Input/Output in Java

Comunicare con il mondo

- Praticamente ogni programma ha la necessità di comunicare con il mondo esterno
  - Con l'utente attraverso tastiera e video
  - Con il file system per leggere e salvare dati
  - Con altre applicazioni sullo stesso computer
  - Con altre applicazioni su altri computer collegati in rete
  - Con dispositivi esterni attraverso porte seriali o USB
- Java gestisce tutti questi tipi di comunicazione in modo uniforme usando un unico strumento: lo stream

Input e Output

- Uno stream (in italiano flusso) è un canale di comunicazione attraverso cui passano dati in una sola direzione
- E' un "tubo" attraverso cui passano informazioni.
- Gli stream sono un'insieme di classi contenute nel package java.io
- Dal momento che gli stream sono monodirezionali avremo bisogno di:
  - Flussi di ingresso: input stream
  - Flussi di uscita: output stream

Sorgenti e destinazioni

- I dispositivi esterni possono essere
  - Sorgenti – per esempio la tastiera – a cui possiamo collegare solo stream di input
  - Destinazioni – per esempio il video – a cui possiamo collegare solo stream di output
  - Sia sorgenti che destinazioni – come i file o le connessioni di rete – a cui possiamo collegare –sia input stream (per leggere) che output stream (per scrivere).
- Attenzione: anche se un dispositivo è bidirezionale uno stream è sempre monodirezionale e quindi per comunicare contemporaneamente sia in scrittura che in lettura dobbiamo collegare due stream allo stesso dispositivo.

Byte e caratteri

- Esistono due misure di “tubi”:
  - stream di byte
  - stream di caratteri
- Java adotta infatti la codifica UNICODE che utilizza due byte (16 bit) per rappresentare un carattere
- Per operare quindi correttamente con i dispositivi o i file che trattano testo dovremo utilizzare stream di caratteri
- Per i dispositivi che trattano invece flussi di informazioni binarie utilizzeremo stream di byte

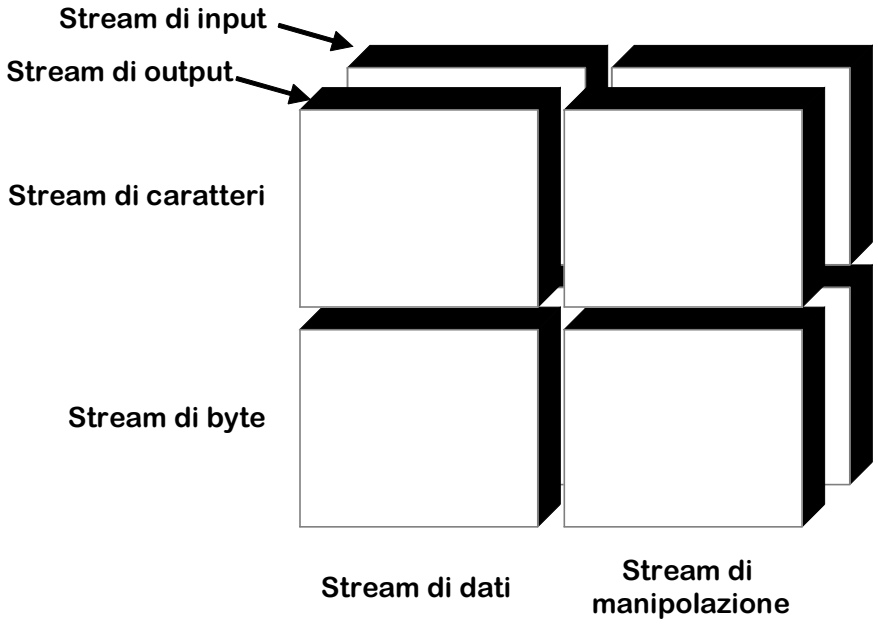
Stream di dati e stream di manipolazione

- Finora abbiamo parlato di stream di dati che, come abbiamo visto, hanno lo scopo di collegare un programma con una sorgente o una destinazione di dati
- Java però ci mette a disposizione anche un altro tipo di stream che hanno come obiettivo quello di fare una elaborare i dati in ingresso o in uscita
- Non si collegano direttamente ad una sorgente o destinazione d dati ma ad un altro stream e forniscono in uscita un contenuto informativo elaborato
- Anche gli stream di elaborazione sono di input o di output e possono trattare byte oppure caratteri

