Espressioni aritmetiche

```
public class Triangolo {
    public static void main ( String [] args ) {
        System.out.println (5*10/2);
    }
}
```

Il programma risolve l'espressione 5*10/2 e stampa il risultato a video

---

Espressioni aritmetiche e precedenza

singoli "letterali"

- Letterali interi: 3425, 12, -34, 0, -4, 34, -1234, ...
- Letterali frazionari: 3.4, 5.2, -0.1, 0.0, -12.45, 1235.3423, ...

operatori aritmetici

- moltiplicazione *
- divisione /
- modulo % (resto della divisione tra interi)
- addizione +
- sottrazione -

Le operazioni sono elencate in **ordine decrescente di priorità** ossia 3+2*5 fa 13, non 25

Le parentesi tonde cambiano l'ordine di valutazione degli operatori ossia (3+2)*5 fa 25

Inoltre, tutti gli operatori sono associativi a sinistra ossia 3+2+5 corrisponde a (3+2)+5 quindi 18/6/3 fa 1, non 9

---

operazione di divisione

- L'operazione di divisione / si comporta diversamente a seconda che sia applicato a letterali interi o frazionari
- 25/2 = 12 (divisione intera)
- 25%2 = 1 (resto della divisione intera)
- 25.0/2.0 = 12.5 (divisione reale)
- 25.0%2.0 = 1.0 (resto della divisione intera)
- Una operazione tra un letterale intero e un frazionario viene eseguita come tra due frazionari
- 25/2.0 = 12.5
- 1.5 + (25/2) = 13.5 (attenzione all'ordine di esecuzione delle operazioni)
- 2 + (25.0/2.0) = 14.5

Casting e promotion

- `( nometipo ) variabile`
- `( nometipo ) espressione`
- Trasforma il valore della variabile (espressione) in quello corrispondente in un tipo diverso
- Il cast si applica anche a `char`, visto come tipo intero positivo
- La promotion è automatica quando necessaria
    - Es. `double d = 3 + 4;`
- Il casting deve essere esplicito: il programmatore si assume la responsabilità di eventuali perdite di informazione
    - Per esempio
    - `int i = ( int ) 3.0 * ( int ) 4.5;` i assume il valore 12
    - `int j = ( int ) (3.0 * 4.5);` j assume il valore 13

---

casting dei tipi reference (oggetti)

- è permesso solo in caso di ereditarietà
- la conversione da sotto-classe a super-classe è automatica
- la conversione da super-classe a sotto-classe richiede cast esplicito
- la conversione tra riferimenti non in relazione tra loro non è permessa

esempio promotion

```
char a = 'a';
// promotion int è più grande e i valori sono compatibili
int b = a;

System.out.println(a); // a
System.out.println(b); // 97
```

---

esempi type casting

```
byte b = (byte) 261;
System.out.println(b); // 5
```

```
System.out.println(  Integer.toBynaryString(b)  ); // 101
System.out.println(  Integer.toBynaryString(261)  ); // 100000101
```

```
int a = (int) 1936.27;

System.out.println(a); // 1936
```

con il tipo boolean non si può fare il typecasting

```
int a = (int) true; // vietato - ... cannot be converted to ...
boolean falso = (boolean) 0; // vietato - ... cannot be converted to ...
```

Stringhe e Caratteri

Caratteristiche principali

Classi disponibili

- String
  - Modella stringhe (sequenze – array di caratteri)
  - **Non modificabile** (dichiarata final)
- StringBuilder
  - Modificabile
- StringBuffer (non si usa più)
  - Modificabile
- Character
- CharacterSet

Definizione

```
String myString; myString = new String ("stringa esempio");
```

- Oppure

```
String myString = new String ("stringa esempio");
```

- Solo per il tipo String vale

```
String myString = "stringa esempio";
```

- Il carattere " (doppi apici) può essere incluso come "
- Il nome della stringa è il riferimento alla stringa stessa
- Confrontare due stringhe NON significa confrontare i riferimenti

NB: I metodi che gestiscono il tipo String NON modificano la stringa, ma ne creano una nuova

---

Concatenare stringhe

- Operatore concat
  - `myString1.concat(myString2)`
  - `String s2 = "Ciao".concat(" a tutti").concat("!");`
  - `String s2 = "Ciao".concat(" a tutti".concat("!"));`
- Utile per definire stringhe che occupano più di una riga
- Operatore + `"questa stringa" + "e` formata da tre" + "stringhe"`
- La concatenazione funziona anche con altri tipi, che vengono automaticamente convertiti in stringhe

```
System.out.println ("pi Greco = " + 3.14);
```

```
System.out.println ("x = " + x);
```

Lunghezza stringa

- int length()
  - myString.length()
  - "Ciao".length() restituisce 4
  - "".length() restituisce 0
- Se la lunghezza è N, i caratteri sono indicizzati da 0 a N-1

Carattere i-esimo

- char charAt (int)
- myString.charAt(i)
-

Confronta stringa con s

- `boolean equals(String s)` * `myString.equals("stringa")` ritorna true o false
- `boolean equalsIgnoreCase(String s)`
- `myString.equalsIgnoreCase("StRiNgA")`

Confronta con s facendone la differenza

- `int compareTo(String str)`
- `myString.compareTo("stringa")` ritorna un valore >=< 0

Trasforma int in String

- `String valueOf(int)`
- Disponibile per tutti tipi primitivi

Restituisce indice prima occorrenza di c

- `int indexOf(char c)`
- `int indexOf(char c, int fromCtrN)`

Altri metodi

- `String toUpperCase(String str)`
- `String toLowerCase(String str)`
- `String substring(int startIndex, int endIndex)`
- `String substring(int startIndex)`

Esempio

```
String s1, s2;
s1 = new String("Prima stringa");
s2 = new String("Prima stringa");
System.out.println(s1);
/// Prima stringa
System.out.println("Lunghezza di s1 = " +
s1.length());
// 26
if (s1.equals(s2)) ...
// true
if (s1 == s2) ...
// false
String s3 = s3.substring (2, 6);
// s3 == "ima s"
```

altri esempi sulle stringhe

Array

- Sequenze ordinate di
  - Tipi primitivi (int, float, etc.)
  - Riferimenti ad oggetti (vedere classi!)
- Elementi dello stesso tipo
  - Indirizzati da indici
  - Raggiungibili con l'operatore di indicizzazione: le **parentesi quadre** []
  - Raggruppati sotto lo stesso nome

