



JAVA

Fondamenti di Informatica L-B

Esercitazione n°4

Java: I/O, Costruttori ed altro!



JAVA



A.A. 2007/08

Tutor: Barbara Pettazzoni

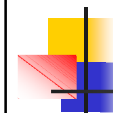
barbara.pettazzoni@studio.unibo.it



JAVA



JAVA



Attenzione ai riferimenti!

Qual'è la differenza fra questi due listati?

```
Contatore c1 = new Contatore();  
Contatore c2 = c1;  
if(c1==c2)  
    System.out.println("Si");  
else  
    System.out.println("No");
```

Questo stampa

Si

```
Contatore c1 = new Contatore();  
Contatore c2 = new Contatore();  
if(c1==c2)  
    System.out.println("Si");  
else  
    System.out.println("No");
```

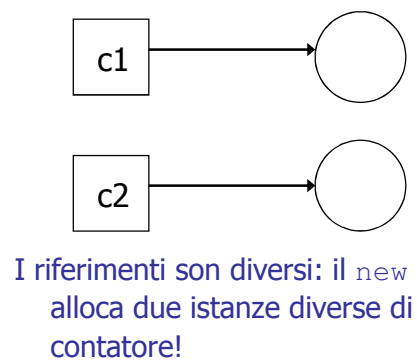
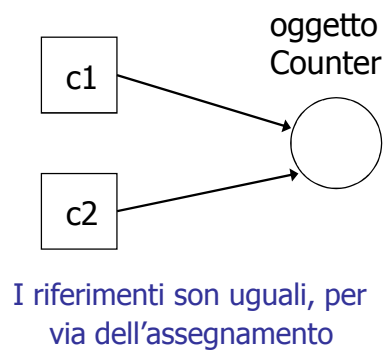
Questo stampa

No



Attenzione ai riferimenti!

- L'operatore di uguaglianza verifica l'uguaglianza fra riferimenti, nel caso di oggetti.



3



Attenzione ai riferimenti!

- Riprendiamo il primo esempio, estendendolo:

```
Contatore c1 = new Contatore();  
Contatore c2 = c1;  
c2.inc();  
System.out.println("c1 vale: " + c1.leggi());
```

- Il riferimento è condiviso; questo significa che ciò che viene stampato è:

c1 vale: 1

- Le "variabili" c1 e c2 permettono di accedere/modificare la stessa istanza della classe Contatore.

4



Attenzione ai riferimenti!

- Per quello detto finora quindi...cosa restituisce questo listato?

```
int c1 = 5;  
int c2 = 5;  
if(c1==c2)  
    System.out.println("Si");  
else  
    System.out.println("No");
```

- Ovviamente, in questo caso viene stampato Si. Questo perché si tratta di un confronto fra interi, cioè un tipo primitivo.
- Questi non sono oggetti! L'operatore d'uguaglianza confronta il valore.

5



Uguaglianza fra oggetti

- Ma se volessi valutare se due oggetti hanno lo stesso valore?
- Devo definire il metodo

```
public boolean equals(Object o)
```

- Es:

```
public class Contatore  
{  
    private int val;  
    ...  
    public boolean equals(Contatore c)  
    {return (this.val == c.val);}  
    ...  
}
```

6



Uguaglianza fra oggetti

- Il codice diventa il seguente:

```
Contatore c1 = new Contatore();  
Contatore c2 = new Contatore();  
if (c1.equals(c2))  
    System.out.println("Si");  
else  
    System.out.println("No");
```

- Il risultato ora sarà Sí
- All'interno del metodo **equals** possiamo definire come vogliamo noi il criterio per stabilire l'uguaglianza fra due oggetti
- Per esempio, potevamo dire che due contatori erano uguali se avevano lo stesso valore massimo!

7



I costruttori

- Molti errori di programmazione sono causati dalla mancata corretta inizializzazione delle variabili.
- Nella programmazione ad oggetti, il **costruttore** é il metodo che automatizza l'inizializzazione dell'oggetto:
 1. Non viene mai chiamato esplicitamente dall'utente, ma attraverso l'operatore **new**
 2. Viene invocato automaticamente dal sistema ogni volta che si crea un nuovo oggetto.

Un costruttore:

- Ha lo stesso nome della classe
- Non ha tipo di ritorno (nemmeno void!). Infatti, il suo compito é inizializzare l'oggetto
- Può non essere unico.

