# Tipi primitivi di Java e valori

Java è un linguaggio tipato, abbiamo già accennato a questo in precedenza, ciò significa che ogni variabile prima di essere utilizzata deve essere dichiarata: dobbiamo quindi assegnarle un nome ed un tipo.

## Cos'è un tipo

Il **tipo** è l'insieme di caratteristiche che qualsiasi valore assunto da una variabile dovrà soddisfare. Ad esempio: "essere un intero", "essere una sequenza di carattteri Unicode", etc.

## I tipi primitivi in Java

Il tipo di una variabile può essere costruito dallo sviluppatore per composizione a partire da un set predefinito di tipi detti comunemente **tipi primitivi**.
I tipi primitivi in Java sono 8 e ciascuno di essi è pensato per rappresentare un certo tipo di informazione e utilizzando una quantià specifica di memoria.

Inoltre, parlando di dichiarazione delle variabili, abbiamo detto che le variabili "locali" devono essere inizializzate per evitare errori di compilazione, questo non è vero per le variabili di istanza per le quali, per ogni tipo primitivo, è specificato un valore di default.

| Tipo | Q.tà di Memoria | Informazione rappresentata | Valore di default |
|---|---|---|---|
| **byte** | 8 bit | Variabile con segno (con rappresentazione "two's complement", complemento a due) e rappresenta valori in un range `[-128 e 127]` (estremi inclusi) | `0` |
| **short** | 16 bit | Numeri interi (con segno) in un range `[-32,768, 32,767]` | `0` |
| **int** | 32 bit | Numeri interi (per default con segno, signed) in un range $[-2^{31}, 2^{31}-1]$ . Con Java 8 è stata introdotta la possibilità di utilizzare gli int per rappresentare quantità *unsigned* che potranno avere range $[0, 2^{32}-1]$ (grazie ad appositi metodi statici introdotti nelle classi Integer e Long) | `0` |
| **long** | 64 bit | Numeri interi (per default con segno, signed) in un range $[-2^{63}, 2^{63}-1]$. Come per gli interi in Java 8 esiste la possibilità di utilizzarli come quantità *unsigned* con range (positivo) che arriva fino a $2^{64}-1$. | `0L` |
| **float** | 32 bit | Numeri in virgola mobile in singola precisione secondo la specifica **IEEE 754**, utilizzando la rappresentazione segno, mantissa esponente. $(-1)^{segno} * mantissa * 2^{esponente}$<br><br>Nella versione a 32bit il range rappresentabile va calcolato pensando ad un bit di segno, una mantissa a 23bit e un esponente a 8bit con valori compresi tra -126 e 127.<br><br>Inoltre lo standard prevede la rappresentazione di due valori per zero (da destra e da sinistra) due per infinito (positivo e negativo), e di valori NaN (not a number) da utilizzare ad esempio come risultati di operazioni impossibili (es. divisioni per zero). | `0.0f` |
| **double** | 64 bit | Numeri in virgola mobile in doppia precisione secondo la specifica **IEEE 754**. La precisione con cui vengono rappresentati i numeri aumenta in virtù dell'aumento del numero di bit utilizzati. | `0.0d` |

| | | | |
|---|---|---|---|
| **boolean** | non specificato, ma sarebbe sufficiente un solo bit | serve a rappresentare solamente 2 valori: vero o falso (`true` o `false`). | `false` |
| **char** | 16 bit | È utilizzato per la memorizzazione di caratteri del charset **Unicode**) nel range `['\u0000', '\uffff']` (in esadecimale) o equivalentemente `[0,65535]`. | `\u0000` |

Va aggiunto che ogni variabile di tipo oggetto (cioè di tipo non primitivo) viene per default inizializzata con il valore speciale `null`.
Va notato che la stessa documentazione ufficiale di Java segnala che, benché i valori di default siano garantiti per tutti i field non inizializzati é da considerarsi una cattiva pratica quella di non inizializzare le variabili e quindi si dovrebbe cercare di evitarla.

## Literals, la codifica dei valori numerici

I tipi primitivi sono elementi con cui poter costruire tutti gli altri oggetti da utilizzare nei programmi, perciò sono da considerarsi dei tipi di dato speciali del linguaggio. Per questo c'è l'esigenza utilizzare modalità specifiche per poter definire i valori nel codice.

