# Caratteristiche principali di Java

Java è un linguaggio di programmazione di alto livello e orientato agli oggetti, concepito da un team guidato da James Gosling a partire dal 1991 e ufficialmente rilasciato dalla Sun Microsystems nel 1995. L'obiettivo principale era sviluppare un linguaggio semplice e familiare. Le sue caratteristiche distintive includono:

1. Orientato agli oggetti:
   * Incorpora concetti come ereditarietà e polimorfismo.
   * Utilizza il collegamento dinamico.
   * Non supporta l'ereditarietà multipla e l'overload degli operatori.
2. Gestione automatica della memoria:
   * La memoria è allocata e deallocata automaticamente dal sistema, riducendo il rischio di errori di gestione della memoria da parte degli sviluppatori.
3. Portabilità:
   * Capacità di eseguire programmi su piattaforme diverse senza richiedere modifiche o ricompilazioni.

## Caratteristiche di Java

Gli obiettivi primari nella creazione del linguaggio sono i seguenti:

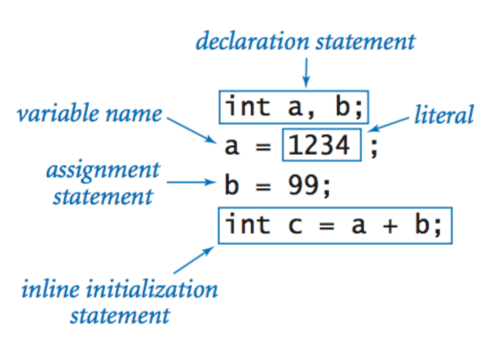
1. Orientato agli oggetti, semplice e familiare:
   * Basato su C, ma sviluppato da zero per essere estremamente semplice, senza l'uso di puntatori, macro o registri.
   * Scopo di semplificare la programmazione e ridurre gli errori.
2. Robusto e sicuro:
   * Supporta la sicurezza attraverso il concetto di sandboxing.
   * Verifica del bytecode, restrizioni nell'accesso alla rete e assegnazione dei tipi restrittiva contribuiscono alla robustezza e sicurezza.
   * La gestione della memoria è affidata al sistema.
3. Indipendente dalla piattaforma:
   * Utilizza la macchina virtuale Java (JVM) per garantire l'indipendenza dalla piattaforma.
   * Il linguaggio macchina è espresso tramite bytecode.
4. Ad alte prestazioni:
   * Risulta più efficiente di altri linguaggi interpretati grazie all'utilizzo della JVM.
5. Interpretato, multi-thread, distribuito e dinamico:
   * Il bytecode deve essere interpretato, offrendo vantaggi in termini di compattezza, efficienza e sicurezza del codice.
   * Supporta la programmazione multi-thread, consentendo l'esecuzione simultanea di diverse attività.
   * Progettato per esecuzioni remote e distribuite, con un sistema dinamico che carica classi quando sono richieste.

In sintesi, Java è un linguaggio versatile e potente che ha raggiunto un'elevata popolarità grazie alla sua facilità d'uso, robustezza e portabilità su diverse piattaforme

# 

# Le variabili e le costanti

* 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 underline
  + NON deve coincidere con una parola chiave del linguaggio
  + è meglio scegliere un identificatore che sia significativo per il programma
* Il modificatore final trasforma la variabile in una costante



### esempio

public class Triangolo {

public static void main ( String [] args ) {

int base , altezza ; //dichiarazione di variabili locali

double 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
* Evita la necessità di un commento per spiegare cosa contiene

### 

### 

### 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;

### Assegnazione

* Si può memorizzare un valore in una variabile tramite l'operazione di assegnazione
* 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 + Assegnazione

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 assegnazione.

