

**Tutte le variabili in Java devono essere identificate con nomi univoci. Questi nomi univoci sono chiamati identificatori, possono essere nomi brevi (come x e y) o nomi più descrittivi (anni, somma, totale).**

**Nota: si consiglia di utilizzare nomi descrittivi per scrivere codice comprensibile e gestibile.**

- 1.// Buono**
- 2.int minutiPerOra = 60;**
- 3.// Non è facile da intuire**
- 4.int m = 60;**

**Le regole principali per la denominazione dei nomi delle variabili sono:**

- **possono contenere lettere, cifre, caratteri di sottolineatura e simboli di dollaro**
  - **devono iniziare con una lettera (consigliato con una lettera minuscola)**
  - **non possono contenere spazi vuoti**
  - **possono anche iniziare con \$ e \_ (non è consigliato)**
  - **fanno distinzione tra maiuscole e minuscole ("myVar" e "myvar" sono variabili diverse)**
  - **le parole chiavi di Java (esempio: int, boolean, ecc ecc...) non possono essere utilizzate come nomi**
-

# I TIPI PRIMITIVI

---

**In java ci sono 8 tipi primitivi e ciascuno di essi è pensato per rappresentare un certo tipo di informazione con un certo range di valori e utilizzando una quantità specifica di memoria.**

**Partendo da un tipo primitivo è possibile creare nuovi tipi di variabili e inoltre le istanze dei tipi primitivi hanno un valore di default senza la necessità di una inizializzazione.**

**La memoria occupata dai tipi primitivi è nella grandezza di "bit" in binario, tramite una conversione si traduce la sequenza di bit in un valore compreso nel range dei valori possibili.**

# I TIPI PRIMITIVI

Tipo	Bit	Rappresentazione	Valori
<b>byte</b>	8	complemento a due	da –128 a 127
<b>short</b>	16	complemento a due	da –32.768 a 32.767
<b>int</b>	32	complemento a due	da –2.147.483.648 a 2.147.483.647
<b>long</b>	64	complemento a due	da –9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
<b>float</b>	32	virgola mobile standard IEEE 754	da $\pm 1.40239846 \times 10^{-45}$ a $\pm 3.40282347 \times 10^{+38}$
<b>double</b>	64	virgola mobile standard IEEE 754	da $\pm 4.94065645841246544 \times 10^{-324}$ a $\pm 1.76769313486231570 \times 10^{+308}$
<b>char</b>	16	standard UNICODE	da \u0000 a \uFFFF
<b>boolean</b>			true e false

PAUSA FINO A E 02 1/2

# I TIPI PRIMITIVI

---

**I tipi numerici primitivi sono divisi in:**

- **interi** - rappresentano i numeri interi.
- **virgola mobile** - rappresentano numeri con una parte frazionaria ovvero con una parte decimale.

**La differenza principale tra i tipi numerici dello stesso gruppo è la bontà del range, più il range è ampio e più bit in memoria richiede (come si vede in tabella).**

---

**Byte:**

**1. byte myByte = 100;**

**Short:**

**1. short myShort = 5000;**

**Int:**

**1. int myInt = 100000;**

**Long:**

**1. long myLong = 15000000000L;**

**(Per il tipo long bisogna specificare una L alla fine)**

**Float:**

**1. float myFloat= 5.75f;**

**(Per il tipo float bisogna specificare una f alla fine)**

**Double:**

**1. double myDouble = 19.99d;**

**(Per il tipo double bisogna specificare una d alla fine)**

**Un numero in virgola mobile può essere rappresentato in base scientifica ovvero con una "e" per indicare la potenza di 10**

- **float myFloat = 35e3f;**
- **double myDouble = 12E4d;**

**Un altro tipo primitivo è il booleano, può avere solo due valori:**

- **true.**
- **false.**

**Vengono utilizzati principalmente per indicare delle condizioni.**

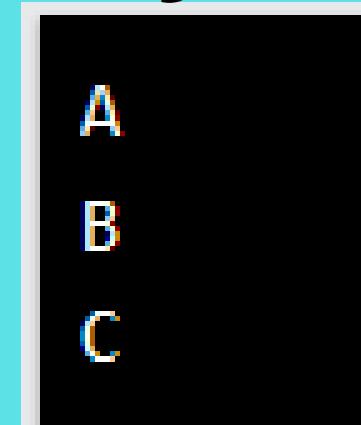
- 1. boolean isJava = true;**
- 2. boolean isFerrari = false;**

**L'ultimo tipo primitivo è il char e memorizza un singolo carattere e deve essere compreso in due apici singoli**

**1. char myChar = 'A';**

**Oppure utilizzando i valori ASCII.**

**1. char myVar1 = 65, myVar2 = 66, myVar3 = 67;  
2. System.out.println(myVar1);  
3. System.out.println(myVar2);  
4. System.out.println(myVar3);**



