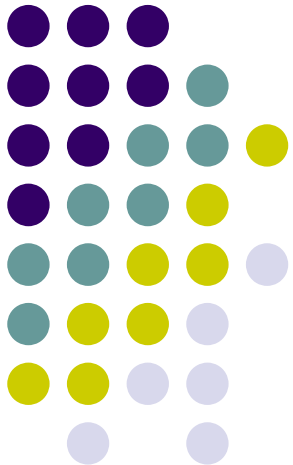
Università di Napoli Federico II - Dip. di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica

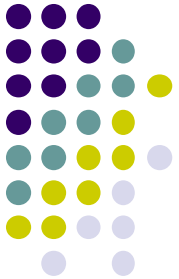
DIETI

Corso di Programmazione

File di testo – Aspetti base

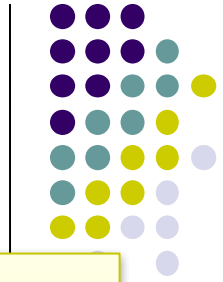


Stream



- Il sistema di I/O dei linguaggi C,C++ e Java si basa sul *concetto* di canale (**stream**) inteso come mezzo attraverso cui fluiscono le informazioni provenienti o inviate dai diversi dispositivi hardware (le unità esterne)
- In altre parole **lo stream è un flusso di dati** (sorgente o destinazione) che può essere associato ad un disco o ad altre periferiche
- Questi canali possono essere connessi a dispositivi fisici diversi mantenendo inalterato il proprio comportamento: un flusso di informazioni, per mezzo delle stesse modalità, può essere mandato in uscita su un video, su una stampante o su un file ed analogamente un flusso di informazioni può provenire adottando le stesse modalità dalla tastiera o da un file presente nella memoria di massa

Stream di dati

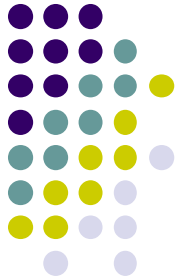


In altre parole uno stream **è una astrazione** che consente di definire una interfaccia comune a diversi dispositivi di I/O in modo da:

- rendere la scrittura dei programmi indipendenti dal particolare dispositivo impiegato;
- semplificare i problemi di portabilità dei programmi stessi.

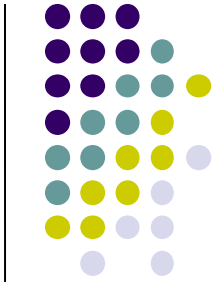
Lo stream consente di definire un'interfaccia semplice (astrazione di un dispositivo generico) e uniforme verso l'utente.

Connessione di uno stream ad un dispositivo



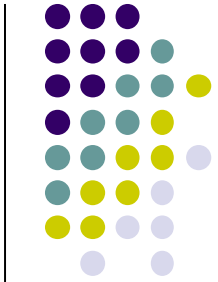
- Uno stream **viene connesso** ad un dispositivo di I/O tramite una **operazione di apertura**.
  - In particolare, quando *un file viene aperto*, è creato un oggetto ed uno stream viene associato all'oggetto
- Gli stream forniscono un **canale di comunicazione** tra un programma ed uno specifico file o dispositivo cui sono associati
- La connessione viene interrotta con **una operazione di chiusura**
- La ricezione di dati da un dispositivo di input è detta "estrazione da un flusso"
- La trasmissione di dati ad un dispositivo di uscita è derra "inserimento in un flusso"