Ad esempio:

int base = 5;

int altezza = 10;

int area = base \* altezza / 2;

## 

## 

## uso della variabile locale

//1) dichiarazione

int mioNumero;

//2) inizializzazione

mioNumero = 100;

//3) uso della variabile locale

System.out.println(mioNumero);

NB: una variabile locale deve SEMPRE essere inizializzata, prima di poter essere utilizzata

## Scope: ambito di visibilità delle variabili

class Nascoste {

static int x,y; //Def. var. globali

static void f() {

int x;

x = 1; // Locale

y = 1; // Globale

System.out.println(x);

System.out.println(y);

}

public static void main (String[] args) {

x = 0; // Globale

y = 0; // Globale

f();

System.out.println(x);

System.out.println(y);

}

}

NB: una variabile locale deve SEMPRE essere inizializzata, prima di poter essere utilizzata

## 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 (impedisce di assegnare un nuovo valore) 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.
* Il modificatore final viene utilizzato anche in altri contesti (per esempio nei metodi)

## L'attributo final

### Definisce un dato elemento come non più modificabile

* Applicato a variabile la trasforma in costante
* Applicato a un metodo
  + Ne impedisce l'overriding in classi derivate
  + Ne rende possibile l'inlining (binding statico - più efficiente)
* Applicato a una classe
  + Impedisce di derivare da essa altre classi (la classe deve essere una foglia dell'albero di ereditarietà)

### 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
* [raccolta esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/00_variabili_costanti.md)
* [altri esempi](https://github.com/maboglia/CorsoJava/tree/master/esempi/01_base/02_variabili)

# Operatori nei linguaggi di programmazione

Negli linguaggi di programmazione, gli operatori sono simboli speciali o parole chiave che eseguono operazioni su uno o più operandi. Gli operandi sono i valori o le variabili su cui l'operatore agisce. Gli operatori sono fondamentali per eseguire operazioni aritmetiche, logiche, di confronto e altre azioni specifiche all'interno di un programma. Ecco una breve definizione di alcuni tipi comuni di operatori:

1. Operatori Aritmetici:
   * Eseguono operazioni matematiche come l'addizione, la sottrazione, la moltiplicazione e la divisione.
   * Esempi: + (addizione), - (sottrazione), \* (moltiplicazione), / (divisione).
2. Operatori di Confronto o Relazionali:
   * Confrontano due valori e restituiscono un valore booleano che indica se la relazione è vera o falsa.
   * Esempi: == (uguale a), != (diverso da), < (minore di), > (maggiore di), <= (minore o uguale a), >= (maggiore o uguale a).
3. Operatori Logici:
   * Eseguono operazioni logiche su valori booleani. Solitamente utilizzati in strutture di controllo decisionale.
   * Esempi: && (AND logico), || (OR logico), ! (NOT logico).
4. Operatori di Assegnamento:
   * Assegnano un valore a una variabile.
   * Esempio: = (assegnamento), += (assegnamento con somma), -= (assegnamento con sottrazione), \*= (assegnamento con moltiplicazione), /= (assegnamento con divisione).
5. Operatori di Incremento e Decremento:
   * Modificano il valore di una variabile incrementandolo o decrementandolo di una certa quantità.
   * Esempi: ++ (incremento), -- (decremento).
6. Operatori Bitwise:
   * Eseguono operazioni bit a bit su numeri interi.
   * Esempi: & (AND bit a bit), | (OR bit a bit), ^ (XOR bit a bit), ~ (NOT bit a bit), << (shift a sinistra), >> (shift a destra).
7. Operatori Ternari:
   * Sono operatori condizionali che valutano una condizione e restituiscono un valore in base al risultato della condizione.
   * Esempio: condizione ? valore\_se\_vero : valore\_se\_falso.

Gli operatori sono essenziali per manipolare dati e controllare il flusso di esecuzione all'interno di un programma, consentendo la creazione di logica complessa e la gestione di variabili e valori.

# 

# 

# 

# 

# Operatori in Java

## Operatori aritmetici

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

| **Operatore** | **Significato** |
| --- | --- |
| + | addizione |
| - | sottrazione |
| \* | motiplicazione |
| / | divisione |
| % | resto |
| ++var | preincremento |
| --var | predecremento |
| var++ | postincremento |
| var-- | postdecremento |

## Operatori di assegnazione

| **Operatore** | **Significato** |
| --- | --- |
| = | assegnazione |
| += | addizione assegnazione |
| -= | sottrazione assegnazione |
| \*= | motiplicazione assegnazione |
| /= | divisione assegnazione |
| %= | resto assegnazione |

