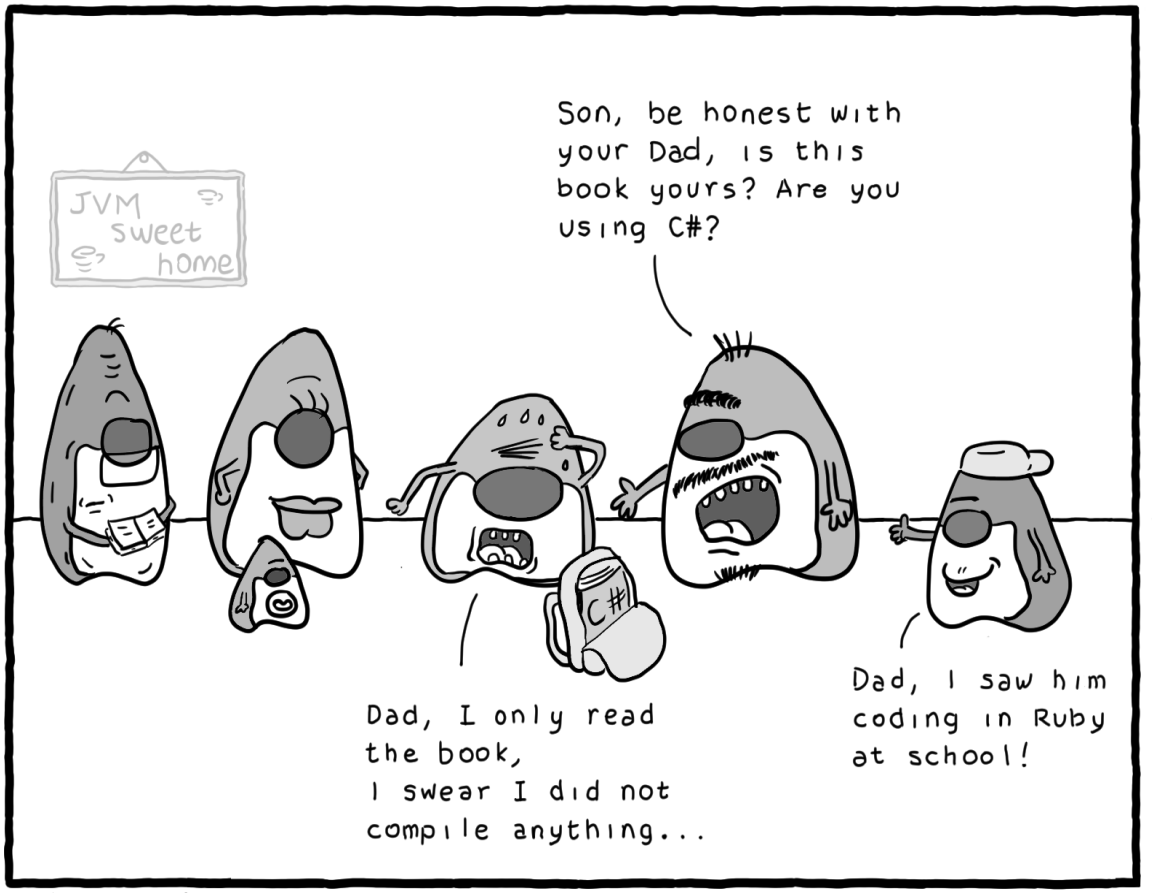
Java primi passi



Prof.Francesco Viglietti
Www.in4matika.altervista.org

Struttura di un programma Java

- un’applicazione può essere costituita da una o più classi. Tra queste una si differenzia dalle altre perchè contiene il metodo main(). EP del programma.

```
class NomeClasse {
 public static void main(String args[]) {
 // dichiarazioni di variabili
 . . .
 // istruzioni
 . . .
 }
}
```

- public indica che il metodo è pubblico ed è visibile
- void indica che non ci sono valori di ritorno
- static indica che il metodo è associato alla classe e non può essere richiamato dai singoli oggetti della classe.