8



I costruttori della classe Counter

```
public class Counter
{
    private int value, maxValue;

    public Counter(){value = 0;}

    public Counter(int val) {value = val;}

    public Counter(int val, int max)
    {
        value = val;
        maxValue = max;
    }

    ...
}
```

9



Ancora sui costruttori

- Il costruttore senza parametri é detto **di default**: anche se non lo specificassimo, verrebbe fornito automaticamente dal sistema! É il costruttore che viene usato quando non si passano i parametri.

In realtà, però, tale costruttore fa pochissimo: *conviene che lo ridefiniamo noi!*

- *Domanda: e se il costruttore avesse un parametro con lo stesso nome di un attributo della classe?
Si può risolvere questa ambiguità? Come?*

10



La parola chiave **this**

- L'ambiguità si risolve utilizzando questa parola chiave:

```
public Counter(int valore)
{this.valore = valore;}
```

- La parola chiave **this** identifica *l'istanza corrente, cioè l'oggetto su cui stiamo lavorando!*
- In generale, è sottintesa: scrivere **this.valore** o **valore** è la stessa cosa.
- È *necessaria* nei casi di omonimia appena descritti!

11



Input da tastiera/file

- Per comodità, per poter leggere da un file o dalla tastiera useremo il package **fiji** (sul sito trovate la documentazione e il jar corrispondente).
- Qui potete trovare una classe **Lettore** che vi permette di leggere stringhe, interi,... leggere la documentazione è fondamentale!
- Esempio di utilizzo:

```
import fiji.io.*;
import java.io.*;

public class TestLettura
{
    public static void main(String args[])
    {
        Lettore input = new Lettore();
        String s = input.leggiLinea();
        System.out.println("Ho letto: " + s);
    }
}
```

12



La classe Lettore

- Come si può leggere dalla Javadoc:

Constructor Summary

<code>Lettore()</code>	Crea un Lettore corrispondente al flusso standard di ingresso, <code>System.in</code> (cioè la tastiera).
<code>Lettore(java.io.Reader rdr)</code>	Crea un Lettore corrispondente al flusso ingresso pre-esistente <code>in</code> .
<code>Lettore(java.lang.String s)</code>	Crea un Lettore per leggere dalla stringa <code>s</code> .

- Ma cosa è un flusso (anche detto **stream**)?

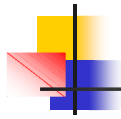
13



Il package java.io

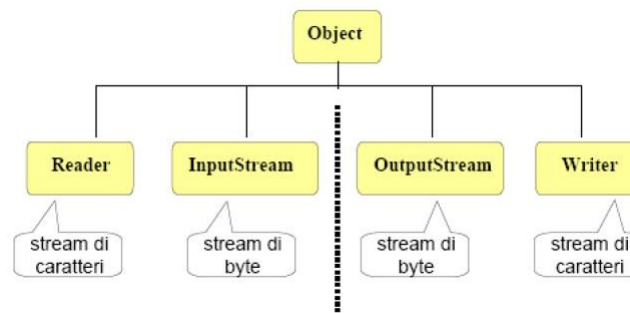
- In Java si usano gli **stream** per poter leggere dai file.
- Uno stream è un *canale di comunicazione* :
 1. monodirezionale (o è input o output)
 2. ad uso generale
 3. che trasferisce byte (o anche caratteri)
- Il package java.io distingue fra *stream di byte* (come i file binari in C) e *stream di caratteri* (come i file testuali in C, presenti a partire da java 1.1)

14



Il package java.io

- Questi stream si traducono poi in una serie di **classi di base astratte**; queste sono alla base della *tassonomia delle classi di stream realizzate da Java*.



15



Leggiamo da file...

```
public class TestLetturaFile
{
    public static void main(String args[])
    {
        try
        {
            Lettore input =
                new Lettore(new FileReader("prova.txt"));
            while(!input.eof())
            {
                String s = input.leggiLinea();
                System.out.println("Letto: " + s);
            }
            System.out.println("Terminata la lettura");
        }
        catch(Exception e){e.printStackTrace();}
    }
}
```

16



Scriviamo su file...

```
public class TestScritturaFile
{
    public static void main(String args[])
    {
        try
        {
            FileWriter fw = new FileWriter("mio.txt");
            BufferedWriter bw = new BufferedWriter(fw);
            int x=2, y=4;
            String s = "x vale: " + Integer.toString(x);
            fw.write(s);
            s = "y vale: " + y;
            fw.write(s);
            fw.flush();
            fw.close(); } Importante!!!
        }
        catch(IOException e) {e.printStackTrace();}
    }
}
```

17



Esercizi

- Dal sito scaricate il file per l'esercitazione, "trama.txt": scrivete un programma in grado di contare il numero di linee, parole e caratteri presenti nel file. Inoltre, contate tutte le occorrenze della parola "Anello".
I risultati ottenuti dovrete salvarli in un apposito file chiamato "risultati.txt", che quindi dovrà contenere 4 linee.
- Suggerimento: date un'occhiata alla classe *StringTokenizer* nella javadoc (si trova in java.util) per poter ottenere le parole.
- Cosa dovrete modificare se volessimo fare un programma "general purpose" (in grado di leggere da un qualunque file e indicare l'occorrenza di una qualunque parola decisa dall'utente)?
- Leggete la Javadoc del package java.io, e osservate il numero di possibili classi per gestire gli stream...

