers*: incapsulano gli oggetti delle classi **`FileReader`** e **`FileWriter`** estendendone le funzionalità.

Un oggetto di tipo `FileWriter` si può inglobare in un oggetto di tipo **`PrintWriter`**

`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. Per scrivere con formato.

Un oggetto di tipo `FileReader` si può inglobare in un oggetto di tipo **`BufferedReader`** 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");
```

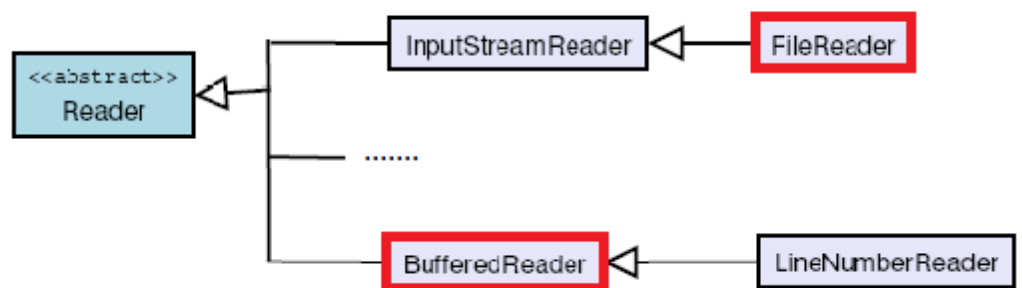
```
BufferedReader out = new BufferedReader (fw);
```

```
/* per scrivere intere stringhe e non singoli caratteri:  
utilizza metodo write()
```

```
evita di dover usare il metodo flush per svuotare il buffer */
```

Letture da file di testo

Flussi di caratteri in input: gerarchia



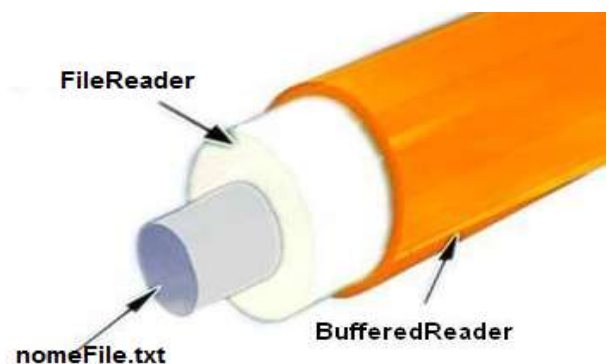
```
BufferedReader nomeStream = new BufferedReader (new FileReader ("nome_file.txt"));
```

FileReader(String nomeFile) apre il file nomeFile collegandolo al lettore.

Sono disponibili i metodi di lettura

- int **`read()`** che legge il successivo byte, lo converte in un carattere e ne restituisce il valore sotto forma di un intero maggiore o uguale a zero. Se si raggiunge la fine del file il metodo restituisce il valore -1
- String **`readLine()`** che ritorna come oggetto String la sequenza di caratteri letti con gestione dell'a capo

```
String s = oggettoBufferedReader.readLine();
```



Leggere un file di testo con stream bufferizzato

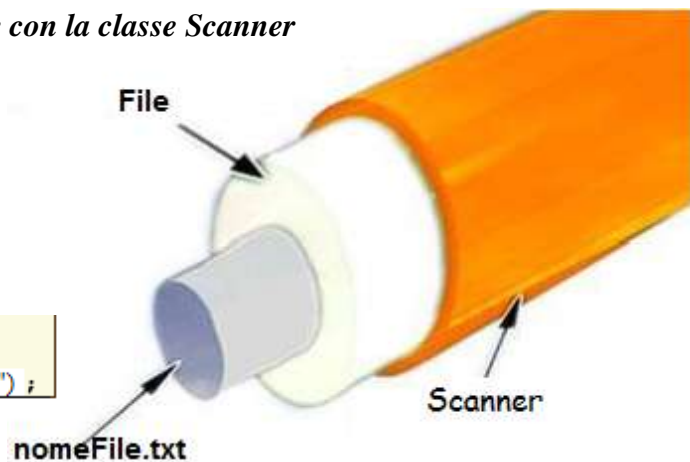
```
try {  
    FileReader file = new FileReader("../src/LeggiSorgente.java");  
  