In Java gli array sono Oggetti

- Sono allocati nell'area di memoria riservata agli oggetti creati dinamicamente (heap)

Dimensione

- Può essere stabilita a run-time (quando l'oggetto viene creato)
- È fissa (non può essere modificata)
- E' nota e ricavabile per ogni array

Array Mono-dimensionali (vettori)

Dichiarazione di un riferimento a un array

- `int[] voti;`
- `int voti[];`

La dichiarazione di un array non assegna alcuno spazio

`voti == null`

Creazione di un Array

L'operatore new crea un array:

- Con costante numerica

```
int[] voti;
...
voti = new int[10];
```

- Con costante simbolica

```
final int ARRAY_SIZE = 10;
int[] voti;
...
voti = new int[ARRAY_SIZE];
```

- Con valore definito a run-time

```
int[] voti;
... definizione di x (run-time) ...
voti = new int[x];
```

**Utilizzando un inizializzatore* (che permette anche di riempire l'array)

- L'operatore new inizializza le variabili
  - 0 - per variabili di tipo numerico (inclusi i char)
  - false - per le variabili di tipo boolean

```
int[] primi = {2,3,5,7,11,13};
...
int [] pari = {0, 2, 4, 6, 8, 10,};
// La virgola finale e' facoltativa
// (elenchi lunghi)
```

- Dichiarazione e creazione possono avvenire contestualmente
- L'attributo length indica la lunghezza dell'array, cioè il numero di elementi
- Gli elementi vanno da 0 a length-1

```
for (int i=0; i<voti.length; i++)
voti[i] = i;
```

In Java viene fatto il bounds checking

- Maggior sicurezza
- Maggior lentezza di accesso

Il riferimento ad array

- Non è un puntatore al primo elemento
- È un puntatore all'oggetto array
- Incrementandolo non si ottiene il secondo elemento

Array di oggetti

Per gli array di oggetti (e.g., Integer) `Integer [] voti = new Integer [5];` ogni elemento e' un riferimento

L'inizializzazione va completata con quella dei singoli elementi

```
voti[0] = new Integer (1);
voti[1] = new Integer (2);
...
voti[4] = new Integer (5);
```

Array Multi-dimensionali (Matrici)

Array contenenti riferimenti ad altri array

Sintatticamente sono estensioni degli array a una dimensione

Sono possibili righe di lunghezza diverse (matrice = array di array)

```
int[][] triangle = new int[3][]
```

Le righe non sono memorizzate in posizioni adiacenti

- Possono essere spostate facilmente

```
// Scambio di due righe
double[][] saldo = new double[5][6];
...
double[] temp = saldo[i];
saldo[i] = saldo[j];
saldo[j] = temp;
```

- L'array è una struttura dati efficiente ogni volta che il numero di elementi è noto
- Il ridimensionamento di un array in Java risulta poco efficiente
- Utilizzare altre strutture dati se il numerodi elementi contenuto non è noto

---

Il pacchetto java.util contiene metodi statici di utilità per gli array

- Copia di un valore in tutti gli (o alcuni) elementi di un array
  - `Arrays.fill (<array>, <value>);`
  - `Arrays.fill (<array>, <from>, <to>,<value>);`
- Copia di array

    - `System.arraycopy (<arraySrc>, <offsetSrc>,<arrayDst>, <offsetDst>, <#elements>);`
- Confronta due array
    - `Arrays.equals (<array1>, <array2>);`
- Ordina un array (di oggetti che implementino l'interfaccia Comparable)
    - `Arrays.sort (<array>);`
- Ricerca binaria (o dicotomica)
    - `Arrays.binarySearch (<array>);`

Esempi di Array

Array Monodimensionali

```
int[] list = new int[10];

list.length;

int[] list = {1, 2, 3, 4};
```

Array Multidimensionali

```
int[][] list = new int[10][10];
list.length;
list[0].length;
int[][] list = {{1, 2}, {3, 4}};
```

Array irregolari

```
int[][] m = {
     {1, 2, 3, 4},
     {1, 2, 3},
     {1, 2},
     {1}
};
```

---

esempi ed esercizi su array

Il controllo del flusso

Java mette a disposizione del programmatore diverse strutture sintattiche per consentire il **controllo del flusso**

Selezione, scelta condizionale

if statements

```
if (condition) {

    //statements;

}
```

```
[optional]
else if (condition2) {

    //statements;

}
```

```
[optional]
else {

//statements;

}
```

---

switch Statements

```
switch (Expression) {

    case value1:
```

```
    //statements;
    break;
    ...
    case valuen:
    //statements;
    break;
    default:
    //statements;
}
```

Cicli definiti

Se il numero di iterazioni è prevedibile dal contenuto delle variabili all'inizio del ciclo.

```
for (init; condition; adjustment) {
//statements;
}
```

Esempio: prima di entrare nel ciclo so già che verrà ripetuto 10 volte

```
int n=10;
for (int i=0; i<n; ++i) {
    ...
}
```

Cicli indefiniti

Se il numero di iterazioni non è noto all'inizio del ciclo.

```
while (condition) {
//statements;
```

```
}
```

```
do {

//statements;

} while (condition);
```

Esempio: il numero di iterazioni dipende dai valori immessi dall'utente.

```
while(true) {
    x = Integer.parseInt(JOptionPane.showInputDialog("Immetti numero
positivo"));
    if (x > 0) break;
}
```

Cicli annidati

Se un ciclo appare nel corpo di un altro ciclo.

Esempio: stampa quadrato di asterischi di lato n

```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) System.out.print("*");
    System.out.println();
}
```

Cicli con filtro

Vengono passati in rassegna un insieme di valori e per ognuno di essi viene fatto un test per verificare se il valore ha o meno una certa proprietà in base alla quale decideremo se prenderlo in considerazione o meno.

Esempio: stampa tutti i numeri pari fino a 100

```
for (int i=1; i<100; ++i) { // passa in rassegna tutti i numeri fra 1 e 100
    if (i % 2 == 0) // filtra quelli pari
```

```
        System.out.println(i);
}
```

Cicli con filtro e interruzione

Se il ciclo viene interrotto dopo aver filtrato un valore con una data proprietà.