File di testo e file binari

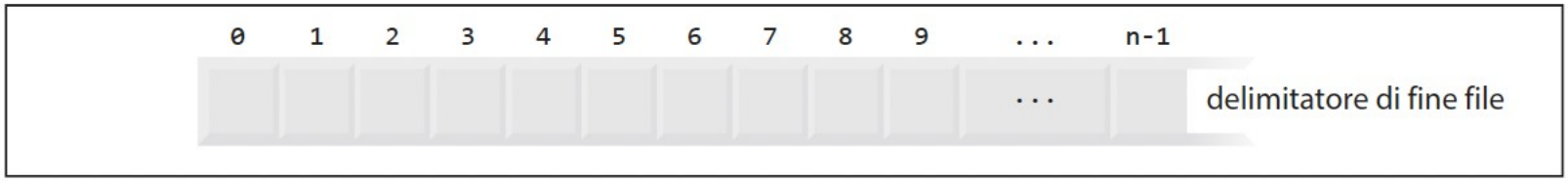


- I file possono essere di due tipi:
- ***file di testo***
  - Contengono una sequenza di linee di caratteri
  - I file di testo (e.g., con estensione `.txt`) possono essere letti dagli editor di testo.
- ***file binari***
  - Contengono una sequenza di byte
  - I file binari vengono letti da programmi in grado di capire il contenuto specifico del file.
- Un valore numerico in un file binario può essere utilizzato per fare calcoli, mentre il carattere 4 è semplicemente un carattere che può essere utilizzato in una stringa di testo, come "Antonia ha 14 anni".

File e stream in JAVA

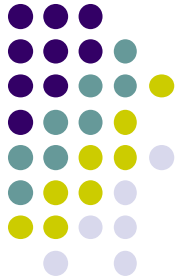


- Ogni sistema operativo ha un meccanismo per stabilire la fine di un file,
  - un delimitatore di fine file (end-of-file)
  - un contatore del numero totale di byte nel file memorizzato in una struttura dati gestita dal sistema.



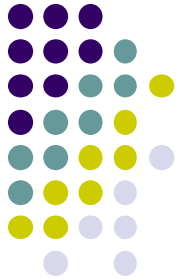
- Un programma Java che elabora uno stream di byte riceve semplicemente un'indicazione dal sistema operativo quando raggiunge la fine dello stream.
- Il programma non deve sapere come la piattaforma sottostante rappresenta i file o gli stream.
  - In alcuni casi, l'indicazione di fine file è riportata come un'eccezione;
  - in altri casi, è il valore di ritorno di un metodo chiamato su un oggetto che elabora lo stream.

Stream basati sui byte e stream basati sui caratteri



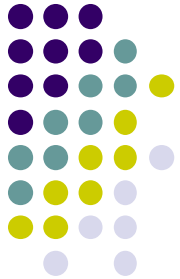
- Gli stream basati sui byte (byte-based stream) emettono e acquisiscono dati nel loro formato binario
  - un char corrisponde a due byte, un int a quattro byte, un double a otto byte, ecc.
- Gli stream basati sui caratteri (character-based stream) emettono e acquisiscono dati come sequenza di caratteri nella quale ogni carattere corrisponde a due byte;
  - il numero di byte per un valore dato dipende dal numero di caratteri di tale valore.
    - Per esempio, il valore 2000000000 richiede 20 byte (10 caratteri da due byte ciascuno), invece il valore 7 richiede solo due byte (1 carattere da due byte)

Oggetti stream per l'accesso ai file



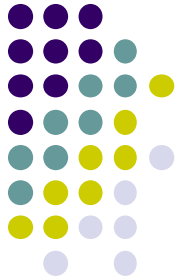
- Un programma Java ***apre*** un file creando un oggetto e associandogli uno stream di byte o di caratteri.
- Il costruttore dell'oggetto interagisce con il sistema operativo per ***aprire*** il file.
- Java può anche associare gli stream con diversi dispositivi.