## 

## Operatori relazionali

== != > < >= <=

| **Operatore** | **Significato** |
| --- | --- |
| < | minore di |
| <= | minore di o uguale a |
| > | maggiore di |
| >= | maggiore di o uguale a |
| == | uguale a |
| != | non uguale / diverso |

### Operatori per Booleani

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

| **Operatore** | **Significato** |
| --- | --- |
| && | AND logico |
| ` |  |
| ! | 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 riferimenti/reference, sono definiti:

* Gli operatori relazionali == e !=
  + test sul riferimento all'oggetto, NON sull'oggetto
* Le assegnazioni
* L'operatore "punto"
* NON è prevista l'aritmetica dei puntatori, vengono gestiti dalla JVM

## Operazioni matematiche complesse

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.sqrtSì

## Comparazione del tipo di dato: Type Comparison Operator

* instanceof - Verifica se un certo oggetto è istanza di un certo Tipo di dato
* p.es: if (a instanceof Automobile) //fai qualcosa

## 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 |

* [raccolta esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/01_Operatori_Tipi.md)
* [altri esempi](https://github.com/maboglia/CorsoJava/tree/master/esempi/01_base/01_Operatori)

# Espressioni

Le espressioni in programmazione sono combinazioni di valori, operatori e chiamate di funzioni che possono essere valutate per produrre un risultato. Le espressioni possono rappresentare calcoli aritmetici, valutazioni booleane, concatenazioni di stringhe e altro ancora. Le espressioni sono fondamentali per la creazione di logica e la manipolazione dei dati all'interno di un programma. Ecco alcuni esempi di espressioni:

1. Espressioni Aritmetiche:  
   int risultato = 2 + 3 \* (5 - 1);
2. In questo esempio, l'espressione aritmetica viene valutata secondo le regole di precedenza degli operatori.
3. Espressioni Booleane:  
   boolean condizione = (x > 5) && (y <= 10);
4. Qui, l'espressione booleana verifica se entrambe le condizioni sono vere.
5. Espressioni di Concatenazione di Stringhe:  
   let nomeCompleto = nome + " " + cognome;
6. Questa espressione concatena le variabili nome e cognome in una stringa più lunga.
7. Espressioni di Assegnamento:  
   x = y + 5;
8. In questo caso, l'espressione assegna a x il valore di y più 5.
9. Chiamate di Funzioni:  
   double risultatoFunzione = Math.sqrt(25);
10. L'espressione chiama la funzione sqrt della classe Math per calcolare la radice quadrata di 25.
11. Espressioni Ternarie:  
    let risultato = (x > 0) ? "Positivo" : "Negativo";
12. Questa espressione ternaria restituisce "Positivo" se x è maggiore di 0 e "Negativo" altrimenti.
13. Espressioni di Array e Oggetti:  
    valore = array[indice];
14. Questa espressione ottiene il valore di un elemento specifico in un array.

Le espressioni possono essere più complesse con l'uso di parentesi per determinare l'ordine di valutazione. Possono anche coinvolgere variabili, costanti e altri costrutti del linguaggio di programmazione. In generale, le espressioni sono fondamentali per scrivere codice che esegue calcoli e prende decisioni in base alle condizioni e ai dati presenti nel programma

## per esempio nel programma

public class Triangolo {

public static void main ( String [] args ) {

double area = 5\*10/2;

System.out.println (area);

}

}

Il programma risolve l'espressione aritmetica 5\*10/2, memorizza in area il risultato e lo stampa 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 usare le parentesi ()

[esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/01_Operatori_Tipi.md)

# Tipi di dato primitivi

I tipi di dato primitivi in Java rappresentano i dati più elementari e fondamentali che possono essere utilizzati per dichiarare variabili e memorizzare valori. Essi non sono oggetti e non hanno metodi. Ecco una lista dei tipi di dato primitivi in Java:

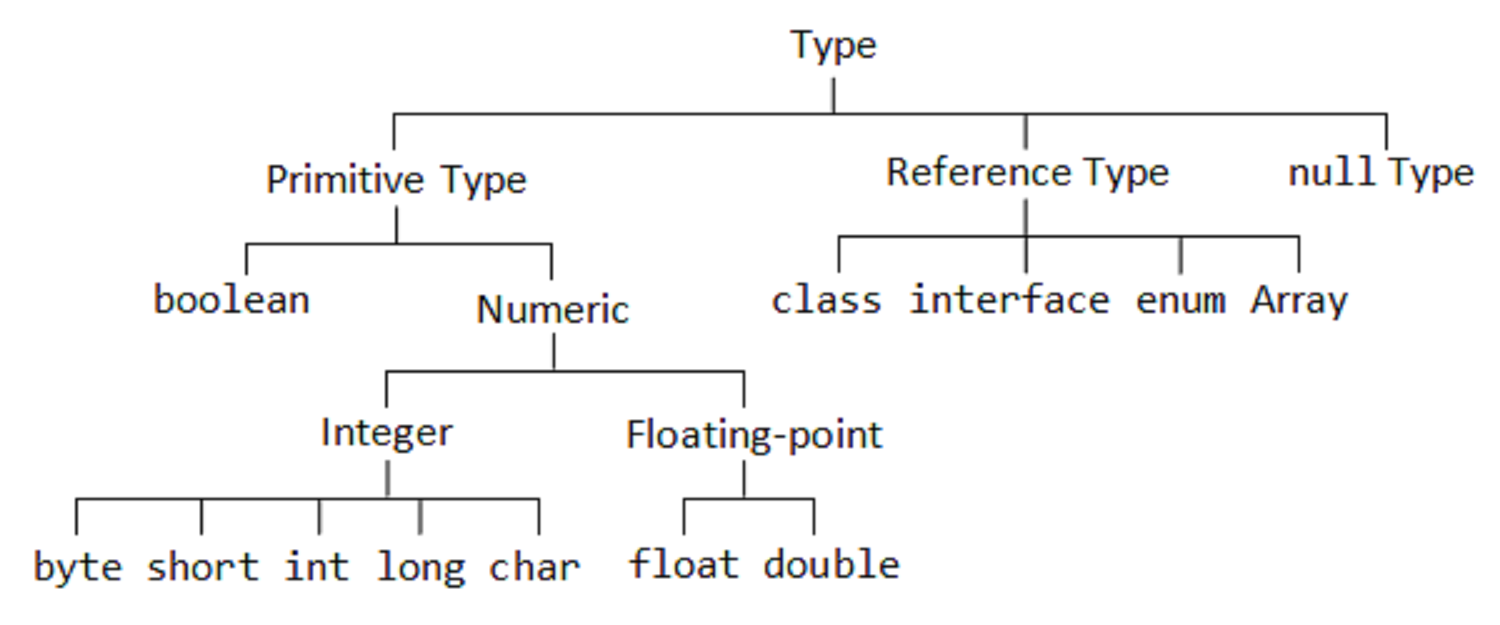
1. byte:
   * Dimensione: 8 bit
   * Intervallo: -128 a 127
2. short:
   * Dimensione: 16 bit
   * Intervallo: -32,768 a 32,767
3. int:
   * Dimensione: 32 bit
   * Intervallo: -2,147,483,648 a 2,147,483,647
4. long:
   * Dimensione: 64 bit
   * Intervallo: -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
5. float:
   * Dimensione: 32 bit
   * Precisione: Circa 7 cifre decimali
6. double:
   * Dimensione: 64 bit
   * Precisione: Circa 15 cifre decimali
7. char:
   * Dimensione: 16 bit
   * Rappresenta un singolo carattere Unicode
8. boolean:
   * Dimensione: Non specificata (spesso implementato come 1 bit)
   * Può assumere solo i valori true o false

Questi tipi di dato primitivi sono essenziali per gestire valori numerici, caratteri e informazioni booleane in maniera efficiente. Quando si utilizzano questi tipi di dato, non si fa riferimento a oggetti come avviene con i tipi di dato non primitivi. Ad esempio, un int è un tipo di dato primitivo, mentre un oggetto di tipo Integer è un tipo di dato non primitivo. I tipi di dato primitivi in Java sono fondamentali per la programmazione e vengono spesso utilizzati nella gestione di variabili e operazioni matematiche di base.

