UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3
e
Laboratorio di Programmazione 3

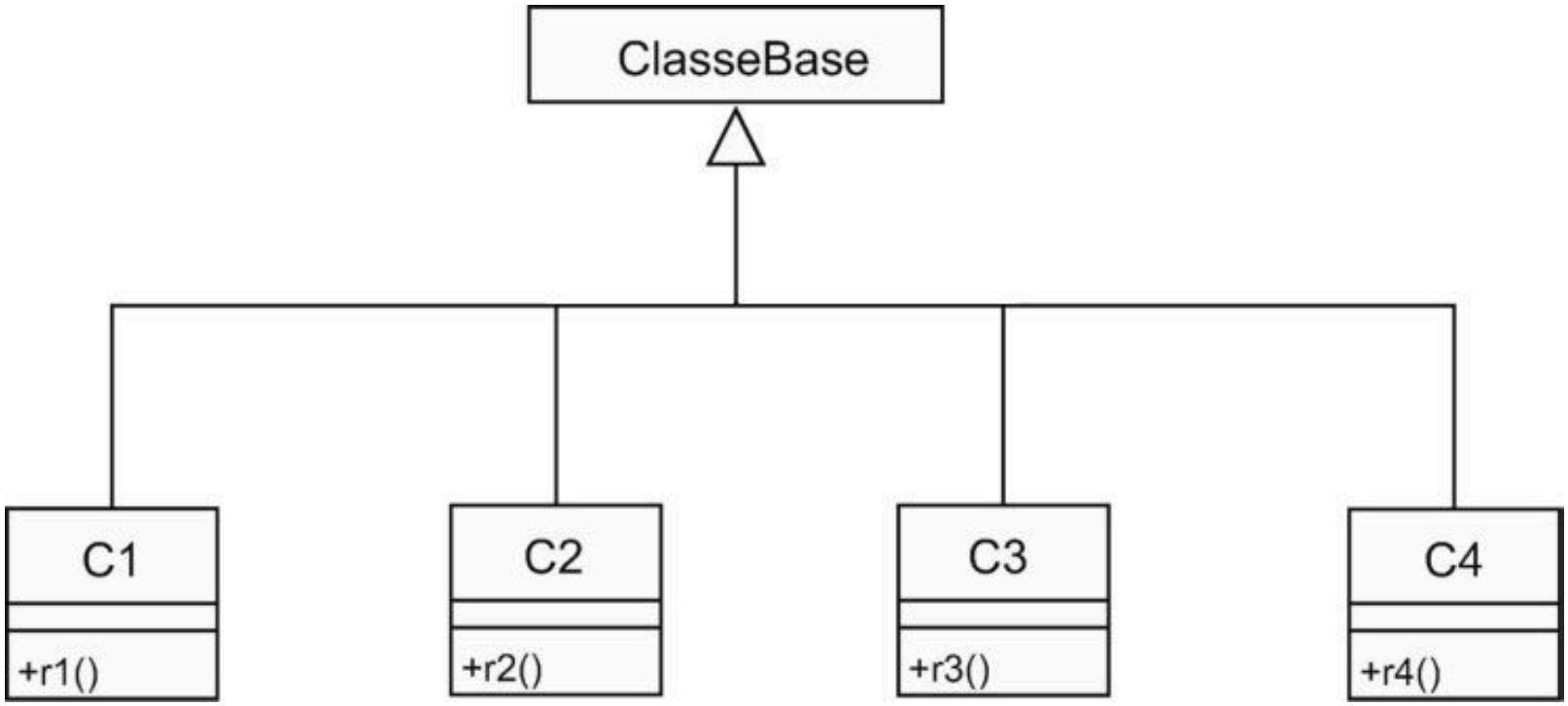
Input-Output

Angelo Ciaramella

Introduzione

- gerarchia delle classi di `java.io`
  - modello di classi definito dal pattern `Decorator`
    - relazione tra classi che rappresenta un'alternativa dinamica alla statica ereditarietà
    - aggiungere responsabilità addizionali agli oggetti al runtime