Esempio: verifica se un array contiene o meno numeri negativi

```
boolean trovato = false;
for (int i=0; i<v.length; ++i) // passa in rassegna tutti gli indici
dell'array v
    if (v[i]<0) { // filtra le celle che contengono valori negativi
        trovato = true;
        break; // interrompe ciclo
    }
// qui trovato vale true se e solo se vi sono numeri negativi in v
```

Cicli con accumulatore

Vengono passati in rassegna un insieme di valori e ne viene tenuta una traccia cumulativa usando una opportuna variabile.

Esempio: somma i primi 100 numeri interi.

```
int somma = 0; // variabile accumulatore di tipo int
for (int i=1; i<100; ++i) { // passa in rassegna tutti i numeri fra 1 e 100
    somma = somma + i; // accumula i valori nella variabile accumulatore
}
```

Esempio: data una stringa s, ottieni la stringa rovesciata

```
String rovesciata = ""; // variabile accumulatore di tipo String
for (int i=0; i<s.length(); ++i) { // passa in rassegna tutti gli indici
dei caratteri di s
    rovesciata = s.substring(i, i+1) + rovesciata; // accumula i caratteri
in testa all'accumulatore
}
```

Cicli misti

Esempio di ciclo definito con filtro e accumulatore: calcola la somma dei soli valori positivi di un array

```
int somma = 0;
for (int i=0; i<v.length; ++i) // passa in rassegna tutti gli indici
dell'array v
    if (v[i]>0) // filtra le celle che contengono valori positivi
        somma = somma + v[i]; // accumula valore nella variabile
accumulatore
```

Tipi di dato primitivi

- In un linguaggio ad oggetti puro, vi sono solo classi e istanze di classi:
- i dati dovrebbero essere definiti sotto forma di oggetti

Java definisce alcuni tipi primitivi

- Per efficienza Java definisce dati primitivi
- La dichiarazione di una istanza alloca spazio in memoria
- Un valore è associato direttamente alla variabile
- (e.g, i == 0)
- Ne vengono definiti dimensioni e codifica
- Rappresentazione indipendente dalla piattaforma

Tabelle riassuntive: tipi di dato

Primitive Data Types

type	bits
byte	8 bit
short	16 bit
int	32 bit
long	64 bit
float	32 bit
double	64 bit
char	16 bit
boolean	true/false

I caratteri sono considerati interi

I tipi numerici, i char

- Esempi
- `123` (int)
- `256789L` (L o l = long)
- `0567` (ottale) `0xff34` (hex)
- `123.75 0.12375e+3` (float o double)
- `'a' '%' '\n'` (char)
- `'\123'` (\ introduce codice ASCII)

Tipo boolean

- true
- false

Esempi

```
int i = 15;
long longValue = 10000000000001;
byte b = (byte)254;

float f = 26.012f;
double d = 123.567;
boolean isDone = true;
boolean isGood = false;
char ch = 'a';
char ch2 = ';';
```

```
public class Applicazione {

    public static void main(String[] args) {

        int mioNumero;
        mioNumero = 100;
        System.out.println(mioNumero);

        short mioShort = 851;
        System.out.println(mioShort);

        long mioLong = 34093;
        System.out.println(mioLong);

        double mioDouble = 3.14159732;
        System.out.println(mioDouble);

        float mioFloat = 324.4f;
        System.out.println(mioFloat);

        char mioChar = 'y';
        System.out.println(mioChar);

        boolean mioBoolean = true;
        System.out.println(mioBoolean);

        byte mioByte = 127;
        System.out.println(mioByte);
    }

}
```

Data Type	Bits	Minimum	Maximum
byte	8	-128	127
short	16	-32,768	32,767
int	32	-2,147,483,648	2,147,483,647
long	64	-9.22337E+18	9.22337E+18
float	32	See the docs	
double	64	See the docs	

Esempi gist

Everything you'll ever need to work with Java primitive types!

Le Variabili

- Una variabile è un'area di memoria identificata da un nome
- Il suo scopo è di contenere un valore di un certo tipo
- Serve per memorizzare dati durante l'esecuzione di un programma
- Il nome di una variabile è un identificatore
- può essere costituito da lettere, numeri e underscore
- non deve coincidere con una parola chiave del linguaggio
- è meglio scegliere un identificatore che sia significativo per il programma

esempio

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

        int base , altezza ;
        int area ;

        base = 5;
        altezza = 10;
        area = base * altezza / 2;

        System.out.println ( area );
    }
}
```

Usando le variabili il programma risulta essere più chiaro:

- Si capisce meglio quali siano la base e l'altezza del triangolo
- Si capisce meglio che cosa calcola il programma

Dichiarazione

- In Java ogni variabile deve essere dichiarata prima del suo uso
- Nella dichiarazione di una variabile se ne specifica il nome e il tipo
- Nell'esempio, abbiamo dichiarato tre variabili con nomi base, altezza e area, tutte di tipo int (numeri interi)
    - int base , altezza ;
    - int area ;

ATTENZIONE! Ogni variabile deve essere dichiarata UNA SOLA VOLTA (la prima volta che compare nel programma)

```
base =5;
altezza =10;
```

```
area = base * altezza /2;
```

Assegnamento

- Si può memorizzare un valore in una variabile tramite l'operazione di assegnamento
- Il valore da assegnare a una variabile può essere un letterale o il risultato della valutazione di un'espressione
- Esempi:

```
base =5;
altezza =10;
area = base * altezza /2;
```

- I valori di base e altezza vengono letti e usati nell'espressione
- Il risultato dell'espressione viene scritto nella variabile area

---

Dichiarazione + Assegnamento

Prima di poter essere usata in un'espressione una variabile deve:

- essere stata dichiarata
- essere stata assegnata almeno una volta (inizializzata)
- NB: si possono combinare dichiarazione e assegnamento.

Ad esempio:

```
int base = 5;
int altezza = 10;
int area = base * altezza / 2;
```

Costanti

Nella dichiarazione delle variabili che **NON DEVONO** mai cambiare valore si può utilizzare il modificatore **final**