Identificatori, parole chiave ...

- Gli identificatori sono i nomi che il programmatore assegna per identificare le variabili, i metodi e le classi.
- Le parole chiave sono un insieme di parole riservate di Java che non possono essere usate come identificatori.

Le Variabili: generalmente sono composte dalla triade tipo, nome,valore
<tipo> <nome variabile> = <valore iniziale>;

int prezzo = 0;

char lettera = 'a';

double altezza = 1.83;

Una costante invece può assumere un solo valore durante tutta l'esecuzione del programma e in java si indicano :

***public static final** double **YARD_METRO** = 0.914;*

Variabili esempio ...

```
public class CestiniUova {
   public static void main(String[] args) {
      int numeroDiCestini, uovaPerCestino, totaleUova;
      numeroDiCestini = 10;
      uovaPerCestino = 6;
      totaleUova = numeroDiCestini * uovaPerCestino;
      System.out.println("Se hai");
      System.out.println(uovaPerCestino + " uova per cestino e");
      System.out.println(numeroDiCestini + " cestini");
      System.out.println("il numero totale di uova e' " + totaleUova);
  }
}
```

Operatori

- L’assegnamento di un valore a una variabile viene eseguito usando l’**operatore di assegnamento =**
- Le **operazioni aritmetiche**: **+**, **-**, *, /.
- L’operatore **%** è usato per calcolare il **resto.**
- Operatori di **incremento** e di **decremento: ++** e **– –**.

```
i++;
```

- Le stringhe possono essere concatenate tra loro usando l’**operatore di concatenazione +**.

Operatori booleani

- L’operatore **&&** (oppure &) indica l’operazione di AND.
- L’operatore **||** (oppure |) indica l’operazione di OR.
- La negazione NOT viene espressa con l’operatore **!**.

Autoconversione e casting

In Java la conversione tra valori numerici è automatica se il tipo di destinazione figura a destra della seguente successione:

byte → *short* → *int* → *long* → *float* → *double*

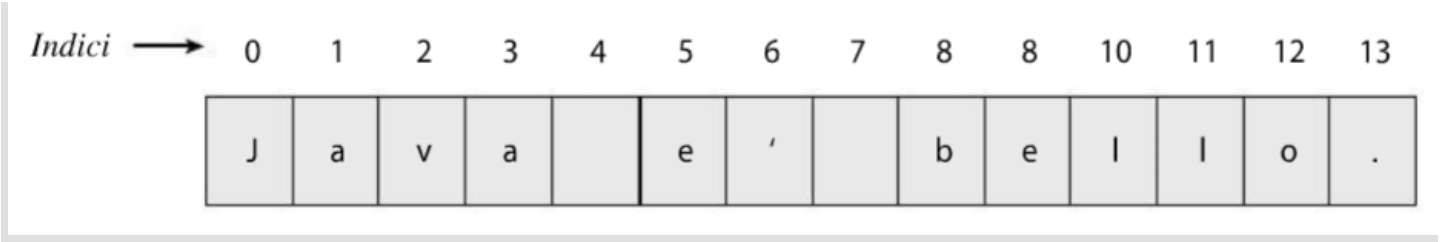
es. int numInt=7; double numDouble=numInt;

Il **casting** è il meccanismo che consente al programmatore di indicare la conversione da un tipo di dato a un altro tipo. Tale conversione viene eseguita : (tipo)espressione

```
Esempio:
floatNum = 13.278f;
intNum = (int) floatNum;
```

Le stringhe

Le stringhe di caratteri in java, non sono un tipo primitivo, ma sono trattate mediante la classe String in modo semplice.
String nome="Francesco";
Concatenazione tra stringhe operatore **+**
System.out.print("il mio nome è "+ nome);
Stampa → il mio nome è Francesco
Il metodo **length()** fornisce la lunghezza della stringa es. nome.length() → 9
Tutte le stringhe partono da 0