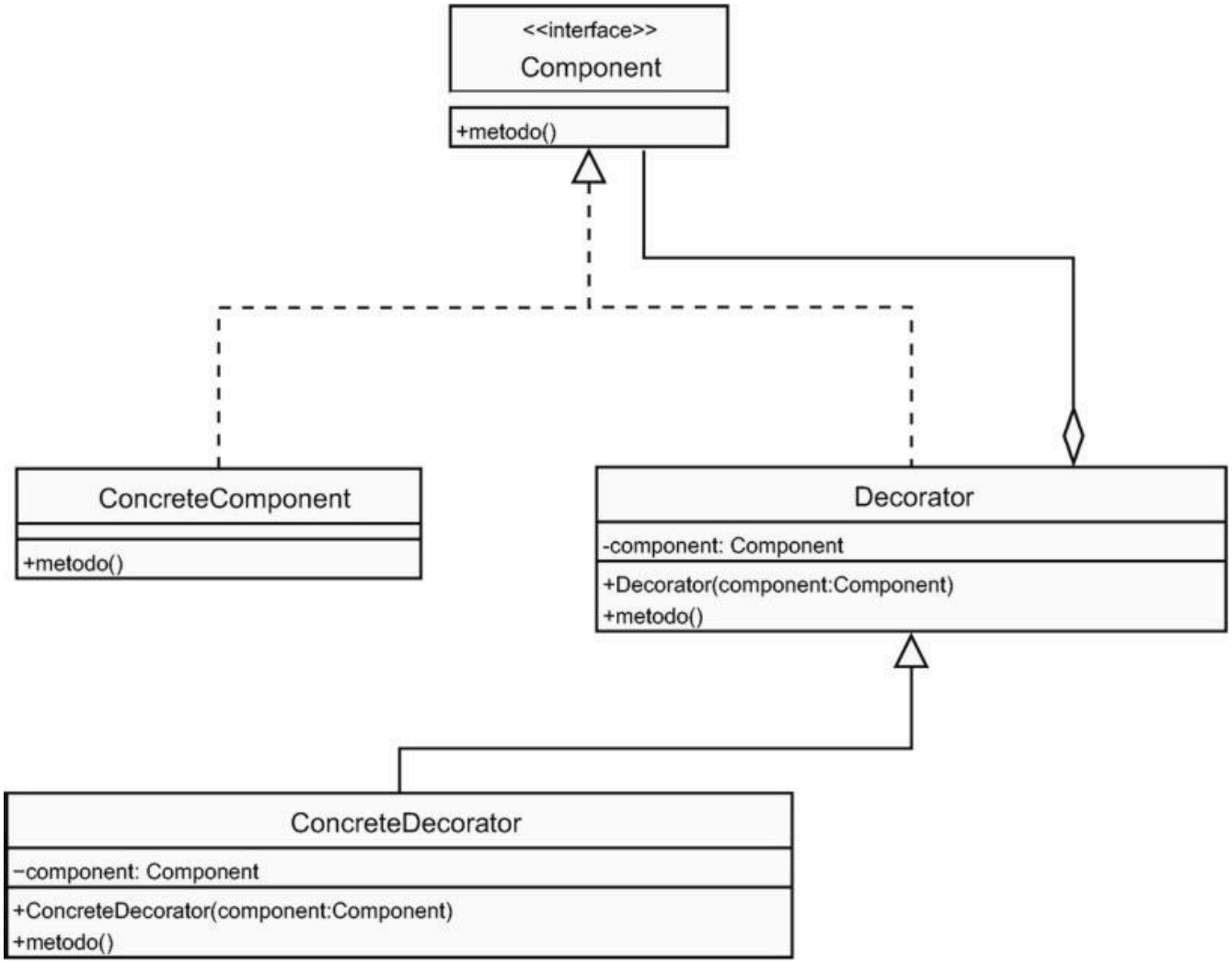
Decorator



Responsabilità

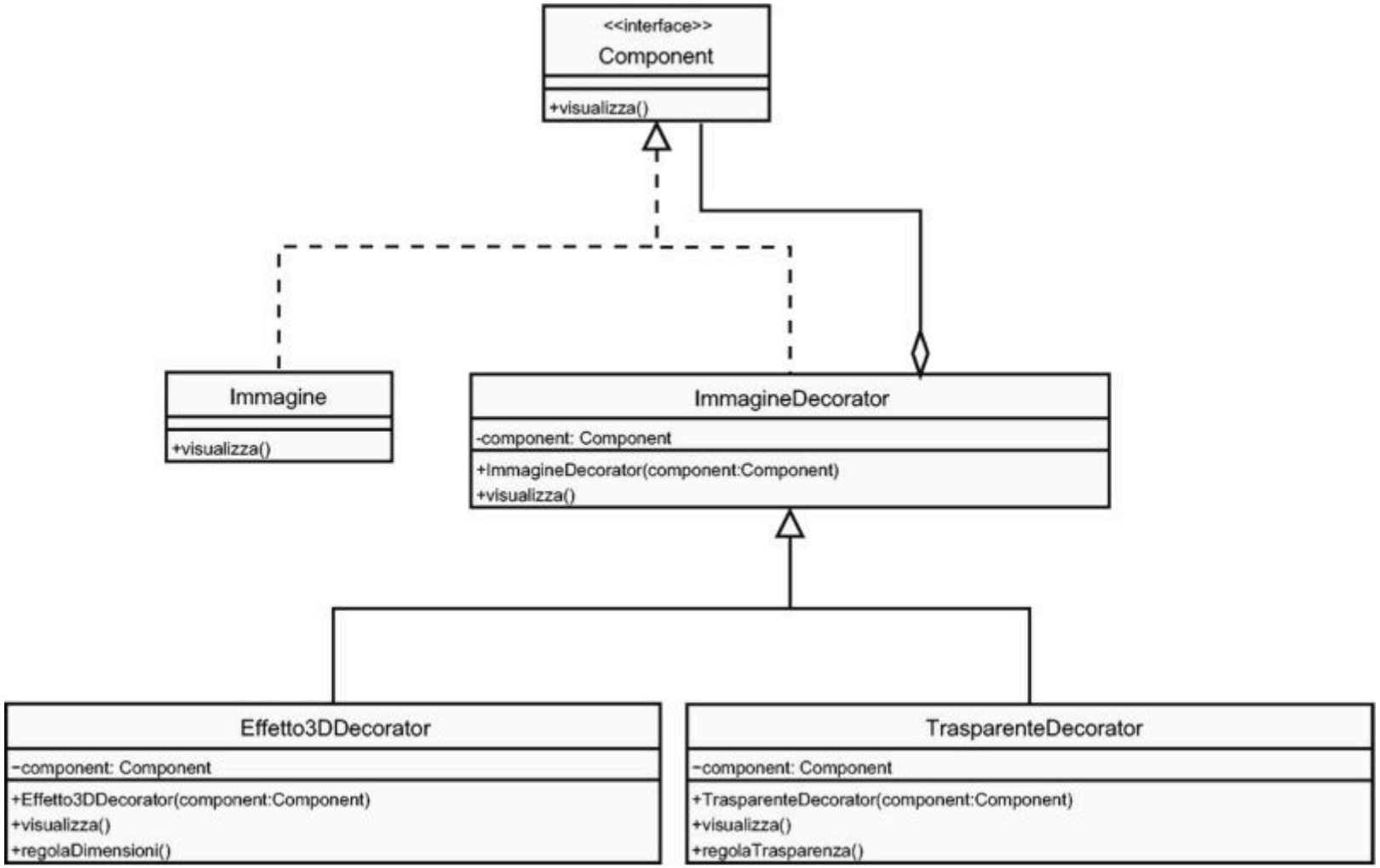
Ereditarietà. Potrebbe servire anche creare classi che hanno più di una di queste responsabilità