Gli oggetti stream standard input, standard output e standard error



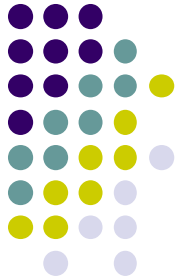
- Quando un programma Java inizia l'esecuzione, crea tre oggetti stream associati con i dispositivi: System.in, System.out e System.err.
- Solitamente
  - l'oggetto System.in consente a un programma di ricevere byte in input dalla tastiera;
  - l'oggetto System.out consente a un programma di scrivere dati in forma di caratteri sullo schermo;
  - l'oggetto System.err consente a un programma di scrivere sullo schermo messaggi dierrore basati su caratteri.
- Ciascuno di questi stream può essere rediretto.
  - Nel caso di System.in, questa possibilità consente al programma di leggere byte in input da una sorgente diversa.
  - Nel caso di System.out e System.err, questa possibilità consente di mandare l'output a una destinazione diversa, come un file su disco.
- La classe System mette a disposizione i metodi setIn, setOut e setErr per redirigere gli stream standard input, output ed error rispettivamente.

I package Java per l'accesso a File



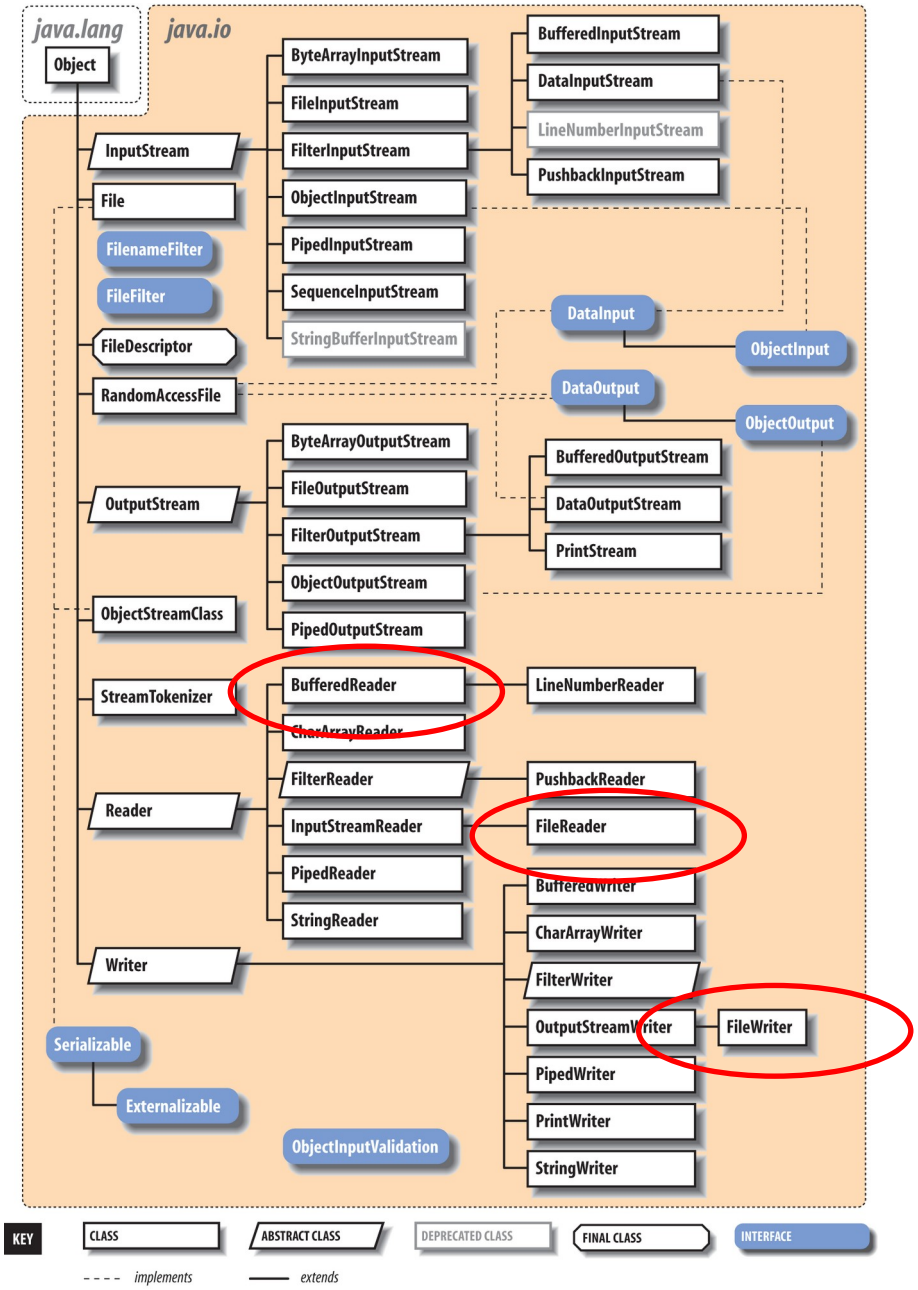
- I programmi Java eseguono elaborazioni basate su stream con classi e interfacce del package `java.io` e dei subpackage di `java.nio`.
  - le API New I/O (`java.nio`) di Java introdotte con Java SE 6 e da allora continuamente migliorate.
  - Ci sono anche altri package tra le API Java che contengono classi e interfacce basate su quelle di java.io e java.nio.
- A partire da Java 8 è stato introdotto un nuovo tipo di stream, utilizzato per collezioni di elementi (come array e ArrayList), al posto degli stream di byte