18



Una possibile soluzione (1)

```
import java.io.*;
import fi.ji.io.*;
import java.util.*;

public class ContaJava
{
    public static void main(String args[])
    {
        Contatore contaLinee = new Contatore();
        Contatore contaParole = new Contatore();
        Contatore contaCaratteri = new Contatore();
        Contatore contaAnello = new Contatore();
        try
        {
            FileReader fr = new FileReader("trama.txt");
            Lettore input = new Lettore(fr);
            ...
        }
    }
}
```

19



Una possibile soluzione (2)

```
...
while(!input.eof())
{
    String linea = input.leggiLinea();
    contaLinee.incrementa();
    for(int i=0; i < linea.length(); i++)
        contaCaratteri.incrementa();
    StringTokenizer tokenizer =
        new StringTokenizer(linea, " ");
    while(tokenizer.hasMoreTokens())
    {
        String parola = tokenizer.nextToken();
        if(parola.equals("Anello"))
            contaAnello.incrementa();
        contaParole.incrementa();
    }
}
...
```

20



Una possibile soluzione (3)

```
FileWriter fw = new FileWriter("risultati.txt");
BufferedWriter writer = new BufferedWriter(fw);
writer.write("Il file trama.txt ha " +
            contaLinee.getValore() + " linee");
writer.newLine();
writer.write("Il file trama.txt ha " +
            contaParole.getValore() + " parole");
writer.newLine();
writer.write("Il file trama.txt ha " +
            contaCaratteri.getValore() + " caratteri");
writer.newLine();
writer.write("Il file trama.txt ha " +
            contaAnello.getValore() +
            " occorrenze della parola Anello");
writer.flush();
writer.close();
catch (Exception e) {e.printStackTrace();}
}
```

21



ContaJava general purpose (1)

- Si potrebbe costruire un nuovo Lettore, che legge due stringhe di input da parte dell'utente (file e parola da cercare)

```
...
Lettore tastiera = new Lettore();
System.out.print("Inserisci il nome del file: ");
String file = tastiera.leggiLinea();
System.out.print("Inserisci la parola da cercare: ");
String cerca = tastiera.leggiLinea();
FileReader fr = new FileReader(file);
Lettore input = new Lettore(fr);
...
if (parola.equals(cerca))
    contaAnello.incrementa();
...
```

22



ContaJava general purpose (2)

- Oppure, si potrebbe lanciare il programma come
`java programma nomefile paroladacercare`
per cui i primi due parametri del main sono i nostri dati!

```
...
String file = args[0];
String cerca = args[1];
FileReader fr = new FileReader(file);
Lettore input = new Lettore(fr);
...
if(parola.equals(cerca))
    contaAnello.incrementa();
...
```

23



Manca qualcosa?

- ...Sì! La classe Contatore non é dotata di costruttore, quindi non é stata inizializzata correttamente.
- A default, ***gli interi sono a 0, e i booleani a false!*** Questo significa che il metodo `incrementa` non funziona.
- Come definire il costruttore di default?

```
public class Contatore
{
    private int value, maxValue;
    private boolean errore;

    public Contatore(){
        value = 0; maxValue = Integer.MAX_VALUE;
        errore = false;
    }
    ...
}
```

24



Compilare ed eseguire (1)

- A differenza delle altre volte, per leggere da file dobbiamo utilizzare il package **fiji**
- Questo non é un package di default fornito da Java, e quindi non é presente nel nostro classpath
- La compilazione dará errore, inevitabilmente!
- Come fare?
- Prima soluzione: sul sito son presenti le [istruzioni](#) su come sistemare il classpath in maniera corretta anche per il package fiji!

25



Compilare ed eseguire (2)

- Oppure, potete indicargli in maniera dinamica dove andare a cercare il jar e le classe annesse...
- Supponiamo che il file **fiji.jar** si trovi dove si trova **ContaJava.java**
- Compilazione:

```
javac -cp ./fiji.jar;. ContaJava.java
```
- Esecuzione:

```
javac -cp ./fiji.jar;. ContaJava
```

26