```
final double IVA = 0.22;
```

- Il modificatore **final** trasforma la variabile in una costante
- Il compilatore si occuperà di controllare che il valore delle costanti non venga mai modificato (ri-assegnato) dopo essere stato inizializzato.

- Aggiungere il modificatore **final** non cambia funzionamento programma, ma serve a prevenire errori di programmazione
- Si chiede al compilatore di controllare che una variabile non venga ri-assegnata per sbaglio
- Sapendo che una variabile non cambierà mai valore, il compilatore può anche eseguire delle ottimizzazioni sull'uso di tale variabile.

Input dall'utente

- Per ricevere valori in input dall'utente si può usare la classe Scanner, contenuta nel package **java.util**
- La classe Scanner deve essere richiamata usando la direttiva import prima dell'inizio del corpo della classe

Operatori aritmetici, relazionali, di assegnazione

- Di assegnazione: = += -= *= /= &= |= ^=
- Di assegnazione/incremento: ++ -- %=

Operatore	Significato
=	assignment
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	remainder assignment

- Operatori Aritmetici: + - * / %

Operatore	Significato
+	addition
-	subtraction
*	multiplication
/	division
%	remainder
++var	preincrement
--var	predecrement
var++	postincrement
var--	postdecrement

- Relazionali: == != > < >= <=

Operatore	Significato
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to

Operatore	Significato
!=	not equal

Operatori per Booleani

- Bitwise (interi): & | ^ << >> ~

Operatore	Significato
&&	short circuit AND
||	short circuit OR
!	NOT
^	exclusive OR

Attenzione:

- Gli operatori logici agiscono solo su booleani
  - Un intero NON viene considerato un booleano
  - Gli operatori relazionali forniscono valori booleani

Operatori su reference

Per i puntatori/reference, sono definiti:

- Gli operatori relazionali == e !=
  - N.B. test sul puntatore NON sull'oggetto
- Le assegnazioni
- L'operatore "punto"
- NON è prevista l'aritmetica dei puntatori

Operatori matematici

Operazioni matematiche complesse sono permesse dalla classe Math (package java.lang)

- `Math.sin (x) calcola sin(x)`
- `Math.sqrt (x) calcola x^(1/2)`
- `Math.PI ritorna pi`
- `Math.abs (x) calcola |x|`
- `Math.exp (x) calcola e^x`
- `Math.pow (x, y) calcola x^y`

Esempio

- `z = Math.sin (x) - Math.PI / Math.sqrt(y)`

Casting e promotion

- `( nometipo ) variabile`
- `( nometipo ) espressione`
- Trasforma il valore della variabile (espressione) in quello corrispondente in un tipo diverso
- Il cast si applica anche a `char`, visto come tipo intero positivo
- La promotion è automatica quando necessaria
    - Es. `double d = 3 + 4;`
- Il casting deve essere esplicito: il programmatore si assume la responsabilità di eventuali perdite di informazione
    - Per esempio
    - `int i = ( int ) 3.0 * ( int ) 4.5;` i assume il valore 12
    - `int j = ( int ) (3.0 * 4.5);` j assume il valore 13

---

casting dei tipi reference (oggetti)

- è permesso solo in caso di ereditarietà
- la conversione da sotto-classe a super-classe è automatica
- la conversione da super-classe a sotto-classe richiede cast esplicito
- la conversione tra riferimenti non in relazione tra loro non è permessa

esempio promotion

```
char a = 'a';
// promotion int è più grande e i valori sono compatibili
int b = a;

System.out.println(a); // a
System.out.println(b); // 97
```

---

esempi type casting

```
byte b = (byte) 261;
System.out.println(b); // 5
```

```
System.out.println(  Integer.toBynaryString(b)  ); // 101
System.out.println(  Integer.toBynaryString(261)  ); // 100000101
```

```
int a = (int) 1936.27;

System.out.println(a); // 1936
```

con il tipo boolean non si può fare il typecasting

```
int a = (int) true; // vietato - ... cannot be converted to ...
boolean falso = (boolean) 0; // vietato - ... cannot be converted to ...
```

Caratteri speciali

Literal	Represents
`\n`	New line
`\t`	Horizontal tab
`\b`	Backspace
`\r`	Carriage return
`\f`	Form feed
`\\`	Backslash
`\"`	Double quote
`\ddd`	Octal character
`\xdd`	Hexadecimal character
`\udddd`	Unicode character

metodo

- Termine caratteristico dei linguaggi OOP
- Un insieme di istruzioni con un nome
- Uno strumento per risolvere gradualmente i problemi scomponendoli in sottoproblemi
- Uno strumento per strutturare il codice
- Uno strumento per ri-utilizzare il lavoro già svolto
- Uno strumento per rendere il programma più chiaro e leggibile

1. Quando il programma da realizzare è articolato diventa conveniente identificare **sottoproblemi** che possono essere risolti individualmente
2. scrivere **sottoprogrammi** che risolvono i sottoproblemi richiamare i **sottoprogrammi** dal programma principale (main)
3. Questo approccio prende il nome di **programmazione procedurale** (o astrazione funzionale)
4. In Java i **sottoprogrammi** si realizzano tramite metodi ausiliari
5. Sinonimi usati in altri linguaggi di programmazione: funzioni, procedure e (sub)routines

---

Metodi ausiliari (static)

- metodi statici: dichiarati `static`
- richiamabili attraverso nome della classe
- p.es: `Math.sqrt()`

```
public class ProvaMetodi
{
    public static void main(String[] args) {
        stampaUno();
        stampaUno();
        stampaDue();
    }

    public static void stampaUno() {
        System.out.stampaln("Hello World");
    }

    public static void stampaDue() {
        stampaUno();
        stampaUno();
    }
}
```

Metodi non static

- I metodi non static rappresentano operazioni effettuabili su singoli oggetti
- La documentazione indica per ogni metodo il tipo ritornato e la lista degli argomenti formali che rappresentano i dati che il metodo deve ricevere in ingresso da chi lo invoca
- Per ogni argomento formale sono specificati:
    - un tipo (primitivo o reference)
    - un nome (identificatore che segue le regole di naming)

Invocazione di metodi non static

- L'invocazione di un metodo non static su un oggetto istanza della classe in cui il metodo è definito si effettua con la sintassi:
- Ogni volta che si invoca un metodo si deve specificare una lista di argomenti attuali
- Gli argomenti attuali e formali sono in corrispondenza posizionale
- Gli argomenti attuali possono essere delle variabili o delle espressioni
- Gli argomenti attuali devono rispettare il tipo attribuito agli argomenti formali
- La documentazione di ogni classe (istanziabile o no) contiene l'elenco dei metodi disponibili
- La classe **Math** non è istanziabile
- La classe **String** è "istanziabile ibrida"
- La classe **StringBuilder** è "istanziabile pura"

Metodi predicativi

Un metodo che restituisce un tipo primitivo `boolean` si definisce **predicativo** e può essere utilizzato direttamente in una condizione. In inglese sono spesso introdotti da `is` oppure `has`: isMale(), hasNext().

Esempi sui metodi

Classi Java

Le classi estendono il concetto di "struttura" di altri linguaggi

Definiscono

- I dati (detti campi o attributi)
- Le azioni (metodi, comportamenti) che agiscono sui dati

Possono essere definite

- Dal programmatore (ex. Automobile)
- Dall'ambiente Java (ex. String, System, etc.)

La "gestione" di una classe avviene mediante

- Definizione della classe
- Instanziazione di Oggetti della classe

Struttura di una classe

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World");
    }

}
```