Lettura e scrittura di file di testo



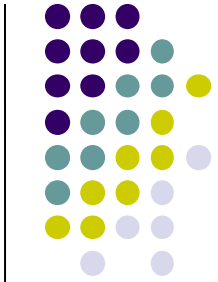
- Java mette a disposizione molti meccanismi per la lettura e la scrittura di file di testo.
- Uno dei meccanismi più utilizzato sfrutta le classi:
  - **`FileReader`**
  - **`BufferedReader`**
  - **`FileWriter`**

L'immagine fornisce una rappresentazione delle principali classi per la gestione dell'I/O in Java



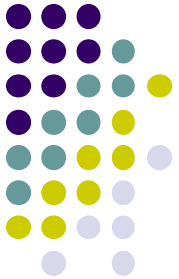
Source: https://www.javaboss.it/inputoutput-in-java/

FileReader



- La classe FileReader permette di leggere dati da file di testo
- Per utilizzare FileReader, è necessario importare la classe java.io.FileReader
- Per aprire in canale di comunicazione in lettura come detto è necessario istanziare un oggetto di tipo FileReader e collegarlo al nome di un file fisico

FileReader - Apertura



Source History

```
18          */
19          public static void main(String[] args) {
20
            unreported exception FileNotFoundException; must be caught or declared to be thrown
            ----
22          (Alt-Enter shows hints)
                filein = new FileReader("numeri.txt");
24          }
25
26      }
27
```

Constructor Detail

FileReader

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

Creates a new `FileReader`, given the name of the file to read from.

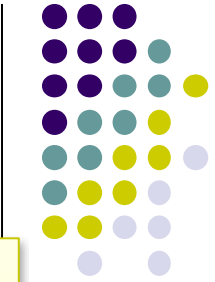
Parameters:

`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.

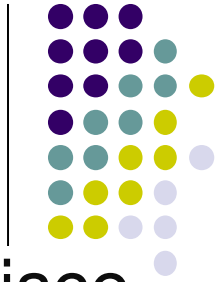
FileReader - Apertura



```
package fileexamples;
import java.io.*;

public class Fileexamples {

public static void main(String[] args) {

      FileReader filein;

      try {
        filein = new FileReader("numeri.txt");
        //... codice che utilizza lo stream
      }
      catch (FileNotFoundException e) {
           System.out.println("Errore nella lettura del file: " + e.getMessage());
      }
    }
}
```

Può essere istanziato anche specificando il percorso del file da leggere o una stringa

FileReader



- **Metodi di Lettura**: La classe FileReader fornisce diversi metodi per leggere dati dal file. Ad esempio:
  1. **int read()**: Legge un singolo carattere dal file e restituisce il valore del carattere letto (o -1 se si raggiunge la fine del file).
  2. **int read(char[] cbuf):** Legge una serie di caratteri dal file e li inserisce nel buffer specificato.
- **Chiusura del FileReader**: È buona pratica chiudere il FileReader dopo aver finito di utilizzarlo usando il metodo **void close()**
  - In alternativa può essere fatto automaticamente utilizzando il costrutto try-with-resources