I **literals** sono appunto le codifiche dei valori di questi tipi nel linguaggio. Vediamo quali sono.
I possibili valori del tipo **boolean** sono esprimibili con le keyword `true` e `false`.
Valori per i tipi **int e long** sono esprimibili come interi in base 10 utilizzando la comune notazione posizionale:

```
int lifeUniverseAndEverything = 42;
```

oppure in *rappresentazione esadecimale* (Hex, in base 16) utilizzando il suffisso '`0x`' o binaria (base 2, dalla versione 7 di java) utilizzando il suffisso '`0b`':

```
int lifeUniverseAndEverythingBis = 0x2A;         // esadecimale
int lifeUniverseAndEverythingTer = 0b00101010;   // binario
```

Questi stessi literals possono essere utilizzati anche per esprimere valori di tipo **byte e short** mentre per esprimere valori **long** si pospone alla rappresentazione la lettera '`L`' (è valido anche il carattere minuscolo ma sconsigliato a causa della sua scarsa leggibilità essendo confondibile con il numero '`1`'):

```
long bigLUE = 4242424242L;
```

**Valori di tipo non intero** possono essere analogamente espressi separando la parte decimale con il simbolo '`.`' (punto) e saranno considerati di tipo **double** a meno che non sia posposta la lettera '`F`' (o '`f`').
I literals di tipo **double** possono essere terminati con la lettera '`D`' (o '`d`') qualora per motivi di leggibilità la si ritenga opportuna (ma non è obbligatoria).
È ammessa anche la cosiddetta **notazione scientifica** per i numeri in virgola mobile che consiste nell'utilizzo della lettera `E` (o `e`) seguita da un numero che esprime la

potenza di 10 da moltiplicare al numero espresso prima di essa:

```
double mille      = 1000.0;
double milleSci   = 1.0e3;
float  milleFloat = 1000.0f;
```

A partire dalla versione 7 di Java è possibile utilizzare **il carattere underscore** (`'_'`) in tutti i literal numerici per aumentarne la leggibilità, ad esempio separando le migliaia:

```
float milleEasy = 1_000.0f;
```

il carattere `'_'` non ha alcun uso se non quello di facilitarne la lettura e può essere utilizzato esclusivamente tra coppie di numeri (non come primo o ultimo carattere, non adiacente al `'.'` o agli altri caratteri ammessi nelle notazioni).

## Character and String Literals

I valori di tipo carattere possono essere espressi per mezzo di caratteri Unicode (UTF-16) racchiusi tra apici singoli che possono eventualmente essere espressi sotto forma di charcode utilizzando "Unicode escape" :

```
char nCirconflesso     = 'ñ';
char nCorconflessoCode = '\u00F1';
```

Sono supportati anche alcune speciali rappresentazioni (dette **escape sequences** o *sequenze di escape*):

| Escape | Carattere |
|---|---|
| **\b** | backspace (indietro) |
| **\t** | tab |
| **\n** | line feed (fine linea) |
| **\f** | form feed (fine pagina / nuova pagina) |
| **\r** | carriage return (ritorno carrello / a capo) |
| \' | apice singolo |
| \" | doppio apice |
| \\ | backslash (\) |

# Stringhe e numeri

In Java, accanto agli 8 tipi primitivi sono da considerarsi tipi di dato speciali (detti comunemente *Simple Data Objects*) anche i tipi **String** e **Number** (e derivati) che fungono in qualche modo da controparte dei dati primitivi dove ci sia l'esigenza di utilizzare un oggetto invece che direttamente un tipo (la differenza risulterà più chiara nel corso delle prossime lezioni).
Basti sapere al momento che le variabili di tipo `String` sono sequenze di `char` che possono essere inizializzate utilizzando le virgolette (doppi

apici):

```
String author = "Douglas Noël Adams";
```

mentre i tipi Integer, Byte, Long, Float e Double (controparti dei medesimi tipi primitivi scritti con la prima lettera minuscola) sono inizializzabili con i medesimi literals presentati per i corrispondenti tipi nativi ma tutti quanti vengono per default inizializzati al literal null (letto nullo) se non assegnamo loro esplicitamente un valore.
Il compilatore è quasi sempre in grado di convertire automaticamente i tipi primitivi nei rispettivi *simple object* (operazione detta **boxing ed unboxing**), fanno eccezione solo alcuni casi particolari.

Link Utili

- Descrizione completa dei range di **valori ammissibili dal tipo float** e delle caratteristiche delle operazioni su questo tipo di dati.
- **Simple Data Objects**, stringhe e numeri.