Java è un linguaggio orientato agli oggetti

- In Java quasi tutto è un oggetto
- Come definire classi e oggetti in Java?
- Classe: codice che definisce un tipo concreto di oggetto, con proprietà e comportamenti in un unico file
- Oggetto: istanza, esemplare della classe, entità che dispone di alcune proprietà e comportamenti propri, come gli oggetti della realtà
- In Java quasi tutto è un oggetto, ci sono solo due eccezioni: i tipi di dato semplici (tipi primitivi) e gli array (un oggetto trattato in modo *particolare*)
- Le classi, in quanto tipi di dato strutturati, prevedono usi e regole più complessi rispetto ai tipi semplici

Le classi in Java

- Le classi, in quanto tipi di dato strutturati, prevedono usi e regole più complessi rispetto ai tipi semplici
- Il primo passo per definire una classe in Java è creare un file che deve chiamarsi esattamente come la classe e con estensione .java
- Java permette di definire solo una classe per ogni file
- Una classe in Java è formata da:
- Attributi: (o campi/proprietà) che immagazzinano alcune informazioni sull'oggetto. Definiscono lo stato dell'oggetto
- Costruttore: metodo che si utilizza per inizializzare un oggetto
- Metodi: sono utilizzati per modificare o consultare lo stato di un oggetto. Sono equivalenti alle funzioni o procedure di altri linguaggi di programmazione

---

Incapsulamento e visibilità in Java

- Quando disegniamo un software ci sono **due aspetti** che risultano fondamentali:
  - Interfaccia: definita come gli **elementi che sono visibili dall'esterno**, come il sw può essere utilizzato
  - Implementazione: definita definendo alcuni attributi e scrivendo il codice dei differenti metodi per leggere e/o scrivere gli attributi

---

Incapsulamento

- L'incapsulamento consiste nell'**occultamento degli attributi** di un oggetto in modo che possano essere **manipolati solo attraverso metodi** appositamente implementati. p.es la proprietà `saldo` di un oggetto `conto corrente`
- Bisogna fare in modo che l'interfaccia sia più indipendente possibile dall'implementazione
- In Java l'incapsulamento è strettamente relazionato con la visibilità

Visibilità

- Per indicare la visibilità di un elemento (attribuito o metodo) possiamo farlo precedere da una delle seguenti parole riservate
- `public`: accessibile da qualsiasi classe
- `private`: accessibile solo dalla classe attuale
- `protected`: solo dalla classe attuale, le discendenti e le classi del nostro package

- Se **non indichiamo la visibilità**: sono accessibili **solo dalle classi del nostro package**

Accesso agli attributi della classe

- Gli attributi di una classe sono strettamente relazionati con la sua implementazione.
- Conviene contrassegnarli come `private` e impedirne l'accesso dall'esterno
- In futuro potremo cambiare la rappresentazione interna dell'oggetto senza alterare l'interfaccia
- Quindi non permettiamo di accedere agli attributi!
- per consultarli e modificarli aggiungiamo i metodi accessori e mutatori: `getters` e `setters`

Modifica di rappresentazione interna di una classe

- Uno dei maggiori vantaggi di occultare gli attributi è che in futuro potremo cambiarli senza la necessità di cambiare l'interfaccia
- Un linguaggio di programmazione **ORIENTATO AGLI OGGETTI** fornisce meccanismi per definire nuovi tipi di dato basati sul concetto di classe
- Una classe definisce un insieme di oggetti (conti bancari, dipendenti, automobili, rettangoli, ecc...).
- Un oggetto è una struttura dotata di proprie **variabili** (che rappresentano il suo stato) propri **metodi** (che realizzano le sue funzionalità)

Classi e documentazione

- Come la maggior parte dei linguaggi di programmazione, Java è dotato di una libreria di classi "pronte all'uso" che coprono molte esigenze
- Usare classi già definite da altri è la norma per non sprecare tempo a risolvere problemi già risolti o a reinventare la ruota (DRY)
- La libreria Java standard è accompagnata da documentazione che illustra lo scopo e l'utilizzo di ciascuna classe presente,
- Dalla versione 9 di Java la libreria è stata divisa in moduli
- Documentazione Java 8
- Documentazione Java 9
- Documentazione Java 11
- Documentazione Java 13

La doppia natura delle classi

- Le classi disponibili nella libreria standard si possono distinguere in due tipologie principali:
  - Classi istanziabili
  - Classi non istanziabili
- La stessa distinzione è applicabile alle nostre classi
- La distinzione tra classi istanziabili e non istanziabili riguarda il senso logico del loro utilizzo
- Il termine "classe non istanziabile " sarà utilizzato per indicare una classe che non ha senso istanziare, date le sue caratteristiche
- Tecnicamente sarebbe possibile usare l'operatore new su classi "non istanziabili " (composte di metodi e attributi tutti static) ma non avrebbe senso pratico
- Alcune classi (p.es. quelle astratte) non permettono l'uso dell'operatore new
- La stragrande maggioranza delle classi è istanziabile ma l'esistenza di alcune classi non istanziabili è necessaria
- La classe (indispensabile) che contiene il main è normalmente non istanziabile
- Poiché i numeri non sono oggetti, i metodi numerici appartengono a classi non istanziabili