Criteri di classificazione

- Possiamo quindi classificare gli stream sulla base di tre criteri:
  - Direzione: input o output
  - Tipo di dati: byte o caratteri
  - Scopo: collegamento con una dispositivo/file o manipolazione di un altro stream
- Le tre classificazioni sono indipendenti (ortogonali) fra loro
- Ogni stream ho quindi una direzione, un tipo di dati trasportati e uno scopo

Schema di classificazione

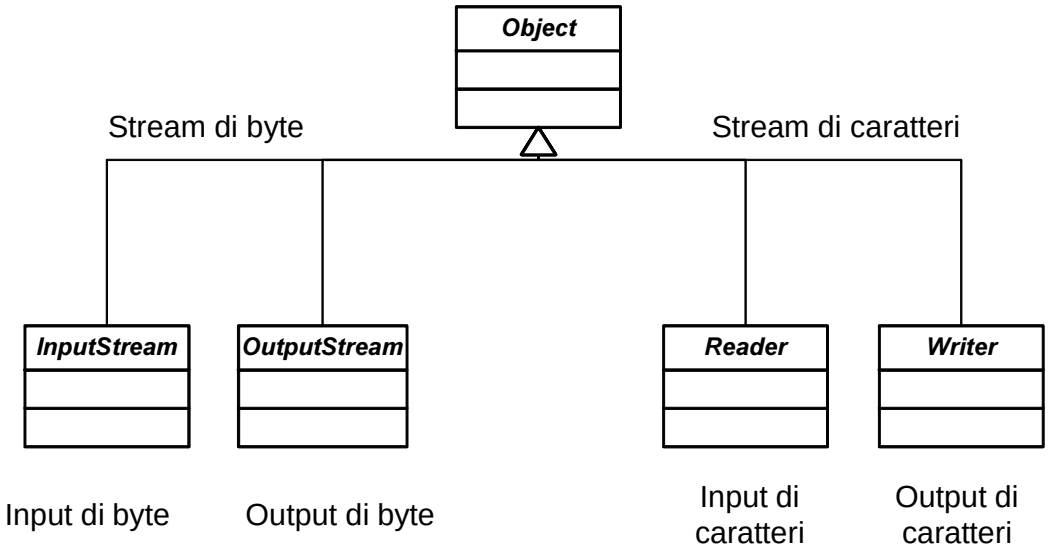


Un gioco di incastri

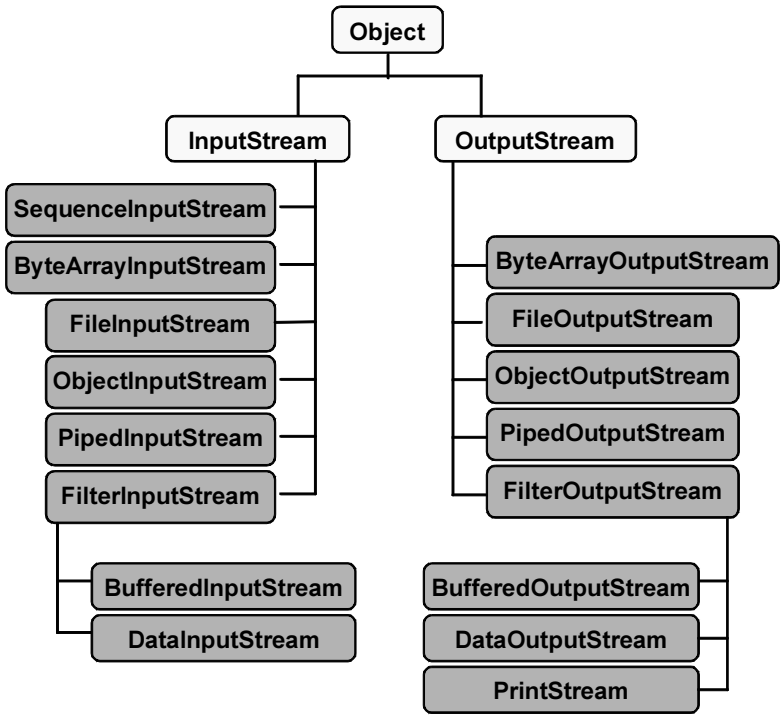
- Le classi stream sono state realizzate in modo da potersi incastrare una con l'altra
- Si può quindi partire con uno stream di dati e incastrare uno dopo l'altro un numero qualsiasi di stream di manipolazione in modo da ottenere il risultato desiderato
- E' un meccanismo molto flessibile e potente
- Inoltre, utilizzando l'ereditarietà, il sistema può essere anche esteso a piacimento