Le stringhe

Metodi della classe String

nome_stringa.`charAt`(*indice*)
Restituisce il carattere che si trova alla posizione *indice* della stringa corrente *nome_stringa* (this). Gli indici sono numerati a partire da 0.

nome_stringa.`compareTo`(*altra_stringa*)
Confronta la stringa corrente *nome_stringa* (this) con *altra_stringa* per individuare quale viene prima in ordine lessicografico. L'ordine lessicografico corrisponde all'ordine alfabetico quando entrambe le stringhe sono costituite solo da lettere maiuscole o solo da lettere minuscole. Il metodo restituisce un valore intero negativo se la stringa corrente viene prima, 0 (zero) se sono uguali o un numero positivo se la stringa corrente viene dopo.

nome_stringa.`concat`(*altra_stringa*)
Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this) concatenati con quelli in *altra_stringa*. Invece di `concat` può essere utilizzato l'operatore + .

nome_stringa.`equals`(*altra_stringa*)
Restituisce `true` se la stringa corrente *nome_stringa* (this) e *altra_stringa* sono uguali. Altrimenti restituisce `false`.

nome_stringa.`equalsIgnoreCase`(*altra_stringa*)
Si comporta coma il metodo `equals`, ma considera uguali le lettere maiuscole e le lettere minuscole della stringa.

nome_stringa.`indexOf`(*altra_stringa*)
Restituisce l'indice della prima occorrenza della sottostringa *altra_stringa* nella stringa corrente *nome_stringa* (this). Restituisce -1 se la sottostringa *altra_stringa* non compare. Gli indici sono numerati a partire da 0.

nome_stringa.`lastIndexOf`(*altra_stringa*)
Restituisce l'indice dell'ultima occorrenza della sottostringa *altra_stringa* all'interno della stringa corrente *nome_stringa* (this). Restituisce -1 se la sottostringa *altra_stringa* non compare. Gli indici sono numerati a partire da 0.

nome_stringa.`length()`
Restituisce la lunghezza della stringa corrente *nome_stringa* (this).

nome_stringa.`toLowerCase()`
Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui tutte le lettere maiuscole sono state sostituite con le minuscole corrispondenti.

nome_stringa.`toUpperCase()`
Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui tutte le lettere minuscole sono state sostituite con le corrispondenti lettere maiuscole.

nome_stringa.`replace`(*vecchio_carattere*, *nuovo_carattere*)
Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui tutte le occorrenze del carattere *vecchio_carattere* sono state sostituite dal carattere *nuovo_carattere*.

nome_stringa.`substring`(*inizio*)
Restituisce una nuova stringa che presenta gli stessi caratteri della sottostringa che inizia all'indice *inizio* della stringa corrente *nome_stringa* (this) fino alla fine della stringa. Gli indici sono numerati a partire da 0.

nome_stringa.`substring`(*inizio*, *fine*)
Restituisce una nuova stringa che presenta gli stessi caratteri della sottostringa che inizia all'indice *inizio* della stringa corrente *nome_stringa* (this) fino all'indice *fine* escluso. Gli indici sono numerati a partire da 0.

nome_stringa.`trim()`
Restituisce una nuova stringa che presenta gli stessi caratteri della stringa corrente *nome_stringa* (this), ma in cui sono stati rimossi i caratteri di spaziatura in testa e in coda alla stringa.

Standard output

```
System.out.println("messaggio da visualizzare 1");
System.out.print("messaggio da visualizzare 2");
```

- **System.out** rappresenta un oggetto associato allo standard output. Con l'istruzione **println / print** Viene stampato a video il messaggio desiderato.
- Esiste anche un oggetto associato allo standard *error* che è **System.err** generalmente usato per stampare messaggi d'errore.