# I TIPI PRIMITIVI

**Il tipo String è così tanto utilizzato che viene considerato :  
"Il nono tipo"**

**Altro non è che un tipo derivato composto da una serie di char  
incorporando vari metodi che un tipo primitivo non può avere.**

**1. String testo= "Ciao Mondo";**

**I tipi non primitivi sono chiamati tipi di riferimento perché si riferiscono agli oggetti.**

**La principale differenza tra tipi di dati primitivi e non primitivi sono:**

- I tipi primitivi sono predefiniti (già definiti) in Java. I tipi non primitivi vengono creati e non è definiti da Java (ad eccezione di String).**
- I tipi non primitivi possono essere utilizzati per chiamare metodi ed eseguire determinate operazioni, mentre i tipi primitivi non possono.**
- Un tipo primitivo ha sempre un valore all'inizializzazione mentre i tipi non primitivi possono, quindi un tipo primitivo ha valore "null" alla dichiarazione.**
- Un tipo primitivo inizia con una lettera minuscola, mentre i tipi non primitivi iniziano con una lettera maiuscola.**
- La dimensione di un tipo primitivo dipende dal tipo di dati.**

**In informatica, in particolare nella programmazione, la conversione di tipo (detta impropriamente casting, o typecasting, dal nome di uno dei modi in cui essa si manifesta) è l'operazione con cui si converte una variabile da un tipo di dato a un altro: questo passaggio è effettuato per avvantaggiarsi di alcune caratteristiche delle gerarchie dei tipi. Per esempio, i valori di un intervallo limitato (come quello degli interi) possono essere immagazzinati in piccole quantità di memoria, per poi essere convertiti in un formato diverso che potenzialmente permette nuove operazioni, come la divisione con varie cifre decimali di precisione.**

**Nei linguaggi di programmazione orientati agli oggetti, la conversione di tipo permette ai programmi di trattare gli oggetti come se fossero di un tipo antenato, ad esempio per semplificare l'interazione con questi oggetti.**

---

**Esistono due tipi di conversione di tipo: implicita ed esplicita. Il termine utilizzato per riferirsi alla conversione implicita è coercion (it.: coercizione), mentre il metodo più comune di conversione esplicita è chiamato casting (sostantivo che, a volte, viene impropriamente usato per indicare il significato generale di conversione).**

**La conversione esplicita può essere eseguita anche con delle funzioni di conversione definite separatamente, come i costruttori che sovraccarichi o i metodi ovveride-ati.**

## Conversione di tipo implicita

**La conversione implicita di tipo, conosciuta anche con il nome inglese coercion, è svolta in maniera automatica dal compilatore. Alcuni linguaggi di programmazione, addirittura, richiedono che questa funzionalità sia implementata nel compilatore. In un'espressione di tipo misto, a runtime, i dati di uno o più sottotipi possono essere convertiti in tipi più generici in maniera che l'espressione sia valutata correttamente.**

**Per esempio, il codice C che segue è valido:**

```
double d;  
long l;  
int i;  
  
if (d > i)    d = i;  
if (i > l)    l = i;  
if (d == l)   d *= 2;
```

## Conversione di tipo implicita

---

**Nonostante d, l ed i appartengano a tipi di dato diversi, essi vengono automaticamente convertiti nello stesso tipo ogni volta che viene eseguito un confronto od una assegnazione. Questo comportamento del linguaggio dovrebbe essere utilizzato con cautela, in quanto potrebbe portare a conseguenze inaspettate.**

**Per esempio, se si tentasse di assegnare il valore di una variabile di tipo double ad una di tipo int, si verificherebbe una perdita di informazione: la parte decimale, infatti, verrebbe persa (ogni compilatore, se impostato al giusto livello di avvertimento, notificherebbe la situazione allo sviluppatore). Allo stesso modo, la conversione da un tipo intero ad uno decimale può far perdere precisione, a causa della granularità di quest'ultimo tipo di dato (per ulteriori informazioni, vedera la voce virgola mobile). Nel caso più comune, questo causa problemi quando devono essere confrontate (per verificarne l'uguaglianza) una variabile intera ed una in virgola mobile.**

**Un esempio di casting in un frammento di codice C:**

```
1. int arrotonda(float value){  
2.     return (int)(value + 0.5);}
```

**Il codice definisce, a titolo puramente esemplificativo, una semplice funzione che implementa l'operazione di arrotondamento di un numero decimale positivo.**  
**Poiché il valore dell'espressione `value + 0.5` è di tipo float, è necessario troncarne la parte decimale: per far questo viene utilizzata una conversione esplicita al tipo int.**  
**La conversione può avvenire anche in senso inverso, come nel codice che segue:**

```
1. int a = 5;  
2. int b = 3;  
3. float c, d;  
4.  
5. c = a / b; /* c adesso contiene il valore 1.0 */  
6. d = (float)a / b; /* d assume il valore atteso: 1.666666 */
```

**Il casting del tipo si verifica quando si assegna un valore di un tipo di dati primitivo a un altro tipo. In Java, ci sono due tipi di SEQUENZA casting:**