Decorator



Modello del pattern `Decorator`

Decorator

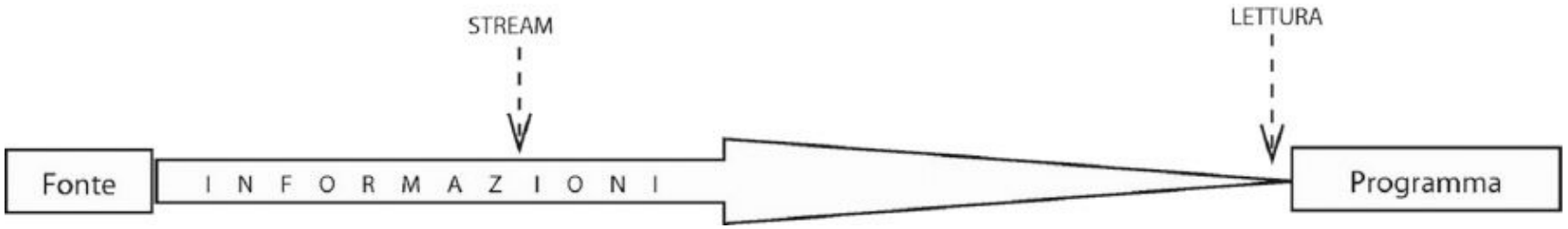


Esempio di implementazione del pattern `Decorator`

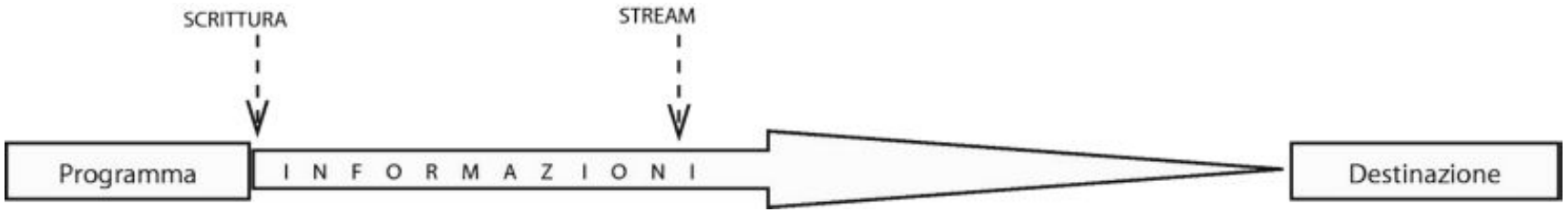
Decorator

- Il pattern `Decorator` permette
  - realizzare qualsiasi tipo di comunicazione con fonti di destinazioni esterne usando un limitato numero di classi