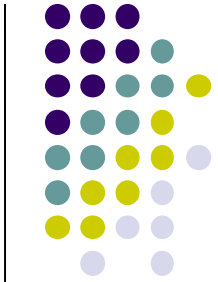
Chiusura - close()

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

      FileReader filein;
      String s = "numeri.txt";

      try {
         filein = new FileReader(s);
         //... codice che utilizza lo stream
          filein.close();
      }
      catch (FileNotFoundException e) {
            System.out.println("Errore nella lettura del file: " + e.getMessage());
        }
      catch (IOException e) {
          System.out.println("Errore nella chiusura del file: " + e.getMessage());
       }
    }
}
```

Lettura da FileReader

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

    FileReader filein;
    String s = "numeri.txt";

    try {
      filein = new FileReader(s);
      int cifra;
      while((cifra = filein.read())!=-1){
          System.out.print((char)cifra);
      }
      filein.close();
    }
    catch (FileNotFoundException e) {
         System.out.println("Errore nella lettura del file: " + e.getMessage());
     }
    catch (IOException e) {
        System.out.println("Errore nella chiusura del file: " + e.getMessage());
     }
  }
}
```

Osservazione

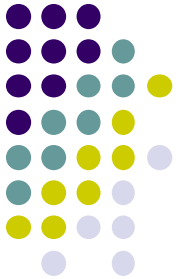


- Nell'esempio precedente viene letto un file carattere per carattere, e di quanto letto viene fatto l'eco a video
- Nel file numeri.txt ci sono solo numeri interi ma read legge un carattere e **ritorna il code point del carattere letto oppure -1**
- Quindi è necessario di convertire il valore di ritorno di nuovo in carattere prima di stamparlo
- Per evitare questa complicazione e anche per altri motivi si utilizza la classe BufferReader

BufferedReader

- BufferedReader fornisce metodi per leggere dati ***da uno stream di input*** (come un FileReader) in modo efficiente, leggendo più caratteri alla volta anziché uno per uno
- **Metodi di Lettura**: BufferedReader fornisce metodi come:
  - **readLine(),** che legge una riga intera dal flusso di input fino al carattere di fine riga (\n)
  - **read(char[] cbuf, int off, int len)** per leggere un numero specificato di caratteri in un buffer
- **Buffering dei Dati**: Utilizza un buffer interno per memorizzare i dati letti dal flusso di input. Questo significa che quando viene chiamata readLine() o altri metodi di lettura, il BufferedReader legge più dati contemporaneamente nel buffer. Questo riduce il numero di chiamate di I/O effettuate al sistema operativo, migliorando le prestazioni.
- **Chiusura Automatica**: È possibile utilizzare BufferedReader insieme al costrutto try-with-resources di

Esempio: caricamento interi da un file in un vettore



```
int v[]=new int[10];
int i;
FileReader f;
BufferedReader b;
String s;
try {
        f=new FileReader("numeri.txt");
        b=new BufferedReader(f);
        s=b.readLine();
        int n=Integer.parseInt(s);
        // lettura array da file
        for(i=0; i<n; i++) {
                s=b.readLine();
                if(s==null)
                      break;
                v[i]=Integer.parseInt(s);
        }
        System.out.println("Il veettore caricato da file è:");
        for(i=0; i<n;i++){
            System.out.println(v[i]);
        }
        f.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
```

Esempio: caricamento interi da un file in un vettore

BufferedReader è una classe che semplifica la lettura del testo da un flusso di input di caratteri. Memorizza i caratteri in un buffer per consentire una lettura efficiente dei dati di testo.