**Ampliamento del casting (Widening Casting): conversione di un tipo più piccolo in un tipo più grande**

- **byte -> short -> char -> int -> long -> float -> double**

**Casting restringente (manualmente): conversione di un tipo più grande in un tipo di dimensioni più piccole**

- **double -> float -> long -> int -> char -> short -> byte**

# Widening Casting

**L'allargamento del casting viene eseguito automaticamente quando si passa un tipo di dimensioni inferiori a un tipo di dimensioni maggiori:**

```
public class Main {  
    public static void main(String[] args) {  
        int myInt = 9;  
        double myDouble = myInt;          // casting automatico: int to double  
  
        System.out.println(myInt);       // Outputs 9  
        System.out.println(myDouble);    // Outputs 9.0  
    } }
```

**Il restringimento del cast deve essere eseguito manualmente posizionando il tipo tra parentesi davanti al valore:**

```
public class Main {  
    public static void main(String[] args) {  
        double myDouble = 9.78d;  
        int myInt = (int) myDouble; // casting manuale: double to int  
  
        System.out.println(myDouble); // Outputs 9.78  
        System.out.println(myInt); // Outputs 9  
    } }
```

# Esercizio

---

**Creare un sistema di inserimento  
che permetta poi di convertire e veder  
stampato nei vari tipi il dato fornito in input**

**Consiglio: Usate uno scanner diverso per ogni  
in entrata e ricordatevi che il boll non può  
essere convertito, provate anche da char a  
string e da string a char**

---

# RECAP

---

- **OOP - significato e componenti principali**
  - **Classi con attributi e metodi**
  - **Java, Sql, Html e CSS introduzione al concetto**
  - **Java hello world e esercizio uno**
  - **Variabili, commenti, costanti, print, metodo main, class base, casting, basilar type e non, scanner basilarmente**
-

# INTRODUZIONE SECONDA GIORNATA

- Check list di oggi:

# SCANNER

**È una classe utilizzata per ottenere input dell'utente, la troviamo nella libreria "java.util". Per utilizzare lo scanner dobbiamo prima importare la classe Scanner.**

```
1. import java.util.Scanner; // Import the Scanner class
2. class Main {
3.     public static void main(String[] args) {
4.         Scanner myObj = new Scanner(System.in); // Create a Scanner object
5.         System.out.println("Enter username");
6.
7.         String userName = myObj.nextLine(); // Read user input
8.         System.out.println("Username is: " + userName);} // Output user input
```

**Per ricavare il valore inserito dall'utente bisogna usare le funzioni di Scanner, come "nextLine()" che legge i valori String.**

***Nota: per accedere alle funzioni (come per i metodi e tutti gli elementi accessibili) bisogna utilizzare il ":";***

# SCANNER

---

**Scanner ha le funzioni per leggere le principali 8 primitive di java e sono:**

- **nextLine()** - legge i valori di tipo **char** e **String**;
- **nextBoolean()** - legge i valori di tipo **booleano**;
- **nextByte()** - legge i valori di tipo **byte**;
- **nextDouble()** - legge i valori di tipo **double**;
- **nextFloat()** - legge i valori di tipo **float**;
- **nextInt()** - legge i valori di tipo **int**;
- **nextLong()** - legge i valori di tipo **long**;
- **nextShort()** - legge i valori di tipo **short**;

***Nota: inserire un valore in input di diverso tipo da quello che si aspetta lo Scanner con le funzioni di lettura genererà un errore (come "InputMismatchException").***

---

**PAUSA, SI RIPRENDE ALLE 10:05**

# Operatori Java

**Gli operatori vengono utilizzati per eseguire operazioni su variabili e valori.  
Nell'esempio seguente, utilizziamo l'operatore ( + ) per sommare due valori:**

**1. int x = 100 + 50;**

**Sebbene l'operatore + sia spesso utilizzato per sommare due valori, come nell'esempio precedente, può anche essere utilizzato per sommare una variabile e un valore, o una variabile e un'altra variabile:**

**1. int sum1 = 100 + 50; // 150 (100 + 50)  
2. int sum2 = sum1 + 250; // 400 (150 + 250)  
3. int sum3 = sum2 + sum2; // 800 (400 + 400)**

# Operatori Java

**Java suddivide gli operatori nei seguenti gruppi:**

- **Operatori aritmetici**
- **Operatori di assegnamento**
- **Operatori di confronto**
- **Operatori logici**
- **Operatori bit a bit**

# Cos'è un operatore logico?

**Gli operatori logici sono operazioni tra due proposizioni A e B legate da un determinato tipo di relazione, tali da dare origine a una terza proposizione C con valore vero o falso.**  
**Gli operatori logici sono detti anche connettivi logici e sono alla base dell'algebra booleana.**

**I principali operatori logici sono la congiunzione logica AND, la congiunzione inclusiva OR e la negazione logica NOT, gli operatori logici sono i seguenti:**