---

Classi istanziabili

- Una classe istanziabile fornisce il prototipo di una famiglia di oggetti (istanze della classe) che hanno struttura simile ma proprietà distinte a livello individuale (valori diversi degli attributi e quindi risultati diversi prodotti dai metodi)
- L'uso tipico è la costruzione di istanze (tramite new) e quindi l'invocazione di metodi su di esse
- Nel caso di una classe istanziabile attributi e metodi rappresentano proprietà possedute da tutti gli oggetti istanza della classe
- Ogni oggetto istanza di una classe ha una sua identità "contiene" individualmente gli attributi e i metodi definiti nella classe
- Ogni volta che si costruisce un'istanza con new si crea un nuovo insieme di attributi e metodi individuali
- Nel caso di una classe non istanziabile attributi e metodi sono "unici" a livello della classe (non esistono istanze diversificate)
- Una classe istanziabile rappresenta "qualcosa" che esiste in molteplici versioni individuali che hanno una struttura comune ma ciascuna con una propria identità:
- esistono molte sequenze di caratteri (la classe String è istanziabile)
- esistono molte valute (la classe Valuta è istanziabile)
- esistono molte persone (un'ipotetica classe Persona è istanziabile)

---

una classe istanziabile

- Normalmente ha costruttori
- Attributi e metodi sono tutti (o quasi) **non static**
- Quando penso all'esecuzione dei suoi metodi ho bisogno di immaginare un'istanza individuale a cui applicarli (anche senza argomenti esterni, perché usano attributi interni)
- Nel caso di classi istanziabili attributi e metodi sono definiti a livello di istanza

- Nel caso di classi non istanziabili attributi e metodi sono definiti a livello di classe

---

Classi non istanziabili

- Una classe non istanziabile contiene un insieme di metodi (ed eventualmente attributi) di natura generale non legati alle proprietà di oggetti individuali specifici
- Non ha senso la nozione di istanza della classe poiché non ci sono caratteristiche differenziabili tra oggetti distinti
- Una classe non istanziabile rappresenta "qualcosa" di concettualmente unico, che non esiste e non può esistere in versioni separate ciascuna con una propria identità:
- esiste una sola matematica (la classe Math non è istanziabile)
- esiste un solo sistema su cui un programma è eseguito (la classe System non è istanziabile)
- esiste un solo punto di inizio di un programma (le classi contenenti il main non sono istanziabili)

---

una classe non istanziabile

- Non ha costruttori
- Attributi e metodi sono tutti static
- Quando penso all'esecuzione dei suoi metodi non ho bisogno di immaginare un'istanza individuale: sono applicabili direttamente alla classe con almeno un argomento

```
Math . sqrt (2)
Math . abs ( - 3)

// In memoria ...
Math.E //2.7182
MATH.PI //3.1415
```

Classi istanziabili “ibride”

- Alcune classi istanziabili (p.e. String) della libreria standard contengono attributi o metodi static ed hanno quindi natura ibrida
- E’ come se la classe avesse due sottoparti (una static e una no) ognuna delle quali segue le proprie regole
- Salvo rari casi, è sconsigliabile realizzare classi istanziabili ibride (sono accettabili attributi costanti definiti come static)

Instanziare una Classe: gli oggetti

Gli oggetti sono caratterizzati da

- Classe di appartenenza - tipo (ne descrive attributi e metodi)
- Stato (valore attuale degli attributi)
- Identificatore univoco (reference - handle - puntatore)

Per creare un oggetto occorre

- Dichiarare una istanza
- La dichiarazione non alloca spazio ma solo una riferimento (puntatore) che per default vale null
- Allocazione e inizializzazione
- Riservano lo spazio necessario creando effettivamente l'oggetto appartenente a quella classe

---

Notazioni Puntate

Le notazioni puntate possono essere combinate

- System.out.println("Hello world!");
- System è una classe del package java.lang
- out è una variabile di classe contenente il riferimento ad un oggetto della classe PrintStream che punta allo standard output
- println è un metodo della classe PrintStream che stampa una linea di testo

---

Operazioni su reference

Definiti gli operatori relazionali == e !=

- Attenzione: il test di uguaglianza viene fatto sul puntatore (reference) e NON sull'oggetto
- Stabiliscono se i reference si riferiscono allo stesso oggetto

È definita l'assegnazione

È definito l'operatore punto (notazione puntata)

NON è prevista l'aritmetica dei puntatori

Variabili di classe

- Rappresentano proprietà comuni a tutte le istanze
- Esistono anche in assenza di istanze (oggetti)
- Dichiarazione: static
- Accesso: NomeClasse.attributo

```
class Automobile {
static int numeroRuote = 4;
}
Automobile.numeroRuote;
```

Metodi di classe

Funzioni non associate ad alcuna istanza

- Dichiarazione: static
- Accesso: nome-classe . metodo()

```
class HelloWorld {
public static void main (String args[]) {
System.out.println("Hello World!");

//p.es  cos(x): metodo static della classe Math, ritorna un double
double y = Math.cos(x);
}
}
```

Operazioni su istanze

- Le principali operazioni che si possono effettuare sulle variabili che riferiscono istanze di una classe sono:
    - assegnamento
    - confronto
    - invocazione di metodi
- Il valore di una variabile di tipo strutturato è il riferimento ad un oggetto (istanza di una classe)
- Una stessa variabile può riferire oggetti diversi in momenti diversi a seguito di operazioni di assegnazione sul suo valore
- Se la variabile contiene il valore null non riferisce nessun oggetto in quel momento

---

Oggetti e riferimenti

- Le variabili hanno un nome, gli oggetti no
- Per utilizzare un oggetto bisogna passare attraverso una variabile che ne contiene il riferimento
- Uno stesso oggetto può essere riferito da più variabili e quindi essere raggiunto tramite nomi diversi (di variabili)

- Il rapporto variabili - oggetti riferiti è dinamico, il riferimento iniziale non necessariamente rimane legato all'oggetto per tutta la sua esistenza
- Se un oggetto non è (più) riferito da nessuna variabile diventa irraggiungibile (e quindi interviene il garbage collector)

Confronti tra variabili di tipo strutturato

- E' possibile applicare gli operatori di confronto == e != a variabili di tipo strutturato
- Se uno dei due termini del confronto è il valore null si verifica se una certa variabile riferisce un oggetto oppure no, p.e. saluto3 != null
- Se entrambi i termini del confronto sono variabili, si verifica se hanno lo stesso valore (cioè riferiscono esattamente lo stesso oggetto)