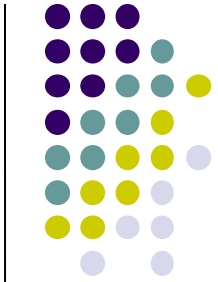
```
int v[]=new int[10];
int i;
FileReader f;
BufferedReader b;
String s;
try {
        f=new FileReader("numeri.txt");
        b=new BufferedReader(f);
        s=b.readLine();
        int n=Integer.parseInt(s);
        // lettura array da file
        for(i=0; i<n; i++) {
                s=b.readLine();
                if(s==null)
                      break;
                v[i]=Integer.parseInt(s);
        }
        System.out.println("Il veettore caricato da file è:");
        for(i=0; i<n;i++){
            System.out.println(v[i]);
        }
        f.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
```

Esempio: caricamento interi da un file in un vettore

Il costruttore di BufferedReader riceve in ingresso un flusso di input (come un file di caratteri) memorizzando così nel suo buffer un insieme di caratteri

```
int v[]=new int[10];
int i;
FileReader f;
BufferedReader b;
String s;
try {
        f=new FileReader("numeri.txt");
        b=new BufferedReader(f);
        s=b.readLine();
        int n=Integer.parseInt(s);
        // lettura array da file
        for(i=0; i<n; i++) {
                s=b.readLine();
                if(s==null)
                      break;
                v[i]=Integer.parseInt(s);
        }
        System.out.println("Il veettore caricato da file è:");
        for(i=0; i<n;i++){
            System.out.println(v[i]);
        }
        f.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
```

Esempio: caricamento interi da un file in un vettore



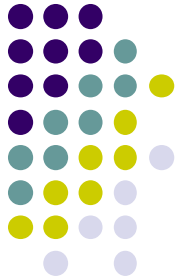
Il metodo readLine() della classe BufferedReader in Java viene utilizzato per leggere una riga di testo alla volta. La fine di una riga deve essere intesa con "\n" o "\r" o EOF.

```
int v[]=new int[10];
int i;
FileReader f;
BufferedReader b;
String s;
try {
        f=new FileReader("numeri.txt");
        b=new BufferedReader(f);
        s=b.readLine();
        int n=Integer.parseInt(s);
        // lettura array da file
        for(i=0; i<n; i++) {
                s=b.readLine();
                if(s==null)
                      break;
                v[i]=Integer.parseInt(s);
        }
        System.out.println("Il veettore caricato da file è:");
        for(i=0; i<n;i++){
            System.out.println(v[i]);
        }
        f.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
```

FileWriter

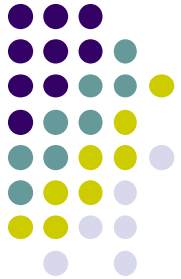
- La classe Filewriter permette di scrivere dati da file di testo
- Per utilizzare FileWriter, è necessario importare la classe java.io.FileWriter
- Per aprire in canale di comunicazione in scrittura come detto è necessario istanziare un oggetto di tipo FileWriter e collegarlo al nome di un file fisico
- Se il file non esiste viene creato
- Se il file esiste e contiene del testo, il testo verrà eliminato

FileWriter



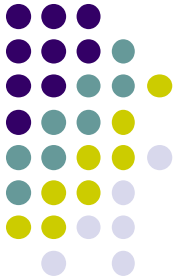
- **Gestione delle Eccezioni**: È importante gestire le eccezioni che potrebbero verificarsi durante la scrittura del file, ad esempio *se il file non può essere creato o se ci sono errori di I/O durante la scrittura.*
- **Metodi di Scrittura**: La classe FileWriter fornisce diversi metodi per scrivere dati nel file. Ad esempio:
  - **void write(String str):** Scrive una stringa nel file.
  - **void write(char[] cbuf):** Scrive un array di caratteri nel file.
  - **void write(int c):** Scrive un singolo carattere nel file.
- **Chiusura del FileWriter**: È buona pratica chiudere il FileWriter dopo aver finito di utilizzarlo. Questo può essere fatto anche automaticamente utilizzando il costrutto try-with-resources.