- **OR ("o" inclusivo) esempio: sia occhi azzurri sia capelli biondi (anche entrambi)**
- **AND ("e" congiunzione) esempio: occhi azzurri e capelli biondi**
- **NOT ("non" negazione) esempio: non occhi azzurri**
- **XOR ("o" elusivo) esempio: o occhi azzurri o capelli biondi (non entrambi contemporaneamente )**
- **AND bitwise (esattamente come l'AND ma esegue il check sia di A che di B, anche se A è false).**

# Operatori Java

**Gli operatori aritmetici vengono utilizzati per eseguire operazioni matematiche comuni.**

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	$x / y$
%	Modulus	Returns the division remainder	$x \% y$
++	Increment	Increases the value of a variable by 1	$++x$
--	Decrement	Decreases the value of a variable by 1	$--x$

## **esercizio1:**

**Estendere l'esempio con due letture di valori dall'utente, ovvero il numero su cui eseguire il modulo e il modulo da utilizzare.**

## **esercizio2:**

**Estendere l'esercizio precedente controllando anche se il numero inserito sia pari.**

**PAUSA FINO ALLE 12**

# Operatori di assegnazione

**Gli operatori di assegnazione vengono utilizzati per assegnare valori alle variabili. Nell'esempio seguente, utilizziamo l' operatore di assegnazione ( = ) per assegnare il valore 10 a una variabile chiamata x :**

**1. int x = 10;**

**L' operatore di assegnazione di addizione ( += ) aggiunge un valore a una variabile:**

**1. int x = 10;  
2. x += 5;**

## Un elenco di tutti gli operatori di assegnazione:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
=	x  = 3	x = x   3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

# Operatori di confronto

**Gli operatori di confronto vengono utilizzati per confrontare due valori (o variabili). Questo è importante nella programmazione, perché ci aiuta a trovare risposte e prendere decisioni. Il valore restituito di un confronto è true o false. Questi valori sono noti come valori booleani e imparerai di più su di essi nel capitolo Booleani e If..Else .**

**Nell'esempio seguente, utilizziamo l' operatore maggiore di > ( ) per scoprire se 5 è maggiore di 3:**

```
int x = 5;  
int y = 3;  
System.out.println(x > y); // returns true
```

## Un elenco di tutti gli Operatori di confronto :

Operator	Name	Example
<code>==</code>	Equal to	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>&gt;</code>	Greater than	<code>x &gt; y</code>
<code>&lt;</code>	Less than	<code>x &lt; y</code>
<code>&gt;=</code>	Greater than or equal to	<code>x &gt;= y</code>
<code>&lt;=</code>	Less than or equal to	<code>x &lt;= y</code>

# Operatori logici Java

**Puoi anche testare true o false valori con operatori logici.**

**Gli operatori logici vengono utilizzati per determinare la logica tra variabili o valori:**

Operator	Name	Description	Example
<code>&amp;&amp;</code>	Logical and	Returns true if both statements are true	<code>x &lt; 5 &amp;&amp; x &lt; 10</code>
<code>  </code>	Logical or	Returns true if one of the statements is true	<code>x &lt; 5    x &lt; 4</code>
<code>!</code>	Logical not	Reverse the result, returns false if the result is true	<code>!(x &lt; 5 &amp;&amp; x &lt; 10)</code>

# Operatori Booleani

## Operatore AND

A	B	$A \wedge B$
F	F	F
F	V	F
V	F	F
V	V	V

## Operatore OR

A	B	$A \vee B$
F	F	F
F	V	V
V	F	V
V	V	V

## Operatore NOT

**Vedremo nello  
specifico questi  
elementi in futuro**

A	$\neg A$
F	V
V	F

## **Check comprensione aula**

**Piccolo test sulle caratteristiche già visionate**

# Stringhe Java

**Le stringhe vengono utilizzate per memorizzare il testo. Una variabile String contiene una raccolta di caratteri racchiusi tra virgolette:**

**1. String greeting = "Hello";**

**Una stringa in Java è in realtà un oggetto, che contiene metodi che possono eseguire determinate operazioni sulle stringhe.**

**Ad esempio, la lunghezza di una stringa è data dal il metodo length():**

**1. String txt = "ABCDEFGHIJKLMNPQRSTUVWXYZ";  
2. System.out.println("The length of the txt string is: " + txt.length());**

# Stringhe Java

**Sono disponibili molti metodi di stringa, ad esempio `toUpperCase()` and `toLowerCase()`:**

```
1.String txt = "Hello World";
2.System.out.println(txt.toUpperCase()); // Outputs "HELLO WORLD"
3.System.out.println(txt.toLowerCase()); // Outputs "hello world"
```

**Trovare un carattere in una stringa Il metodo `indexOf()` restituisce l' indice della prima occorrenza di un testo specificato in una stringa (spazi inclusi):**

```
1.String txt = "Please locate where 'locate' occurs!";
2.System.out.println(txt.indexOf("locate")); // Outputs 7
```