## 

## 

## Tipi di dato in Java



## I mattoncini fondamentali

I mattoncini fondamentali

## Tabelle riassuntive: tipi di dato primitivi

| **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

//tipi interi

byte b = 254;//8 bit

short s = 10;//16 bit

int i = 15;//32 bit

long l = 1000000000000l;//64 bit

//tipi reali

float f = 26.012f;//32 bit

double d = 123.567;//64 bit

//booleans

boolean fatto = true;

boolean daFare = false;

//char - rappresentabili anche come interi

char ch = 'a';

char ch2 = ';';

## uso della variabile locale

//1) dichiarazione

int mioNumero;

//2) inizializzazione

mioNumero = 100;

//3) uso della variabile locale

System.out.println(mioNumero);

NB: una variabile locale deve SEMPRE essere inizializzata, prima di poter essere utilizzata

## 

## 

## 

## Esempio tipi primitivi

//dichiarazione e inizializzazione contemporanea

byte mioByte = 127;

System.out.println(mioByte);

short mioShort = 851;

System.out.println(mioShort);

long mioLong = 34093L;

System.out.println(mioLong);

double mioDouble = 3.14159732;

System.out.println(mioDouble);

float mioFloat = 324.4f;

System.out.println(mioFloat);

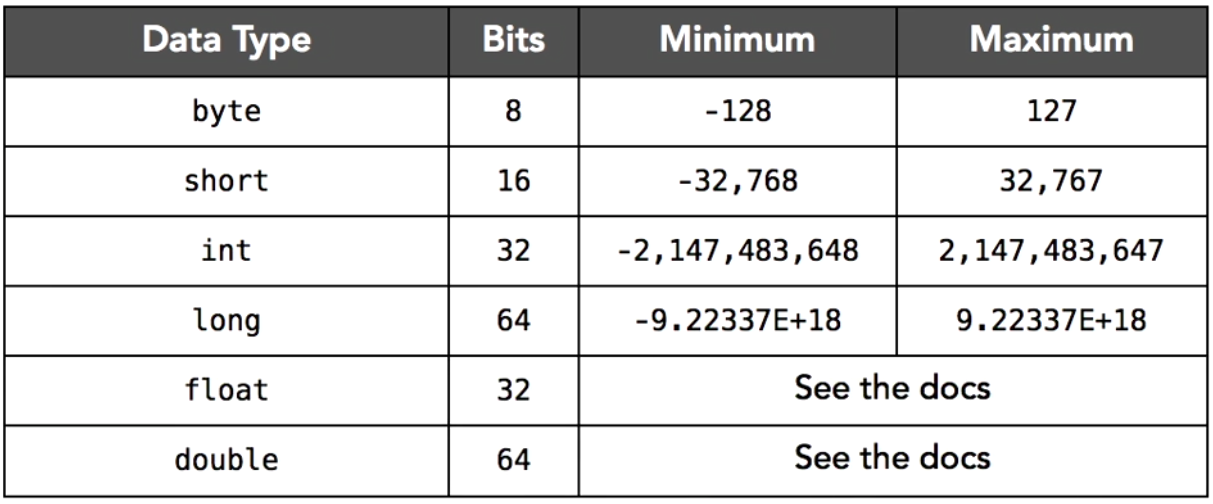
char mioChar = 'y';

System.out.println(mioChar);

boolean mioBoolean = true;

System.out.println(mioBoolean);

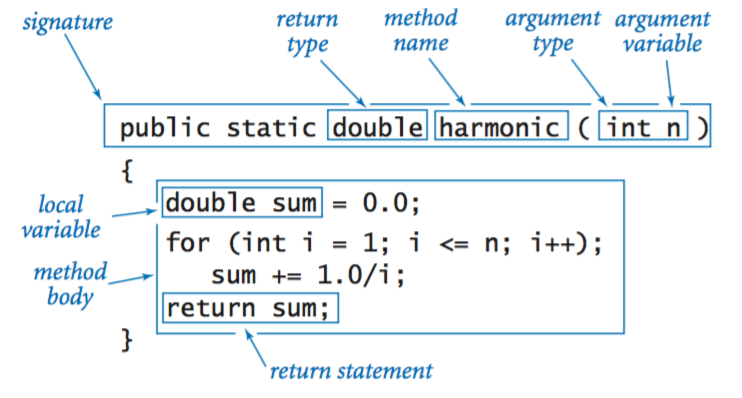
## Tipi primitivi: range di valori ammissibili