Esempio copia di file carattere per carattere



```
public static void main(String[] args) {
        try {
            //Parte che legge  carattere per carattere da un file di testo
            //dati.txt e li scrive in un nuovo file copia.txt
            // apre il file in lettura
            FileReader filein = new FileReader("dati.txt");
            // apre il file in scrittura
            FileWriter fileout = new FileWriter("copia.txt");
            int next = filein.read(); // legge il prossimo carattere
            while(next!=-1) {
                System.out.printf("%c", next);
                fileout.write((char)next);
                next = filein.read();
            }
            filein.close();
            fileout.close();

        } catch (IOException e) {
            System.out.println(e);
        }
        System.out.println();
        System.out.println("Fine Copia File !");
    }
```

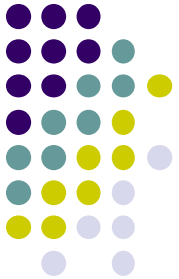
Esempio copia di file carattere per carattere



```
public static void main(String[] args) {
        try {
            //Parte che legge  carattere per carattere da un file di testo
            //dati.txt e li scrive in un nuovo file copia.txt
            // apre il file in lettura
            FileReader filein = new FileReader("dati.txt");
            // apre il file in scrittura
            FileWriter fileout = new FileWriter("copia.txt");
            int next = filein.read(); // legge il prossimo carattere
            while(next!=-1) {
                System.out.printf("%c
                fileout.write((char)n
                next = filein.read();
            }
            filein.close();
            fileout.close();

        } catch (IOException e) {
            System.out.println(e);
        }
        System.out.println();
        System.out.println("Fine Copia File !");
    }
```

- Il metodo read() legge e restituisce un singolo carattere o -1 se lo stream è terminato.
- Dopo la lettura si "punta automaticamente al carattere successivo"

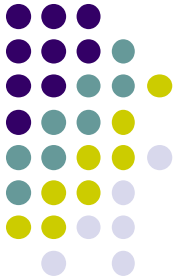
Esempio copia di file carattere per carattere



```
public static void main(String[] args) {
        try {
            //Parte che legge  caratt
            //dati.txt e li scrive i
            // apre il file in lettu
            FileReader filein = new FileReader("dati.txt");
            // apre il file in scrittura
            FileWriter fileout = new FileWriter("copia.txt");
            int next = filein.read(); // legge il prossimo carattere
            while(next!=-1) {
                System.out.printf("%c", next);
                fileout.write((char)next);
                next = filein.read();
            }
            filein.close();
            fileout.close();

        } catch (IOException e) {
            System.out.println(e);
        }
        System.out.println();
        System.out.println("Fine Copia File !");
    }
```

- Il metodo write di FileWriter scrive un singolo carattere nel file

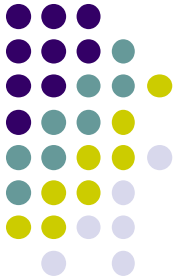
Esempio copia di file carattere per carattere



```
public static void main(String[] args) {
        try {
            //Parte che legge  carattere per carattere da un file di testo
            //dati.txt e li scrive in un nuovo file copia.txt
            // apre il file in lettura
            FileReader filein = new Fi
            // apre il file in scrittu
            FileWriter fileout = new F
            int next = filein.read();
            while(next!=-1) {
                System.out.printf("%c"
                fileout.write((char)ne
                next = filein.read();
            }
            filein.close();
            fileout.close();

        } catch (IOException e) {
            System.out.println(e);
        }
        System.out.println();
        System.out.println("Fine Copia File !");
    }
```

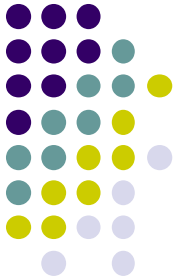
- Il metodo close() delle classi FileReader e FileWriter in Java sono utilizzati per chiudere rispettivamente il lettore e lo scrittore di file.
- Non possiamo più utilizzare FileReader e FileWriter dopo l'esecuzione del metodo close().
  - Questa procedura chiude il flusso e libera tutte le risorse di sistema associate.