**Java conta le posizioni da zero, quindi, 0 è la prima posizione in una stringa, 1 è la seconda, 2 è la terza..**

**L'operatore + può essere utilizzato tra le stringhe per combinarle. Questo si chiama concatenazione :**

```
1. String firstName = "John";  
2. String lastName = "Doe";  
3. System.out.println(firstName + " " + lastName);
```

**Si noti che abbiamo aggiunto un testo vuoto (" ") per creare uno spazio tra firstName e lastName sulla stampa.**

**Puoi anche utilizzare il metodo concat() per concatenare due stringhe:**

```
1. String firstName = "John ";  
2. String lastName = "Doe";  
3. System.out.println(firstName.concat(lastName));
```

**AVVERTIMENTO!** Java utilizza l'operatore sia per l'addizione che per la concatenazione. I numeri vengono aggiunti. Le stringhe sono concatenate.

**Se aggiungi due numeri, il risultato sarà un numero:**

- 1. int x = 10; int y = 20;**
- 2. int z = x + y; // z will be 30 (an integer/number)**

**Se aggiungi due stringhe, il risultato sarà una concatenazione di stringhe:**

- 1. String x = "10";**
- 2. String y = "20";**
- 3. String z = x + y; // z will be 1020 (a String)**

**Se aggiungi un numero e una stringa, il risultato sarà una concatenazione di stringhe:**

- 1. String x = "10";**
- 2. int y = 20;**
- 3. String z = x + y; // z will be 1020 (a String)**

**Poiché le stringhe devono essere scritte tra virgolette, Java fraintenderà questa stringa e genererà un errore:**

```
String txt = "We are the so-called "Vikings" from the north.;"
```

**La soluzione per evitare questo problema è utilizzare il carattere di escape barra rovesciata . Il carattere escape barra rovesciata ( \ ) trasforma i caratteri speciali in caratteri stringa:**

Escape character	Result	Description
\'	'	Single quote
\\"	"	Double quote
\\\	\	Backslash

**La sequenza \" inserisce un doppio apice in una stringa:**

```
String txt = "We are the so-called \"Vikings\" from the north.;"
```

**La sequenza \' inserisce una singola virgoletta in una stringa:**

```
String txt = "It\'s alright.;"
```

**La sequenza \\ inserisce una singola barra rovesciata in una stringa:**

```
String txt = "The character \\ is called backslash.;"
```

## Altre sequenze di escape comuni valide in Java sono:

Code	Result
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed

**La soluzione standard per dividere una stringa è usando split() metodo fornito dal String classe. Accetta un'espressione regolare come delimitatore e restituisce un array di stringhe. Per dividere su qualsiasi carattere di spazio vuoto, puoi utilizzare la classe di caratteri predefinita \s che rappresenta uno spazio vuoto.**

```
1. import java.util.Arrays;  
2.  
3. public class Main{  
4.     public static void main(String[] args){  
5.         String str = "Hello World";  
6.         String[] words = str.split("\\s");  
7.         System.out.println(Arrays.toString(words));}
```

**Vedremo in futuro cos'è un array per ora ci basti sapere che è un aggregazione**

**La classe String espone anche numerosi metodi per l'accesso alle proprietà della stringa sulla quale stiamo lavorando; uno di questi è il metodo length(), che ritorna il numero di caratteri contenuti nell'oggetto. La sua sintassi è:**

- 1. int length()**
- 2.**
- 3.**
- 4. public void printLength() {**
- 5. String descrizione = "Articolo sulle stringhe ...";**
- 6. int length = descrizione.length();**
- 7. System.out.println("Lunghezza: "+length);**
- 8. }**

## **Substring(), estrarre una sottostringa**

**Per prelevare e manipolare solo una porzione di una stringa possiamo utilizzare il metodo substring(), presente in 2 forme (overloaded vedremo più avanti cos'è):**

**String substring(int beginIndex);**

**String substring(int beginIndex, int endIndex);**

**La prima ritorna una stringa (sotto stringa di quella di partenza) a partire dall'indice specificato fino alla fine della stringa; la seconda invece, ritorna una stringa che è anch'essa sottostringa di quella di partenza, ma che parte dall'indice beginIndex e termina in endIndex -1.**

```
1.String titolo = "I promessi Sposi";  
2.String a = titolo.substring(2); // a vale "promessi Sposi"  
3.String b = titolo.substring(12); // b vale "Sposi"  
4.String c = titolo.substring(2,9); // c vale "promessi"
```

**Nota: sia l'operazione di concatenamento sia quella di estrazione di una sottostringa (e tutti i metodi che operano sulle stringhe per la verità), sono caratterizzati dal fatto di non modificare la stringa su cui vengono applicate ma di ritornarne una nuova. Ad esempio titolo.substring(12) non modifica titolo ma ritorna una nuova variabile di tipo String che contiene la sottostringa "Sposi";**