L'albero genealogico

- La gerarchia delle classi stream (contenute nel package java.io) rispecchia la classificazione appena esposta
- Abbiamo una prima suddivisione fra stream di caratteri e stream di byte e poi all'interno di ogni ramo tra stream di input e stream di output (sono tutte classi astratte)



La gerarchia degli stream di byte



InputStream

- E' il capostipite degli stream di input per i byte
- E' una classe astratta e definisce pochi metodi
- La sua definizione (semplificata) è:

```
package java.io
public abstract class InputStream
{
  public abstract int read()
    throws IOException;
  public int available()
    throws IOException
  { return 0; }
  public void close()
    throws IOException {}
}
```

Lettura di un byte

Numero di byte disponibili

Chiusura del canale

- read() è astratto e deve essere implementato in modo specifico dalla classi concrete
- N.B. Tutti i metodi possono generare eccezioni

OutputStream

- E' il capostipite degli stream di output per i byte
- E' una classe astratta e definisce pochi metodi
- La sua definizione (semplificata) è:

```
package java.io;
public abstract class OutputStream
{
  public abstract void write(int b)
    throws IOException;
  public void flush()
   throws IOException {}
  public void close()
   throws IOException {}
}
```

Scrittura di un byte

Forza l'emissione dei byte trasmessi

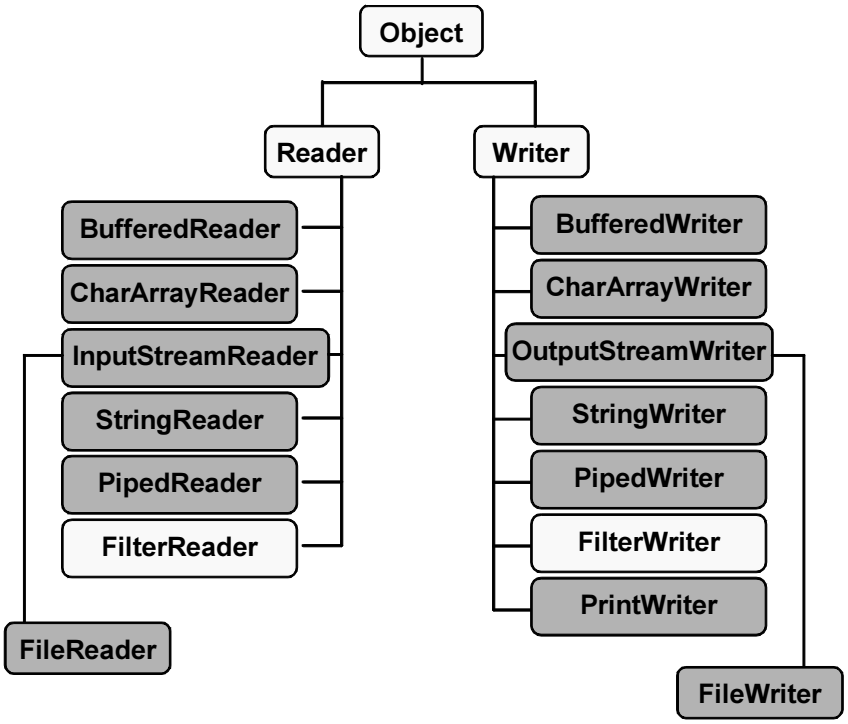
Chiusura del canale

- write() è astratto e deve essere implementato in modo specifico dalla classi concrete
- N.B. Tutti i metodi possono generare eccezioni