Stream



Rappresentazione grafica di input

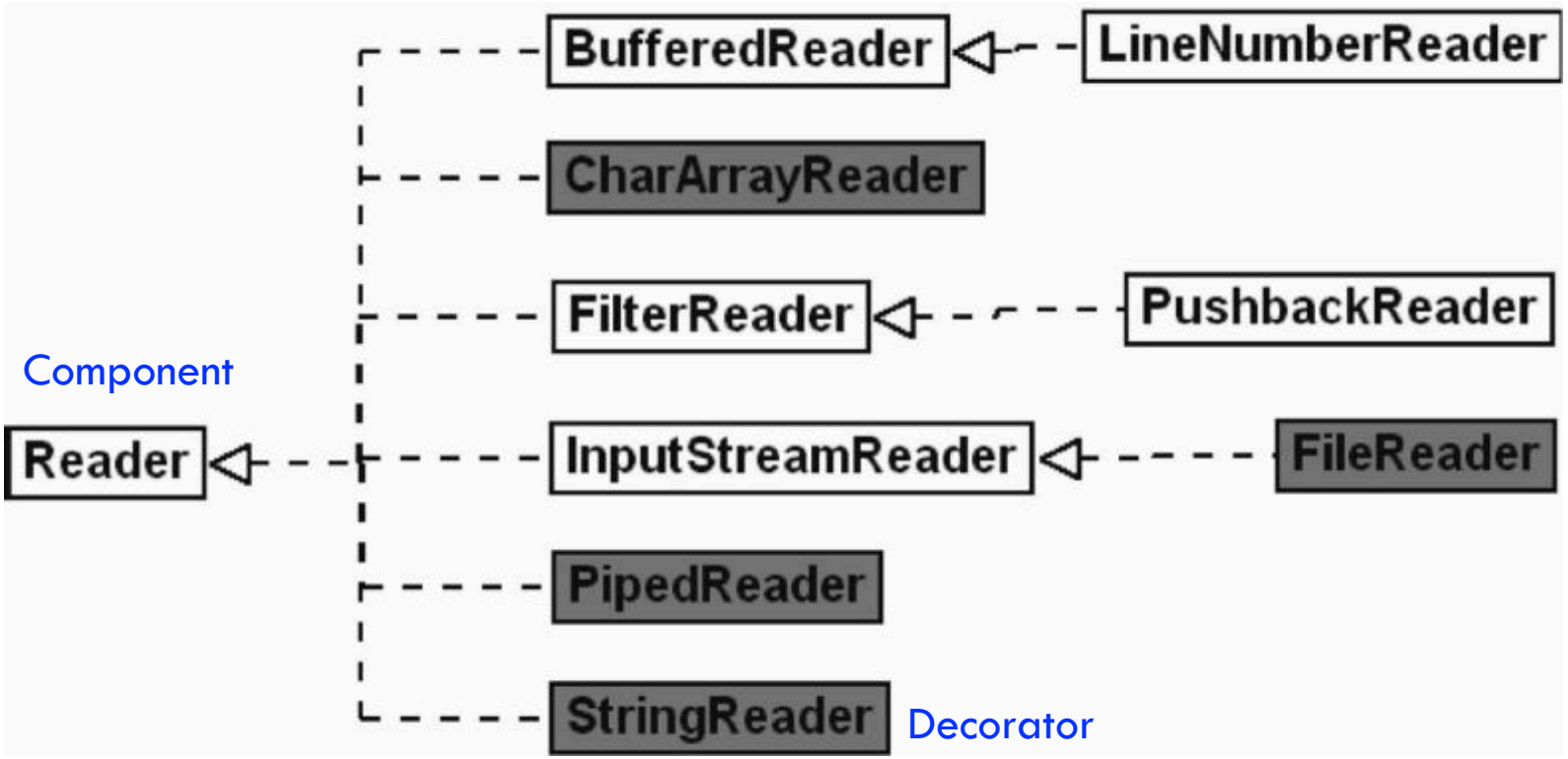


Rappresentazione grafica di output

Character Stream

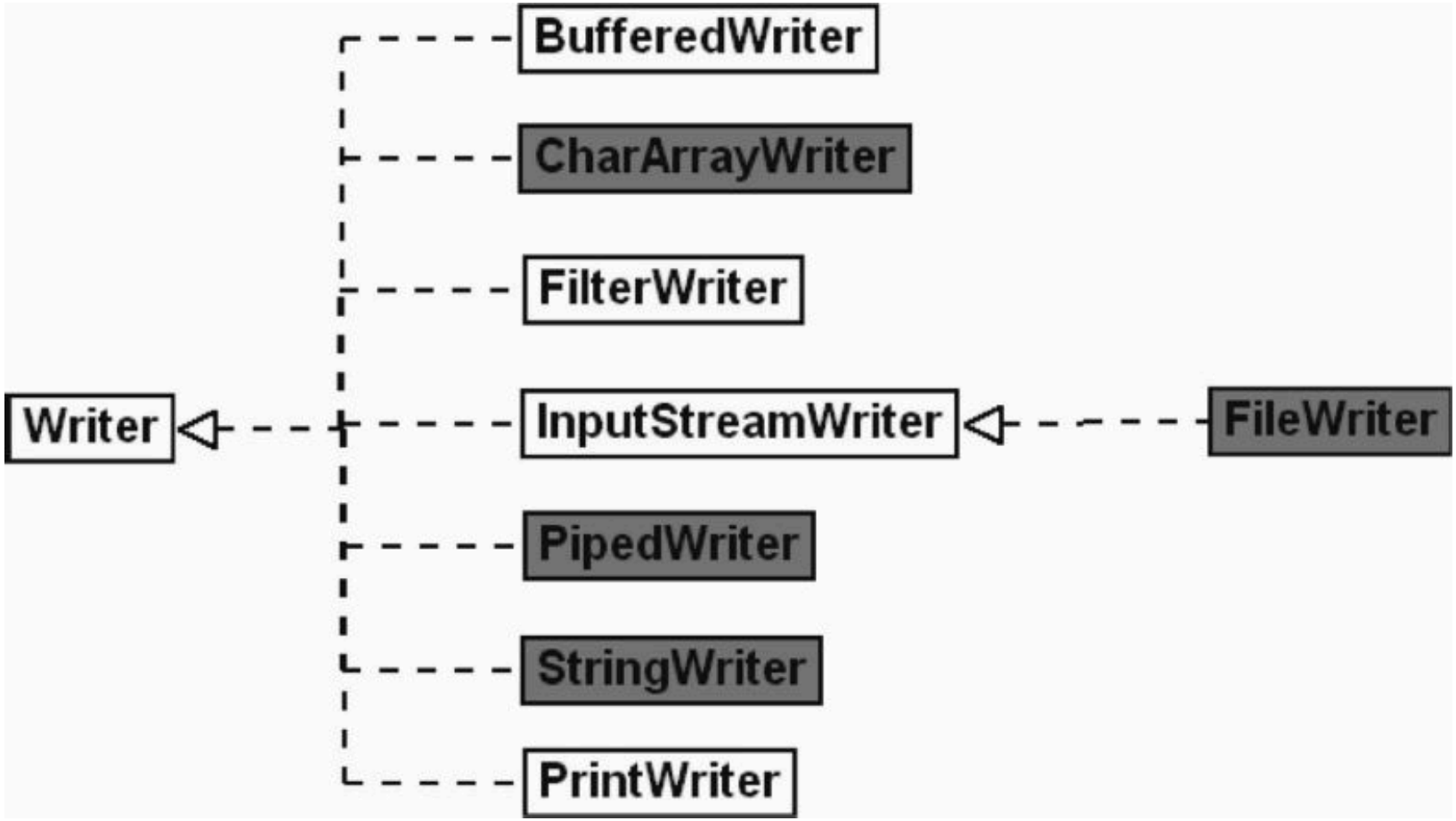
- `Reader` e `Writer` sono le due superclassi astratte per i flussi di caratteri
  - dividono i dati in 16 bit ognuno, compatibili con il tipo `char` di Java