# Altri metodi per la modifica

Metodo	Descrizione
boolean <b>contains</b> (CharacterSequence s)	ritorna true se e solo se la stringa contiene la sequenza di caratteri specificati dal parametro s
boolean <b>equals</b> (Object anObj)	confronta la stringa con l'oggetto obj specificato
boolean <b>isEmpty</b> ()	ritorna true se e solo se la lunghezza della stringa è 0
String[] <b>split</b> (String regex)	suddivide la stringa intorno ad ogni occorrenza con l'espressione regex e ritorna array con tutte le sottostringhe
String <b>trim</b> ()	ritorna una copia della stringa di partenza eliminando tutti gli spazi bianchi all'inizio e alla fine della stringa

# **Esercizio**



**La classe Java Math ha molti metodi che ti consentono di eseguire compiti matematici sui numeri.**

**Math.max( x,y )**

**Il metodo può essere utilizzato per trovare il valore più alto di x e y :Math.max(x,y)**

**Math.max(5, 10);**

**Matematica.min( x,y )**

**Il metodo può essere utilizzato per trovare il valore più basso di x e y :Math.min(x,y)**

**Math.min(5, 10);**

**Matematica.sqrt( x )**

**Il metodo restituisce la radice quadrata di x :Math.sqrt(x)**

**Math.sqrt(64);**

**Matematica.abs( x )**

**Il metodo restituisce il valore assoluto (positivo) di x :`Math.abs(x)`**

**`Math.abs(-4.7);`**

**Numeri casuali**

**Math.random() restituisce un numero casuale compreso tra 0.0 (incluso) e 1.0 (escluso):**

**`Math.random();`**

**Per ottenere un maggiore controllo sul numero casuale, ad esempio, se desideri solo un numero casuale compreso tra 0 e 100, puoi utilizzare la seguente formula:**

**`int randomNum = (int)(Math.random() * 101); // 0 to 100`**

# usare i bool in java

**Molto spesso, in programmazione, avrai bisogno di un tipo di dati che può avere solo uno di due valori, come:**

**YES / NO || ON / OFF || TRUE / FALSE**

**Per questo, Java ha un booleano tipo di dati, che può memorizzare true o false.**

**Un tipo booleano viene dichiarato con la booleana parola chiave e può assumere solo i valori true o false:**

```
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun); // Outputs true
System.out.println(isFishTasty); // Outputs false
```

# usare i bool in java

---

**pausa fino alle 15.01**

# Espressioni booleane

**Un'espressione booleana restituisce un valore booleano: true o false.**  
**Questo è utile per costruire la logica e trovare risposte concrete per agire.**  
**Ad esempio, puoi utilizzare un operatore di confronto , come l' operatore maggiore di ( >), per scoprire se un'espressione (o una variabile) è vera o falsa:**

- 1. int x = 10;**
- 2. int y = 9;**
- 3. System.out.println(x > y); // returns true, because 10 is higher than 9**

**O ancora più semplice:**

- 1. System.out.println(10 > 9); // returns true, because 10 is higher than 9**

**Negli esempi seguenti, utilizziamo l' operatore uguale a  
( == ) per valutare un'espressione:**

**1. int x = 10;  
2. System.out.println(x == 10); // returns true, the value of x is  
equal to 10**

**1. System.out.println(10 == 15); // returns false, because 10 is not  
equal to 15**

# Esempio

**Pensiamo a un esempio in cui dobbiamo scoprire se una persona ha l'età per votare. Nell'esempio seguente, utilizziamo l'operatore  $\geq$  di confronto per scoprire se l'età è maggiore o uguale al limite di età per votare, che è impostato su 18:**

- 1. int myAge = 25;**
- 2. int votingAge = 18;**
- 3. System.out.println(myAge  $\geq$  votingAge);**

**Un approccio migliore, è quello di racchiudere il codice sopra in un'istruzione if...else, in modo da poter eseguire azioni diverse a seconda del risultato:**

- 1. int myAge = 25;**
- 2. int votingAge = 18;**
- 3. if (myAge  $\geq$  votingAge) {**
- 4.   System.out.println("Old enough to vote!");**
- 5. } else {**
- 6.   System.out.println("Not old enough to vote."); }**