Gli stream di caratteri

- Le classi per l'I/O da stream di caratteri (Reader e Writer) sono più efficienti di quelle a byte
- Hanno nomi analoghi e struttura analoga
- Convertono correttamente la codifica UNICODE di Java in quella locale:
  - specifica del sistema operativo: Windows, Mac OS-X, Linux… (tipicamente ASCII)
  - e della lingua in uso (essenziale per l'internazionalizzazione)
- Per esempio gestiscono correttamente le lettere accentate e gli altri segni diacritici delle lingue europee

La gerarchia degli stream di caratteri



Reader

- E' il capostipite degli stream di input per i caratteri
- E' una classe astratta e definisce pochi metodi
- Una sua definizione (semplificata) è:

```
package java.io
public abstract class InputStream
{
  public abstract int read()
    throws IOException;
  public boolean ready()
    throws IOException
  { return 0; }
  public void close()
    throws IOException  { }
}
```

Lettura di un carattere

Dice se c'è qualcosa da leggere

Chiusura del canale

- read() restituisce un intero e quindi bisogna ricorrere ad un cast esplicito

Writer

- E' il capostipite degli stream di output per i caratteri
- E' una classe astratta e definisce pochi metodi
- Una sua definizione (semplificata) è:

```
package java.io;
public abstract class Writer
{
  public abstract void write(int c)
    throws IOException;
  public abstract void write(String str)
    throws IOException;
  public void flush()
   throws IOException {}
  public void close()
   throws IOException {}
}
```

Scrittura di un carattere

Scrittura di una stringa

Forza l'emissione dei byte trasmessi

Chiusura del canale

- Esistono più versioni di write() (overloading)

I/O Standard

- Esistono due stream standard definiti nella classe System: System.in e System.out
- Sono attributi statici e quindi sono sempre disponibili
- Gestiscono l'input da tastiera e l'output su video
- Attenzione: purtroppo per ragioni storiche (in Java 1.0 non c'erano gli stream di caratteri), sono stream di byte e non di caratteri
- In particolare:
  - System.in è di tipo InputStream (punta effettivamente ad un'istanza di una sottoclasse concreta) e quindi fornisce solo i servizi base
  - System.out è di tipo PrintStream e mette a disposizione i metodi print() e println() che consentono di scrivere a video qualunque tipo di dato

Gestione della tastiera: problemi

- System.in è molto rudimentale e non consente di trattare in modo semplice è corretto l'input da tastiera
- Infatti:
  - Essendo uno stream di byte non gestisce correttamente le lettere accentate
  - Non possiede metodi per leggere comodamente un'intera stringa
- Fortunatamente il meccanismo degli "incastri" di Java ci permette di risolvere in maniera efficace questi problemi.
- Per farlo useremo due classi che discendono da Reader: InputStreamReader e BufferedReader
- Sono entrambe stream di manipolazione

Gestione tastiera: da byte a caratteri

- Innanzitutto risolviamo i problemi legati al fatto che System.in è uno stream di byte
- InputStreamReader è una sorta di adattatore: converte uno stream di byte in uno stream di caratteri:
- Il suo costruttore è definito in questo modo:

```
public InputStreamReader(InputStream in)
```

- Grazie al subtyping può quindi "agganciarsi" ad un qualunque discendente di InputStream, quindi a tutti gli strem di input a byte,
- Per eseguire l'adattamento scriveremo:

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

- In questo modo possiamo utilizzare isr per leggere singoli caratteri da tastiera con un gestione corretta dei caratteri speciali (lettere accentate, umlaut ecc.)

Gestione tastiera: leggere le stringhe

- Vediamo ora come fare per leggere le stringhe
- BufferedReader è un discendente di Reader che aggiunge un metodo che ci consente di leggere una stringa dalla tastiera:
- `public String readLine()`
- E' quindi uno stream di manipolazione a caratteri
- Il costruttore è definito in questo modo:
- `public BufferedReader(Reader in)`
- Possiamo quindi agganciarlo a qualunque stream di caratteri.
- Per completare la nostra "conduttura" scriveremo quindi:

```
BufferedReader kbd =
 new BufferedReader(isr);
```