Confronto tra riferimenti vs. confronto tra oggetti

- Usare == fa il confronto tra i riferimenti non fra i valori contenuti negli oggetti (p.e. le sequenze di caratteri contenute nelle istanze di String)
- Di solito si vogliono confrontare i contenuti non i riferimenti: per questo si usa il metodo **equals**
- Il metodo booleano equals della classe String accetta come argomento il riferimento ad un altro oggetto e ritorna true se le stringhe contenute sono uguali (in modo case sensitive), false altrimenti
- Il metodo booleano equalsIgnoreCase fa lo stesso senza distinguere maiuscole/minuscole

Il Metodo Costruttore

Specifica le operazioni di inizializzazione (attributi, etc.) che vogliamo vengano eseguite su ogni oggetto della classe appena viene creato

Tale metodo ha

- Lo **stesso nome** della classe
- Tipo **non** specificato

Non possono esistere attributi non inizializzati

- Gli attributi vengono inizializzati comunque con valori di **default**

Se non viene dichiarato un costruttore, ne viene creato uno di default vuoto e senza parametri

Spesso si usa l'**overloading** definendo diversi costruttori

La distruzione di oggetti (garbage-collection) non è a carico del programmatore

---

Il costrutto new

- Crea una nuova istanza della classe specificata, allocandone la memoria
- Restituisce il riferimento all'oggetto creato
- Chiama il costruttore del nuovo oggetto

```
Automobile a = new Automobile ();
Motorcycle m = new Motorcycle ();
String s = new String ("ABC");
```

Per "gestire" una classe occorre

- Accedere ai metodi della classe
- Accedere agli attributi della classe

Messaggi

- L'invio di un messaggio provoca l'esecuzione del metodo

Inviare un messaggio ad un oggetto

- Usare la notazione "puntata" oggetto.messaggio(parametri)
- Sintassi analoga alla chiamata di funzioni in altri linguaggi
- I metodi definiscono l'implementazione delle operazioni
- I messaggi che un oggetto può accettare coincidono con i nomi dei metodi
- p.es mettiInMoto(), vernicia(), etc.

- Spesso i messaggi includono uno o più parametri
- .vernicia("Rosso")

Esempi

```
Automobile a = new Automobile();
a.mettiInMoto();
a.vernicia("Blu");
```

All'interno della classe

- I metodi che devono inviare messaggi allo stesso oggetto cui appartengono
- non devono obbligatoriamente utilizzare la notazione puntata: è sottinteso il riferimento

```
public class Libro {
    int nPagine;
    public void leggiPagina (int nPagina) {...}
    public void leggiTutto () {
        for (int i=0; i<nPagine; i++)
            leggiPagina (i);
    }
}
```

Attributi

- Stessa notazione "puntata" dei messaggi `oggetto.attributo`
- Il riferimento viene usato come una qualunque variabile

```
Automobile a=new Automobile();
a.colore = "Blu";
boolean x = a.accesa;
```

I metodi che fanno riferimento ad attributi dello stesso oggetto possono tralasciare il rif-oggetto

```
public class Automobile {
    String colore;
    void vernicia(){
        colore = "Verde";// colore si riferisce all'oggetto corrente
    }
}
```

- Esempio (messaggi e attributi)

```
public class Automobile {
    String colore;
    public void vernicia () {
        colore = "bianco";
    }
    public void vernicia (String nuovoCol) {
        colore = nuovoCol;
    }
}

Automobile a1, a2;
a1 = new Automobile ();
a1.vernicia ("verde");
a2 = new Automobile ();
```

Esempio (costruttori con overloading)

```
Class Finestra {
    String titolo;
    String colore;
    // Finestra senza titolo nè colore
    Finestra () {
    ...
    }
    // Finestra con titolo senza colore
    Finestra (String t) {
    ...
    titolo = t;
    }
    // Finestra con titolo e colore
    Finestra (String t, String c) {
    ...
    titolo = t; colore = c;
    }
}
```

Operatore `this` (Puntatore Auto-referenziante)

La parola riservata `this` e' utilizzata quale puntatore auto-referenziante

- `this` riferisce l'oggetto (e.g., classe) corrente

Utilizzato per:

- Referenziare la classe appena istanziata
- Evitare il conflitto tra nomi

```
class Automobile{
String colore;
...
...
void vernicia (String colore) {
this.colore = colore;
}
}
. . .
Automobile a2, a1 = new Automobile;
a1.vernicia("bianco"); // a1 == this
a2.vernicia("rosso");
// this == a2
```

```
class Automobile{
String colore;
...
...
void vernicia (String colore) {
this.colore = colore;
}
}
. . .
Automobile a2, a1 = new Automobile;
a1.vernicia("bianco"); // a1 == this
a2.vernicia("rosso");
// this == a2
```

Package java.lang

- Il package java.lang è il package più importante dell'API di Java, in quanto contiene moltissime classi e interfacce fondamentali per la programmazione Java, tanto che questo package viene importato in automatico in tutti i programmi.

Astrazioni di classe, oggetto, sistema, ...

- Object
- System
- Package
- Class
- ClassLoader
- ClassValue

Classi wrapper e gestione tipi

- Boolean
- Byte
- Character
- Double
- Float
- Integer
- Long
- Short
- Void
- Enum

Stringhe

- String
- StringBuffer
- StringBuilder

Matematica

- Math
- StrictMath
- Number

Altre funzionalità

- Compiler
- Process
- Runtime
- SecurityManager
- StackTraceElement
- Thread

- Throwable

Classe Runtime

- Questa classe astrae il concetto di runtime (esecuzione) del programma. Non ha costruttori pubblici e una sua istanza si ottiene chiamando il metodo factory getRuntime().
- Caratteristica interessante di questa classe è permette di eseguire comandi del sistema operativo direttamente da Java, come ad esempio exec (di cui esistono più versioni).
- Bisogna tener conto che l'uso della classe Runtime potrebbe compromettere la portabilità delle applicazioni, infatti questa classe dipende fortemente dal sistema operativo.

I membri della classe System.