# strutture condizionali

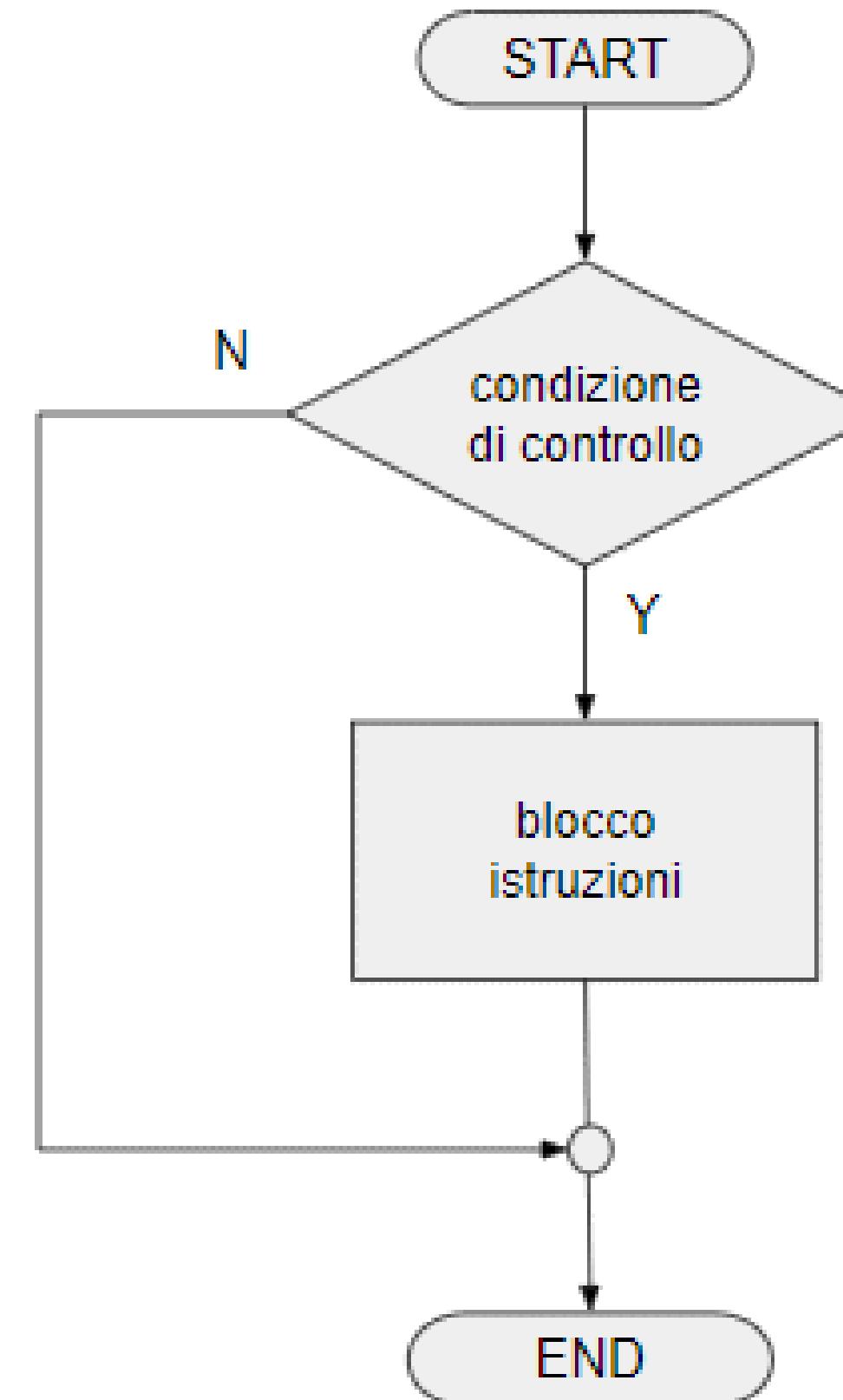
---

**In informatica, nell'ambito della programmazione, la selezione (detta struttura condizionale) è, all'interno di un algoritmo risolutivo di un problema dato, una struttura di controllo del flusso di esecuzione di un programma che indica all'elaboratore, in base alla verifica di una condizione logica specificata, quale fra due sequenze o blocchi di istruzioni eseguire, realizzando dunque un controllo logico di elaborazione.**

**Assieme alla sequenza e all'iterazione è una struttura fondamentale per la risoluzione algoritmica di un dato problema secondo il Teorema di Böhm-Jacopini.**

---

# strutture condizionali



**Sai già che Java supporta le solite condizioni logiche della matematica:**

**Minore di:  $a < b$**

**Minore o uguale a:  $a \leq b$**

**Maggiore di:  $a > b$**

**Maggiore o uguale a:  $a \geq b$**

**Uguale a  $a == b$**

**Diverso da:  $a != b$**

**È possibile utilizzare queste condizioni per eseguire azioni diverse per decisioni diverse.**

**Java ha le seguenti istruzioni condizionali:**

**Utilizzare if per specificare un blocco di codice da eseguire, se una condizione specificata è vera**

**Utilizzare else per specificare un blocco di codice da eseguire, se la stessa condizione è falsa**

**Utilizzare else if per specificare una nuova condizione da testare, se la prima condizione è falsa**

**Utilizzare switch per specificare molti blocchi alternativi di codice da eseguire**

**Utilizzare l'istruzione if per specificare un blocco di codice Java da eseguire se una condizione è true.**

**Sintassi:**

```
1. if (condition) {  
2.   // block of code to be executed if the condition is true  
3. }
```

**Si noti che if è in lettere minuscole.**

**Le lettere maiuscole (If o IF) genereranno un errore.**

**Nell'esempio seguente, testiamo due valori per scoprire se 20 è maggiore di 18. Se la condizione è true, stampa del testo:**

```
1. if (20 > 18) {  
2.     System.out.println("20 is greater than 18");  
3. }
```