* [raccolta esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/02_tipi_primitivi.md)
* [altri esempi sui tipi primitivi](https://github.com/maboglia/CorsoJava/blob/master/esempi/01_base/01_tipi)
* [Everything you'll ever need to work with Java primitive types!](https://github.com/deletescape/Primitives)

# 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



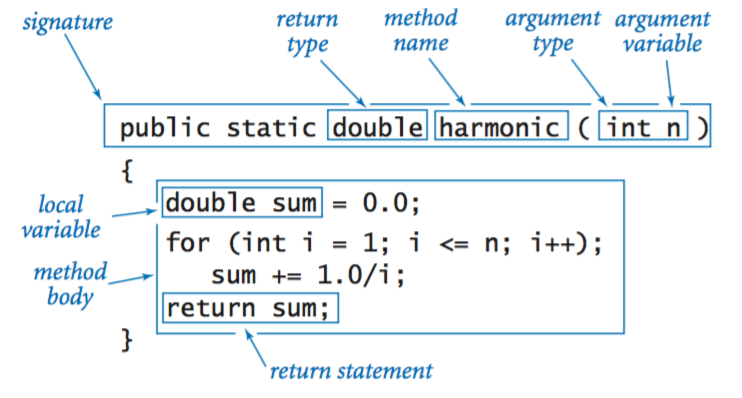
## Componenti dei metodi

Più in generale, le dichiarazioni di metodo hanno sei componenti (alcuni sono opzionali), nell'ordine:

* Modificatori, come public, private e altri che imparerai in seguito.
* Il tipo restituito: il tipo di dati del valore restituito dal metodo o void se il metodo non restituisce un valore.
* Il nome del metodo: le regole per i nomi dei campi si applicano anche ai nomi dei metodi, ma la convenzione è leggermente diversa.
* L'elenco di parametri tra parentesi: un elenco delimitato da virgole di parametri di input, preceduti dai rispettivi tipi di dati, racchiusi tra parentesi (). Se non sono presenti parametri, è necessario utilizzare parentesi vuote.
* Un elenco di eccezioni, opzionali, da discutere in seguito.
* Il corpo del metodo, racchiuso tra parentesi graffe: il codice del metodo, inclusa la dichiarazione delle variabili locali, va qui.

## Argomenti attuali e formali

* 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



## Overloading dei metodi

* E' possibile definire metodi con lo stesso nome ma liste degli argomenti diverse, cioè varianti diverse dello stesso metodo
* Si definisce *signature*, o firma del metodo l'insieme di nome ed argomenti: p.es raddoppia(String s);,raddoppia(int i);
* La diversità delle liste riguarda il numero, tipo e ordine di argomenti formali, non il loro nome
* A seconda degli argomenti passati verrà selezionato ed eseguito il metodo appropriato (se esiste)
* Non è ammesso overloading sul tipo ritornato: metodi con nome e lista degli argomenti uguali ma tipo ritornato diverso non vengono distinti e danno luogo ad errori di compilazione

## Metodi ausiliari (static)

* Il modificatore static permette di creare metodi statici: quelli dichiarati static
* I metodi static sono richiamabili attraverso nome della classe
* p.es: Math.sqrt()
* Se sono anche private e aiutano a separare la logica dal metodo main, in caso di interfacce testuali (che girano nella console), vengono definiti anche metodi ausiliari
* Nella programmazione ad oggetti bisogna farne un uso estremamente limitato!

public class ProvaMetodiStatic

{

public static void main(String[] args) {

metodoUno();

metodoUno();

metodoDue();

}

public static void metodoUno() {

System.out.metodoln("Hello World");

}

public static void metodoDue() {

metodoUno();

metodoUno();

}

}

## quando e perché usare i metodi (ausiliari)

1. Quando il programma da realizzare è articolato diventa conveniente identificare sottoproblemi che possono essere risolti singolarmente
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 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"

Argomenti correlati

* [I metodi costruttori](https://github.com/maboglia/CorsoJava/blob/master/appunti/010_3_classi_costruttori.md)
* [I metodi getters e setters](https://github.com/maboglia/CorsoJava/blob/master/appunti/010_4_classi_metodi_getter_setter.md)

## 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().