Character Stream



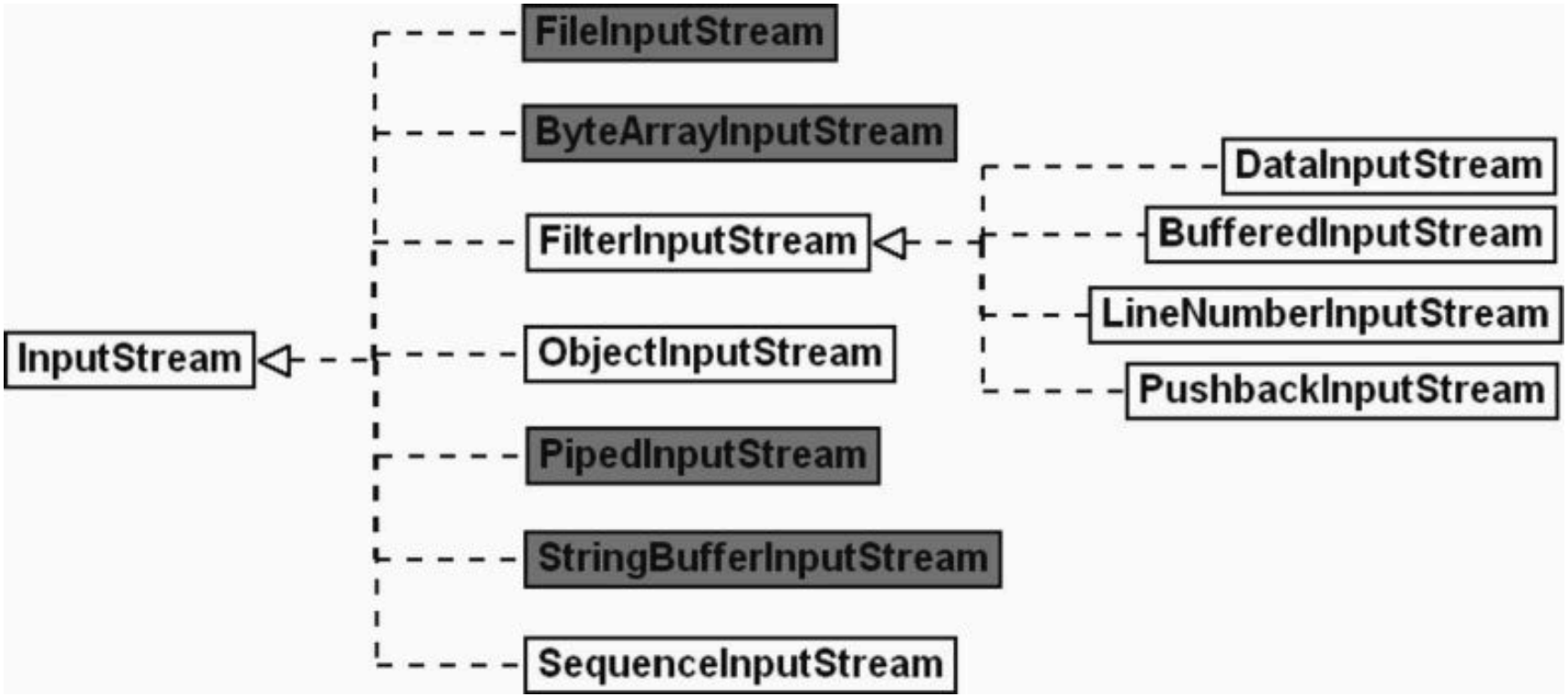
Gerarchia di Reader

Character Stream



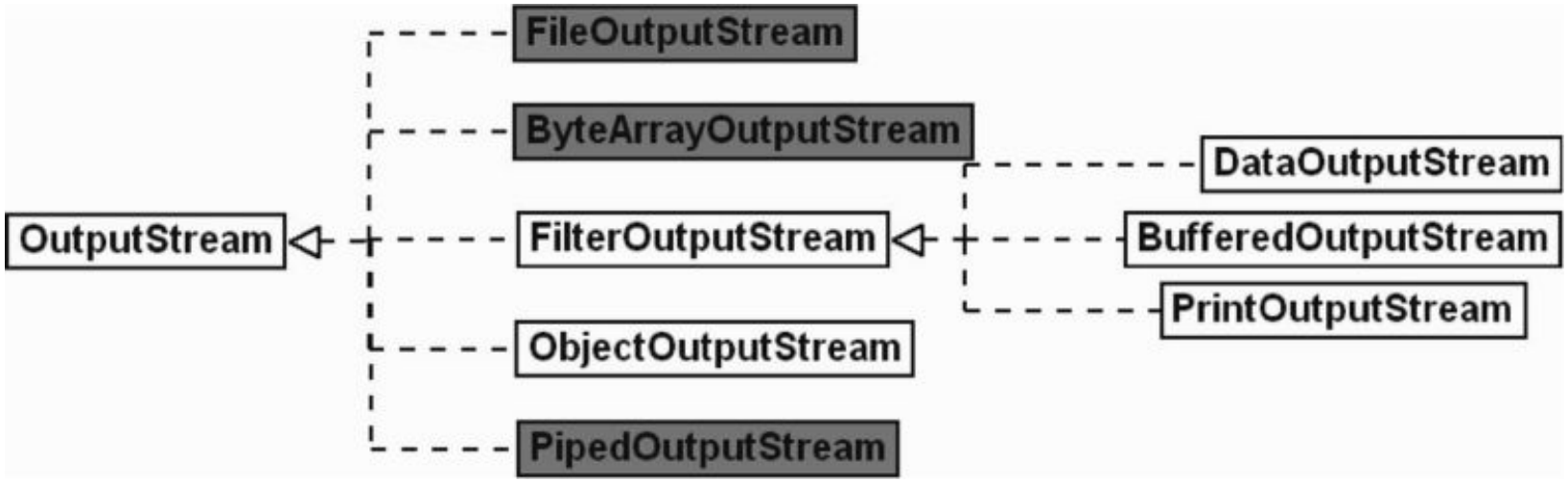
Gerarchia di Writer

Byte Stream



Gerarchia di InputStream (lettura di array di byte)

Byte Stream



Gerarchia di OutputStream (scrittura di array di byte)

Lettura da tastiera

```
import java.io.*;
public class KeyboardInput {
  public static void main (String args[]) throws IOException {
        String stringa = null;
        System.out.println("Digita qualcosa e premi invio..." +
        "\nPer terminare il programma digitare \"fine\"");
        try (InputStreamReader ir = new
        InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(ir)) {
        stringa = in.readLine();
        while ( stringa != null ) {
        if (stringa.equals("fine")) {
        System.out.println("Programma terminato");
        break;
    }
    System.out.println("Hai scritto: "+ stringa);
    stringa = in.readLine();
    }