Costanti pubbliche statiche:

- java.io.PrintStream **err**
- java.io.InputStream **in**
- java.io.PrintStream **out**

Metodi pubblici statici:

- void **arraycopy**(Object src, int srcPos, Object dest, int destPos, int length)
- long **currentTimeMillis**()
- void **exit**(int status)
- void **gc**()
- java.util.Properties **getProperties**()
- String **getProperty**(String key)
- String **getProperty**(String key, String default)
- SecurityManager **getSecurityManager**()
- void **runFinalization**()
- void **setErr**(java.io.PrintStream err)
- void **setIn**(java.io.InputStream in)
- void **setOut**(java.io.PrintStream out)
- void **setProperties**(java.util.Properties properties)
- String **setProperty**(String key, String value)
- void **setSecurityManager**(SecurityManager s)

Classe System

La classe System ha il compito di interfacciare il programma Java con il sistema operativo sul quale sussiste la virtual machine.

Tutto ciò che esiste nella classe System è dichiarato statico.

VARIABILI

Iniziamo subito esaminando tre attributi statici, che rappresentano i flussi (stream) di informazioni scambiati con la console (standard input, standard output, standard error):

```
static PrintStream out

static PrintStream err

static InputStream in
```

- Ciascuno di questi tre attributi è un oggetto e sfrutta i metodi della classe relativa.
- l'oggetto out è di tipo PrintStream e viene usato per indicare l'output di default del sistema.

- l'oggetto err, anch'esso di tipo PrintStream, che viene usato per segnalare gli errori che avvengono durante l'esecuzione del programma.
- l'oggetto in è di tipo InputStream: serve per ricevere il flusso di informazioni dallo standard input. p.es la tastiera.
- E' possibile modificare il puntamento di queste tre variabili verso altre fonti di input o di output usando i metodi statici setOut(), setErr() e setIn().

METODI Principali

- il metodo arraycopy() permette di copiare il contenuto di un array in un altro.
- il metodo exit(int code) che consente di bloccare istantaneamente l'esecuzione del programma.

```
if (continua == false) {
    System.err.println("Si è verificato un problema!");
    System.exit(0);
}
```

Altri metodi interessanti:

```
    setProperty(String key, String value) e
    getProperty(String key) che servono rispettivamente ad impostare le
proprietà del sistema e a recuperare informazioni sulle proprietà del
sistema.
```

Gli oggetti di tipo Properties sono specializzazioni di tabelle hash di Java, semplici coppie chiave-valore.

Per esempio:

```
System.out.print("Versione Java Runtime Environment (JRE): ");
System.out.println(System.getProperty("java.version"));
*
System.out.print("Java è installato su: ");
System.out.println(System.getProperty("java.home"));
```

impostare una nuova proprietà mediante il codice:

```
System.setProperty("User.lastName", "Verdi");
```

Le proprietà automaticamente disponibili nell'ambiente Java.

Chiave	Valore
java.version	La versione di Java in uso.
java.vendor	Il produttore della versione di Java in uso.
java.vendor.url	L'URL del produttore della versione di Java in uso.
java.home	La directory di installazione di Java.
java.vm.specification.version	La versione delle specifiche della macchina virtuale in uso.
java.vm.specification.vendor	Il produttore delle specifiche della macchina virtuale in uso.
java.vm.specification.name	Il nome delle specifiche della macchina virtuale in uso.
java.vm.version	La versione della macchina virtuale in uso.
java.vm.vendor	Il produttore della macchina virtuale in uso.
java.vm.name	Il nome della macchina virtuale in uso.
java.specification.version	La versione delle specifiche di Java in uso.
java.specification.vendor	Il produttore delle specifiche di Java in uso.
java.specification.name	Il nome delle specifiche di Java in uso.
java.class.version	La versione delle classi di Java.
java.class.path	Il percorso delle classi di Java.
java.library.path	Il percorso delle librerie di Java.
java.io.tmpdir	Il percorso della directory dei file temporanei.
java.ext.dirs	I percorsi delle directory che contengono le estensioni di Java.
os.name	Il nome del sistema operativo in uso.
os.arch	L'architettura del sistema operativo in uso.
os.version	La versione del sistema operativo in uso.
file.separator	La sequenza per la separazione degli elementi dei percorsi nel sistema in uso.
path.separator	La sequenza per la separazione dei percorsi nel sistema in uso.
line.separator	La sequenza impiegata dal sistema in uso per esprimere il ritorno a capo.
user.name	Il nome dell'utente che sta usando l'applicazione.
user.home	La home directory dell'utente che sta usando l'applicazione.
user.dir	L'attuale cartella di lavoro dell'utente che sta usando l'applicazione.

I membri della classe Object.

Costruttori pubblici: Object()

Metodi protetti: Object **clone**() void **finalize**()

Metodi pubblici:

- boolean **equals**(Object obj)
- final Class **getClass**()
- int **hashCode**()
- final void **notify**()
- final void **notifyAll**()
- String **toString**()
- final void **wait**()
- final void **wait**(int millis)
- final void **wait**(int millis, int nanos)

In Java everything is object!

I membri della classe Math.

Questa classe serve per fare calcoli matematici e ha due attributi:

```
static double E ; //E di Eulero
static double PI; //Pi greca
```

metodi disponibili per le principali funzioni matematiche:

- valore assoluto,
- tangente,
- logaritmo,
- potenza,
- massimo,
- minimo,
- seno,
- coseno,
- esponenziale,
- radice quadrata
- arrotondamento classico, per eccesso e per difetto
- generazione di numeri casuali

Costanti pubbliche statiche:

- double E
- double PI

Metodi pubblici statici:

- double **abs**(double a)
- float **abs**(float a)
- int **abs**(int a)
- long **abs**(long a)
- double **acos**(double a)
- double **asin**(double a)
- double **atan**(double a)
- double **atan2**(double y, double x)
- double **ceil**(double a)
- double **cos**(double a)
- double **exp**(double a)
- double **floor**(double a)
- double **log**(double a)
- double **max**(double a, double b)
- float **max**(float a, float b)
- int **max**(int a, int b)

- long **max**(long a, long b)
- double **min**(double a, double b)
- float **min**(float a, float b)
- int **min**(int a, int b)
- long **min**(long a, long b)
- double **pow**(double a, double b)
- double **random**()
- double **rint**(double a)
- long **round**(double a)
- int **round**(float a)
- double **sin**(double a)
- double **sqrt**(double a)
- double **tan**(double a)
- double **toDegrees**(double angrad)
- double **toRadians**(double angdeg)