Esempio di scrittura su file con System.out



```
public static void main(String[] args) {
        // Definiamo il nome del file
        String fileName = "output.txt";

        try {
            // Creiamo un nuovo PrintStream per il file
            PrintStream fileStream = new PrintStream(new FileOutputStream(fileName));

            // Salviamo l'attuale System.out in modo da poterlo ripristinare successivamente
            PrintStream consoleStream = System.out;

            // Reindirizziamo l'output di System.out su fileStream
            System.setOut(fileStream);

            // Ora tutto ciò che viene stampato su System.out verrà scritto direttamente sul file

            // Esempio di scrittura
            System.out.println("Questo è un esempio di scrittura su file utilizzando System.out.");

            // Ripristiniamo l'output di System.out sulla console
            System.setOut(consoleStream);

            System.out.println("Scrittura completata con successo!");
        } catch (IOException e) {
            // Gestione delle eccezioni in caso di errori di I/O
            System.out.println("Si è verificato un errore durante la scrittura nel file: " +
e.getMessage());
        }
    }
```

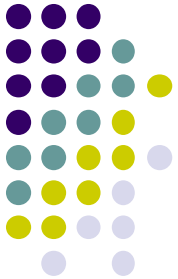
Esempio di scrittura su file con System.out



```
public static void main(String[] args) {
        // Definiamo il nome del file
        String fileName = "output.txt";

        try {
            // Creiamo un nuovo PrintStream per il file
            PrintStream fileStream = new PrintStream(new FileOutputStream(fileName));

            // Salviamo l'attuale Syste
            PrintStream consoleStream =

            // Reindirizziamo l'output
            System.setOut(fileStream);

            // Ora tutto ciò che viene                                          file

            // Esempio di scrittura
            System.out.println("Questo                                          ut.");

            // Ripristiniamo l'output c
            System.setOut(consoleStream

            System.out.println("Scrittu
        } catch (IOException e) {
            // Gestione delle eccezioni
            System.out.println("Si è ve
e.getMessage());
        }
    }
```

- FileOutputStream è una classe per gestire uno stream di output per scrivere dati su un File.
  - La disponibilità o la possibilità di creare un file dipende dalla piattaforma sottostante.
- Lo scopo generale della classe PrintStream è inviare informazioni a qualche stream.
- Poiché la variabile System.out è un oggetto PrintStream, invochiamo uno dei metodi di questa classe quando eseguiamo System.out.println().

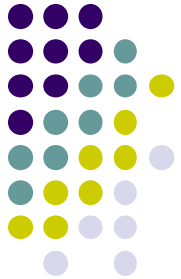
Esempio di scrittura su file con System.out



```
public static void main(String[] args) {
        // Definiamo il nome del file
        String fileName = "output.txt";

        try {
            // Creiamo un nuovo PrintStream per
            PrintStream fileStream = new PrintStream(new FileOutputStream(fileName));

            // Salviamo l'attuale System.out in modo da poterlo ripristinare successivamente
            PrintStream consoleStream = System.out;

            // Reindirizziamo l'output di System.out su fileStream
            System.setOut(fileStream);

            // Ora tutto ciò che viene stampato su System.out verrà scritto direttamente sul file

            // Esempio di scrittura
            System.out.println("Questo è un esempio di scrittura su file utilizzando System.out.");

            // Ripristiniamo l'output di System.out sulla console
            System.setOut(consoleStream);

            System.out.println("Scrittura completata con successo!");
        } catch (IOException e) {
            // Gestione delle eccezioni in caso di errori di I/O
            System.out.println("Si è verificato un errore durante la scrittura nel file: " +
e.getMessage());
        }
    }
```

- Ci serve per ripristinare in seguito

Riferimenti



- Programmare in Java
  - Cap. 15
    - §15.1, §15.2, §15.4