   }
  }
}
```

InputStreamReader consente di trasformare byte in char

Gestione dei File

```
File dir = new File("/usr", "local");
File file = new File(dir, "Abc.java");
File dir2 = new File("C:\\directory");
File file2 = new File(dir2, "Abc.java");


File file = new File(".." + File.pathSeparator +
"Abc.java");
```

Costante statica

File di oggetti

- Leggere e scrivere oggetti

```
Coin c = . . .;
   ObjectOutputStream out = new ObjectOutputStream(new

FileOutputStream("coins.dat"));
   out.writeObject(c);
```

```
ObjectInputStream in = new ObjectInputStream(new

FileInputStream("coins.dat"));
   Coin c = (Coin)in.readObject(); //cast per Object
```

```
    // scrittura
      ArrayList a = new ArrayList();
   out.writeObject(a);

   //lettura
   ArrayList a = (ArrayList)in.readObject();
```


Serializable

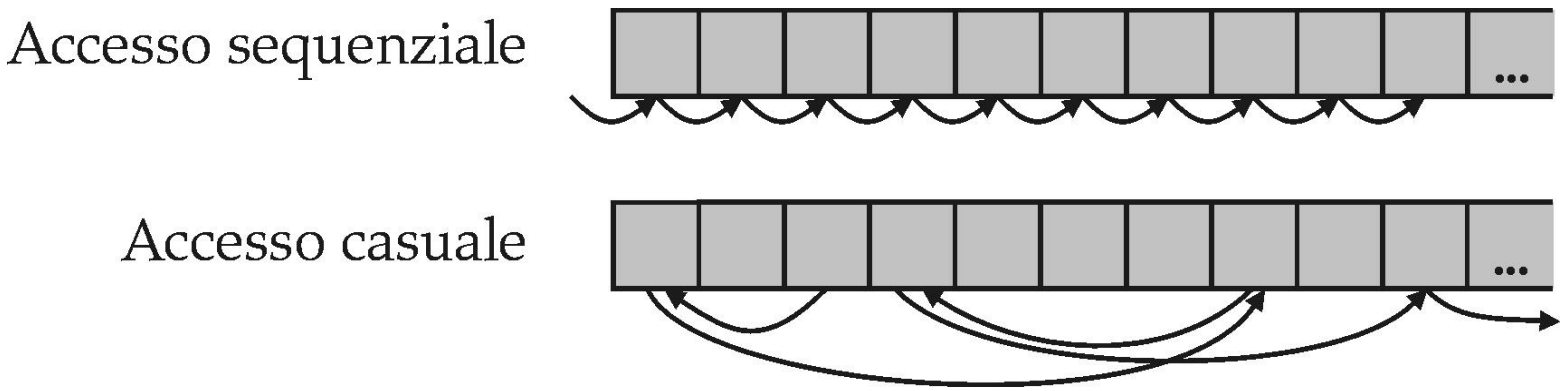
- Per inserire oggetti di una classe in un flusso di oggetti dobbiamo implementare `Serializable`