**Possiamo anche testare variabili:**

```
1. int x = 20;  
2. int y = 18;  
3. if (x > y) {  
4.     System.out.println("x is greater than y");}
```

# Condizioni: IF and ELSE

**Utilizzare l'istruzione else per specificare un blocco di codice da eseguire se la condizione è false.**

## Sintassi

```
1. if (condition) {  
2.   // block of code to be executed if the condition is true  
3. } else {  
4.   // block of code to be executed if the condition is false  
5. }
```

```
1. int time = 20;  
2. if (time < 18) {  
3.   System.out.println("Good day.");  
4. } else {  
5.   System.out.println("Good evening."); } // Outputs "Good evening."
```

```
1.int time = 22;  
2.if (time < 10) {  
3.    System.out.println("Good morning.");  
4.} else if (time < 18) {  
5.    System.out.println("Good day.");  
6.} else {  
7.    System.out.println("Good evening.");  
8.}  
9.// Outputs "Good evening."
```

C'è anche una scorciatoia if else, che è nota come **operatore ternario** perché consiste di tre operandi.

Può essere utilizzato per sostituire più righe di codice con una singola riga e viene spesso utilizzato per sostituire semplici istruzioni if else:

## Sintassi

```
variable = (condition) ? expressionTrue : expressionFalse;
```

**Invece di scrivere:**

```
1. int time = 20;  
2. if (time < 18) {  
3.     System.out.println("Good day.");  
4. } else {  
5.     System.out.println("Good evening."); }
```

**Puoi semplicemente scrivere:**

```
1. int time = 20;  
2. String result = (time < 18) ? "Good day." : "Good evening."  
3. System.out.println(result);
```

# Condizioni: Switch

**Invece di scrivere molte if..else , puoi usare la dichiarazione switch .  
L'istruzione switch seleziona uno dei tanti blocchi di codice da eseguire:**

## Sintassi

```
1. switch(expression) {  
2.   case x:  
3.     // code block  
4.     break;  
5.   case y:  
6.     // code block  
7.     break;  
8. default:  
9.   // code block  
10.}
```

## Condizioni: Switch

---

**L'espressione switch viene valutata una volta. Il valore dell'espressione viene confrontato con i valori di ciascun case.**

**Se c'è una corrispondenza, viene eseguito il blocco di codice associato.**

**Le parole chiave break e default sono facoltative e verranno descritte più avanti in questo capitolo**

**L'esempio seguente utilizza il numero del giorno della settimana per calcolare il nome del giorno della settimana:**

---

```
1. int day = 4;  
2. switch (day) {  
3.     case 1:  
4.         System.out.println("Monday");  
5.         break;  
6.     case 2:  
7.         System.out.println("Tuesday");  
8.         break;  
9.     case 3:  
10.        System.out.println("Wednesday");  
11.        break;  
12.    case 4:  
13.        System.out.println("Thursday");  
14.        break;  
15.    case 5:  
16.        System.out.println("Friday");  
17.        break;  
18.    case 6:  
19.        System.out.println("Saturday");  
20.        break;  
21.    case 7:  
22.        System.out.println("Sunday");  
23.        break;}
```

**Quando Java raggiunge una parola chiave break, esce dal blocco switch.**

**Ciò interromperà l'esecuzione del codice e del test dei casi all'interno del blocco.**

**Quando viene trovata una corrispondenza il lavoro è terminato, è tempo di una pausa. Non c'è bisogno di ulteriori test.**

**Un'interruzione può far risparmiare molto tempo di esecuzione perché "ignora" l'esecuzione di tutto il resto del codice nel blocco switch.**

---

# Condizioni: Switch

**La parola chiave default specifica del codice da eseguire se non c'è corrispondenza tra maiuscole e minuscole:**

```
1. int day = 4;  
2. switch (day) {  
3.     case 6:  
4.         System.out.println("Today is Saturday");  
5.         break;  
6.     case 7:  
7.         System.out.println("Today is Sunday");  
8.         break;  
9.     default:  
10.        System.out.println("Looking forward to the Weekend");  
11.    } // Outputs "Looking forward to the Weekend"
```

**Si noti che se l'istruzione default viene utilizzata come ultima istruzione in un blocco switch, non necessita di un'interruzione.**

**Creare un sistema di inserimento che prenda tre dati in fila, un numero, una string e che setti un bool**

**Dopo di che un menu dovrà farci scegliere tra tre opzioni**

**Funzioni matematiche, Funzioni Stringa, Casting**

**Funzioni matematiche: Sul numero che abbiamo eseguiremo un operazione a scelta fra 4 disponibili e vedremo il risultato**

**Funzioni String: Devono eseguire un sub string o un concatenamento a scelta**

**Casting trasforma un dato da un tipo ad un 'altro e valorizza il risultato a schermo**

---