Gestione tastiera: soluzione completa

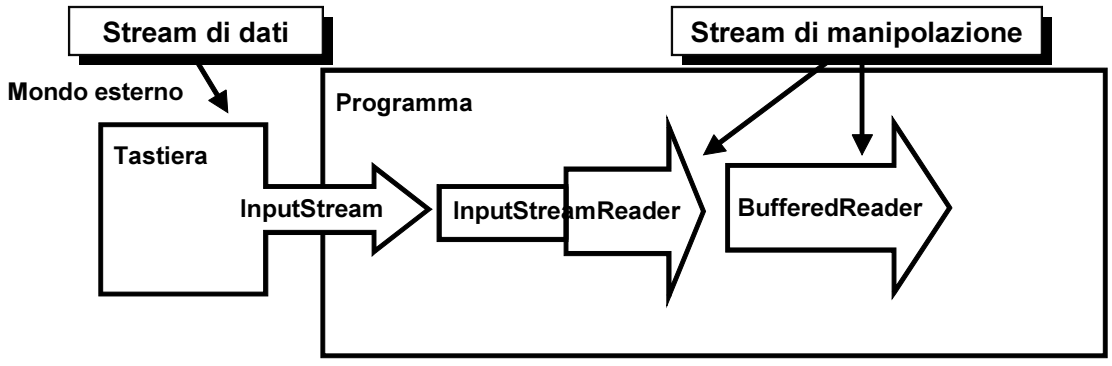
- Per gestire correttamente la tastiera useremo quindi una sequenza di istruzioni di questo tipo:

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

- Oppure in forma sintetica:

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

- Abbiamo quindi realizzato la nostra "conduttura":



Gestione del video

- System.out è già sufficiente per gestire un output di tipo semplice: print() e println() forniscono i servizi necessari
- E' uno stream di byte ma non crea particolari problemi.
- Tuttavia volendo possiamo utilizzare una tecnica simile a quella utilizzata per la tastiera
- E' sufficiente usare un solo stream di manipolazione - PrintWriter - che svolge anche la funzione di adattamento.
- Definisce infatti un costruttore di questo tipo:

```
public PrintWriter(OutputStream out)
```

- E mette a disposizione i metodi print() e println() per i vari tipi di dati da stampare

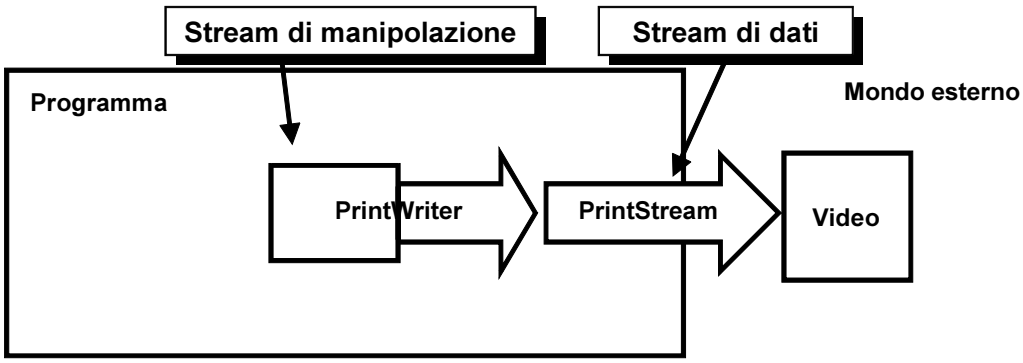
Gestione del video: soluzione completa

- Potremo quindi scrivere:

```
PrintWriter video = new PrintWriter(System.out);
```

- E utilizzarlo nello stesso modo con cui useremmo System.out

```
video.println(12);
video.println("Ciao");
video.println(13,56);
```



File di testo

- Possiamo leggere e scrivere file di testo utilizzando stream di caratteri
- In particolare:
  - La classe FileReader (derivata da Reader) ci permette di leggere un file di testo
  - La classe FileWriter (derivata da Writer) ci permette di scrivere in un file di testo
- Sono entrambi stream di dati: il loro scopo è quindi quello di creare un collegamento con un dispositivo esterno.