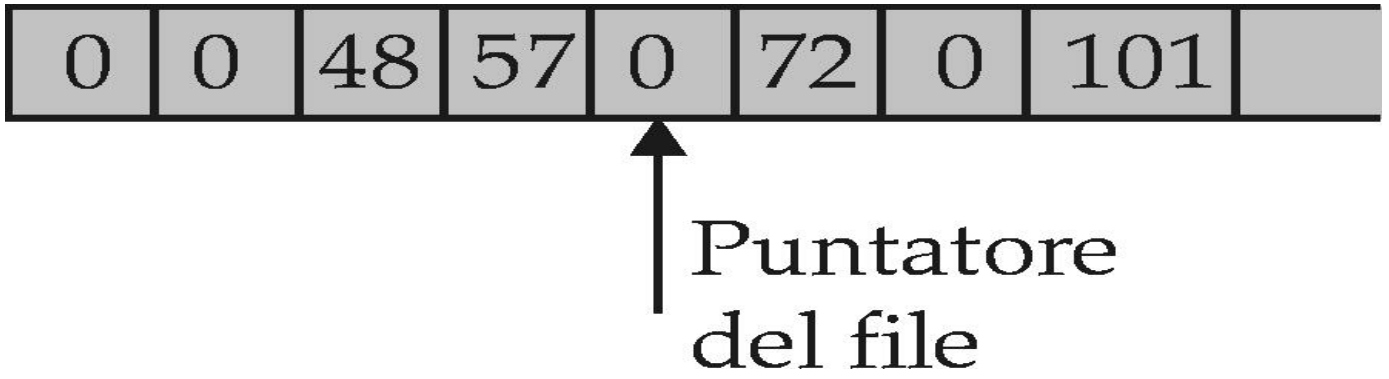
```
class Coin implements Serializable
   {. . .}
```

Codice di riferimento

```
PurseTest.java; Coin.java; Purse.java
```


Accesso casuale





Serializable

```
RandomAccessFile f = new RandomAccessFile("ac.dat","rw");
```

Apertura di un file ad accesso casuale

```
f.seek(n)
```

Spostamento del puntatore al byte n

```
n=f.getFilePointer();
```

Posizione corrente del puntatore

Codice di riferimento

```
BankDataTest.java; BankData.java; SavingsAccount.java;
BankAccount.java
```


NIO 2.0

- Con Java 7 sono state introdotte novità sulla gestione dei file
  - `java.nio`
  - NIO consente operazioni più complesse che la libreria IO non permette

Classe Files

```
Charset charset = Charset.forName("UTF-8");
String contenutoDelFile = "Ciao";
Path path =
Paths.get("C:\\Users\\user\\Desktop\\test.txt");
try (BufferedWriter writer =
Files.newBufferedWriter(path, charset)) {
writer.write(contenutoDelFile, 0,
contenutoDelFile.length());
} catch (IOException x) {
System.err.format("IOException: %s%n", x);
}
```

Esempio di utilizzo della classe Files

Classe Files

```
Path directory = Paths.get("C:\\Users\\user\\Desktop");
Path file =
Paths.get("C:\\Users\\user\\Desktop\\test.txt");
System.out.println("Files.exists(directory): "
+ Files.exists(directory));
System.out.println("Files.isReadable(file): " +
Files.isReadable(file));
System.out.println("Files.isWritable(file): " +
Files.isWritable(file));
System.out.println("Files.isExecutable(file): "
+ Files.isExecutable(file));
System.out.println("Files.isSameFile(file): " +
Files.isSameFile(directory, file));
```

Esempio di utilizzo della classe Files