Formattazione numeri

```
import java.text.DecimalFormat;
import java.text.NumberFormat;
public class DecimalFormatExample {
        public static void main(String[] args) {
                double money = 100550000.75;
                NumberFormat formatter = new DecimalFormat("#0.00");
                // Print the number using scientific number format.
                System.out.println(money);
                // Print the number using our defined decimal format pattern as above.
                System.out.println(formatter.format(money));
        }
}
```

Operatori di confronto

- **Operatori di confronto, vengono utilizzati nelle operazioni di controllo.**
- l'operatore di uguaglianza è rappresentato da due simboli di uguale (==).

```
if (voto == 6){
  System.out.println("sufficiente");
}
```

- La disuguaglianza (diverso da) è espressa usando l'operatore **!=**.
- Gli altri operatori di confronto sono **<**, **<=**, **>**, e **>=**.

Operatori di confronto

- **Operatori di confronto, vengono utilizzati nelle operazioni di controllo.**
- l'operatore di uguaglianza è rappresentato da due simboli di uguale (**==**).

```
if (voto == 6){
  System.out.println("sufficiente");
}
```

- La disuguaglianza (diverso da) è espressa usando l'operatore **!=**.
- Gli altri operatori di confronto sono **<**, **<=**, **>**, e **>=**.

Standard input 1

- **System.in** gestisce il flusso di dati inseriti da tastiera.
- classe **BufferedReader**

```
InputStreamReader input = new InputStreamReader(System.in);
BufferedReader tastiera = new BufferedReader(input);
```

- **lettura di una stringa:**

```
String nome;
nome = tastiera.readLine();
```

Attenzione: per poter usare lo standard input si deve importare la classe seguente:
Import java.io.*;

Standard input 2

Per usare un oggetto in di tipo Scanner bisogna importare la classe Scanner dal package java.util **import java.util.*;** dichiarare una variabile in di tipo Scanner, creare l'oggetto che rappresenta la tastiera mediante l'istruzione **in = new Scanner(System.in);** usare opportunamente le operazioni dell'oggetto in:

- int nextInt()
- double nextDouble()
- String nextLine()
- String next()

```
import java.util.*;
....
Scanner in =new Scanner(System.in);
int numero=0;
numero=in.nextInt();
```

esercizio1

Struttura di sequenza

La struttura di **sequenza** viene realizzata posizionando le istruzioni una di seguito all'altra e separandole con il punto e virgola.
Ogni dichiarazione e istruzione deve terminare con il punto e virgola.

Ogni blocco viene racchiuso tra due parentesi graffe { }
I commenti su singola riga sono preceduti da //
I commenti su più righe iniziano con /** e terminano con */

Struttura di selezione

- La struttura di **selezione:**

```
if (condizione){
  // istruzioni eseguite se la condizione è vera
}
else {
  // istruzioni eseguite se la condizione è falsa
}
```

*In tutte le strutture, i blocchi {…} sono obbligatori se contengono più di un'istruzione!

esercizio2

Selezione multipla

- La struttura di **selezione multipla:**

```
switch (espressione)
{
  case valore1: // istruzioni
                break;
  case valore2: // istruzioni
                break;
  . . . .
  default: // istruzioni
                break;
}
```

l'istruzione **break** serve per interrompere la sequenza delle istruzioni, forzando l'uscita dal blocco. Senza il **break,** la sequenza continua fino alla fine del blocco o a trovare l'istruzione **break.**

Struttura di ripetizione

- **While**

```
while (condizione){
  // istruzioni
}
```

- **Do while**

```
do {
  // istruzioni
} while(condizione)
```

- **for**

```
for(int i=..;i<..;i++) {
  // istruzioni
}
```

N.B.
Per ciclare la condizione dev'essere sempre vera!
Altrimenti si interrompe l'iterazione!

esercizio3

esercizio3a

Eccezioni

- Un'**eccezione** è una situazione anomala che si verifica durante l'esecuzione del programma.

```
try
{
  // istruzioni da controllare
}
catch(eccezione)
{
  // operazioni da eseguire se si verifica l'eccezione
}
```

Eccezioni predefinite

- **ArithmeticException**: segnala errori aritmetici
- **NullPointerException**: errore dovuto all'utilizzo di un riferimento che possiede il valore *null*
- **IndexOutOfBoundException**: errore nell'indice di un array
- **IOException**: generico errore di input/output.