Lettura di un file di testo - Esempio

- Vediamo come si viene gestita la lettura di un file di testo con un esempio
- Supponiamo di voler leggere un file di testo (inventory.dat) che contiene l'inventario di una cartoleria.
- Ogni riga del file è un prodotto e per ogni prodotto abbiamo nome, quantità e prezzo unitario, separati da spazi:

```
Quaderno 14 1.35
Matita 132 0.32
Penna 58 0.92
Gomma 28 1.17
Temperino 25 1.75
Colla 409 3.12
Astuccio 142 5.08
```

La classe InventoryItem

- I dati letti vengono messi in un array di oggetti di classe InventoryItem definita così:

```
public class InventoryItem
{
  private String name;
  private int units;
  private float price;

  public InventoryItem(String nm, int num, float pr)
  {
    name = nm; units = num; price = pr;
  }
  public String toString()
  {
    return name + ": " + units + " a euro " + price;
  }
}
```


Accesso e lettura del file

- Per accedere al file possiamo utilizzare la classe FileReader
- Il costruttore di questa classe prende come parametro il nome del file da leggere e lo apre in lettura
- FileReader è però uno stream di dati e offre solo le funzionalità base: lettura a caratteri singoli
- Sappiamo però come risolvere questo problema: ricorriamo allo stream di manipolazione BufferedReader e lo agganciamo al FileReader:

```
FileReader fr = new FileReader("inventory.dat");
BufferedReader inFile = new BufferedReader(fr);
```

- Possiamo quindi utilizzare `inFile.readLine()` per leggere le righe del file
- Quando il file termina `readLine()` restituisce `null`

StringTokenizer

- Ci rimane però un problema: all'interno di ogni riga abbiamo più informazioni separate da spazi e quindi dobbiamo scomporla
- La classe StringTokenizer, inclusa nel package java.util svolge proprio questo compito
- Il costruttore prende come parametro la stringa da scomporre e con il metodo nextToken() possiamo estrarre le singole sottostringhe e convertirle:

```
...
tokenizer = new StringTokenizer (line);
name = tokenizer.nextToken();
units = Integer.parseInt (tokenizer.nextToken());
price = Float.parseFloat (tokenizer.nextToken());
...
```


L'esempio completo - 1

- N.B. La lettura del file è sotto gestione di eccezioni

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

public class CheckInventory
{
  public static void main (String[] args)
  {
    String line, name;
    int units, count = 0;
    float price;

    InventoryItem[] items = new  InventoryItem[100];
    StringTokenizer tokenizer;
```

L'esempio completo - 2

```
try
{
  FileReader fr = new FileReader("inventory.dat");
  BufferedReader inFile = new BufferedReader(fr);
  line = inFile.readLine();
  while (line != null)
  {
    tokenizer = new StringTokenizer(line);
    name = tokenizer.nextToken();
    try
    {
      units = Integer.parseInt(tokenizer.nextToken());
      price = Float.parseFloat (tokenizer.nextToken());
      items[count++] = new InventoryItem(name,units,price);
    }
    catch (NumberFormatException e)
    { System.out.println("Error in input. Line:"+line);}
    line = inFile.readLine();
  }
```


L'esempio completo - 3

```
      inFile.close();

      // Scrive a video i dati letti
      for (int scan=0; scan<count; scan++)
        System.out.println(items[scan]);
    }

    // Gestione delle eccezioni in cascata

    catch (FileNotFoundException e)
    {
      System.out.println("File " + file + " not found.");
    }
    catch (IOException e)
    {
      System.out.println(e);
    }
  }
}
```

Scrittura un file di testo

- Vediamo con un esempio come si scrive in un file di testo
- Il programma scrive su un file la tavola pitagorica
- Usiamo un oggetto di classe FileWriter
- File Writer però è uno stream di dati e fornisce solo le funzionalità base
- Procediamo quindi come al solito agganciando uno stream di manipolazione – PrintWriter – che consente di scrivere agevolmente righe di testo
- In questo esempio non gestiamo le eccezioni e quindi siamo obbligati a dichiarare che main() può emettere eccezioni di tipo IOException

Esempio completo

```
import java.io.*;

public class Tabelline
{
  public static void main (String[] args) throws IOException
  {
    FileWriter fw = new FileWriter("tabelline.txt");
    PrintWriter outFile = new PrintWriter(fw);
    for (int i=1; i<=10; i++)
    {
      for (int j=1; j<=10; j++)
      {
       outFile.print((i*j)+" ");
      }
      outFile.println();
    }
    outFile.close();
  }
}
```