    BufferedReader buff = new BufferedReader (file);  
    boolean eof = false;  
    while(!eof){  
        String line = buff.readLine();  
        if ( line == null )                // se la stringa letta è null  
            eof = true;  
        else  
            System.out.println (line);  
    }  
    buff.close();  
} catch (IOException e) { }
```

Leggere un file con la classe Scanner

La classe Scanner

Un oggetto di **tipo Scanner** consente di leggere da qualsiasi flusso di ingresso (ad es. un **file**)

```
import java.util.Scanner;  
...  
Scanner in = new Scanner( new File("nomeFile") );
```



Alcuni metodi di Scanner

Prima di tutto si deve importare la classe **Scanner** all'interno del file .java che ne fa uso.

All'interno del codice si deve creare un nuovo **oggetto** della classe **Scanner**

Poi si possono invocare i metodi:

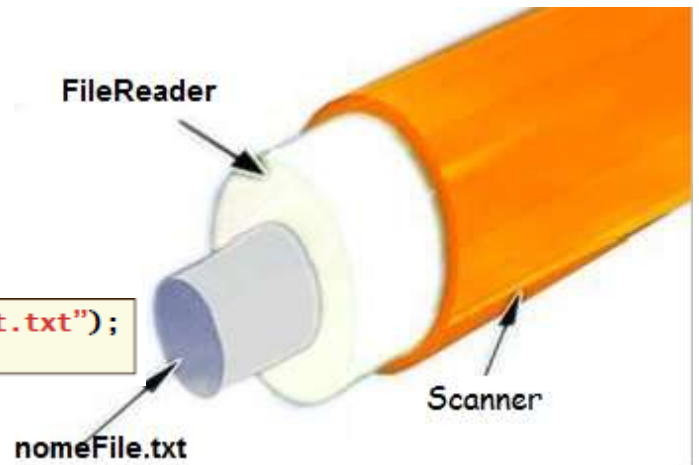
- Leggere una intera **riga** (fino alla pressione di **Enter**)
String city = in.**nextLine**();
- Leggere una **parola** (fino al primo **carattere di spaziatura**: spazio, fine riga, tabulazione)
String state = in.**next**();

nb: La classe **File** definisce oggetti file che vengono associati ai file fisici del file system e un insieme di metodi per creare, cancellare e gestire file. Non definisce metodi per leggere o scrivere.

Metodo alternativo

- ⤴ Creare un oggetto “lettore di file” (FileReader)
- ⤴ Creare un oggetto Scanner
- ⤴ **Collegare** l’oggetto Scanner al lettore di file

```
FileReader reader = new FileReader("input.txt");  
Scanner in = new Scanner(reader);
```



In questo modo si possono usare i metodi di Scanner (next, nextLine, ecc.) per leggere i dati contenuti nel file

```
FileReader reader = new FileReader("file.txt");  
Scanner in = new Scanner(reader);  
while(in.hasNextLine())  
{ String line = in.nextLine();  
  //... elaborazione della stringa  
} // il costruttore FileReader lancia IOException, da gestire!!
```

Chiudere file in lettura

Al termine della lettura del file (che non necessariamente deve procedere fino alla fine...) **occorre chiudere il file**

```
FileReader reader = new FileReader("file.txt");  
...  
reader.close();
```

- ⤴ Il metodo **close()** lancia **IOException**, da gestire obbligatoriamente
- ⤴ Se il file non viene chiuso non si ha un errore, ma una **potenziale situazione di instabilità** per il sistema operativo

```
import java.io.*;  
import java.util.Scanner;
```

```
public class LeggiScanner2 {  
    public LeggiScanner2() throws IOException{  
        try{  
            FileReader fr = new FileReader("../src/xanadu.txt");  
            Scanner in = new Scanner(fr);  
            while (in.hasNextLine()) { // se l'input termina hasNextLine() ritorna falso  
                String line = in.nextLine(); // lettura di una riga alla volta  
                System.out.println(line);  
            }  
            fr.close();  
        }catch (FileNotFoundException e){ }  
    }  
    public static void main(String[] args){  
        try{  
            new LeggiScanner2();  
        }catch (IOException e){ }  
    }  
}
```