

## Linguaggio Java → Staticamente tipizzato

Il tipo di c'è  
ogni espressione deve  
essere noto durante la  
compilazione

↳ ogni espressione avrà un tipo

Il compilatore controlla che le operazioni siano compatibili (Type checking)

I tipi primitivi sono otto e non i soliti, oltre al void, usato come tipo di ritorno dei metodi.

C'è il supporto alle conversioni implicite, ma non per il tipo boolean.

## Funzionamento di riferimenti

Rispetto ad altri linguaggi OO, Java ha solo riferimenti ad oggetti (variabili che contengono l'indirizzo di un oggetto (Simili ai puntatori C))

↓  
meno potenti

Java prende solo il passaggio per valore, sia per i tipi che per i riferimenti. Invece, gli oggetti possono essere manipolati / passati per riferimento.

Distinzione di che tipo di ogni variabile ad espressione di categoria "riferito":

Animal a = new Dog ("Lilli");  
↳ tipo dichiarato  
↳ tipo effettivo  
↳ riferimento a dog

5 tipi riferimento sono le classi, le interfacce, le enumerazioni, gli array

↓  
Tipo composto

## Relazione di sottotipo

Per permettere il polymorfismo, esiste una relazione binaria tra tipi, chiamata relazione di sottotipo, definita dalle seguenti regole, in cui T ed U sono tipi arbitrari (esclusi i tipi di base)

- 1) T è sottotipo di se stesso;
- 2) T è sottotipo di object;
- 3) Se T estende U, o implementa U, T è sottotipo di U;
- 4) Se il tipo nullo è sottotipo di T;
- 5) Se T è sottotipo di U allora T[] è sottotipo

Questa relazione serve a definire il comportamento di instanceof:

Data un'espressione exp e il nome di un classe/interfaccia T, la seguente  
espressione ...

exp instanceof T

... restituisce vero  $\Leftrightarrow$  il tipo effettivo di exp non è nullo ed è sottotipo di T

Nota  $\Rightarrow$  Il primo argomento di instanceof deve essere di categoria riguardato, se no, errori di compilazione

# Relazione d'assegnabilità

La relazione di compatibilità / assegnabilità tra tipi stabilisce quando un valore  $T$  può essere assegnato ad una variabile di tipo  $U$ .

$T$  è assegnabile ad  $U$  se :

- $T$  è sottotipo di  $U$  ;
- $T$  ed  $U$  sono tipi di base e avranno un cast implicito da  $T$  ad  $U$

Questa relazione viene applicata in questi contesti :

Assegnazione  $\Rightarrow a = \text{exp}$

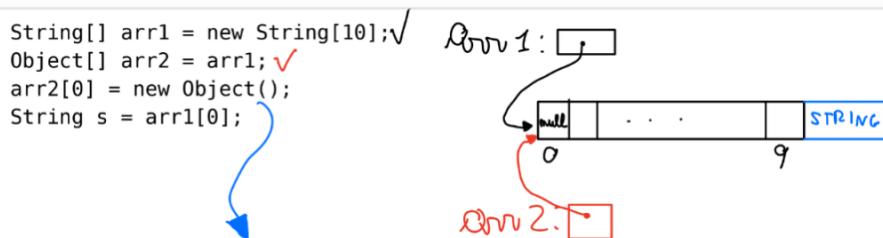
Chiamata a metodo  $\Rightarrow x.f(\text{exp})$

Ritorno da metodo  $\Rightarrow \text{return exp}$

## Array non primitivo

Scegliendo la definizione di sottotipo, un array di qualsiasi tipo non primitivo è sottotipo di array di object.

Permette di passare ogni array non primitivo ad un metodo con un array di object come parametri formale



DA ERRORE A RUN-TIME PER  
UN PROBLEMA DI COMPATIBILITÀ  
DI TIPI (`new String()`; SAREBBE CORRETTO)

## *Last* fra tipi primitivi

Java permette le conversioni esplicite tramite cast  $\Rightarrow$  (T) exp

Tra i tipi primitivi:

- Si può effettuare una promozione (*Superglio*)
- " " " " " promozione al contrario;
- da double a int;
- perdita totale di informazioni;

## *Last* fra riferimenti

È permesso il cast tra un riferimento A ad un ref. B:

① Se B è *supertipo* di A

- *Upcast*
- Superglio, i valori di tipo A non già assegnabili al tipo B

② Se B è *sottotipo* di A

- *Down cast*
- A run-time, la JVM controlla che l'oggetto da convertire appartenga effettivamente ad una sottoclasse di B (ottenuti: *ClassCastException*)
- il downcast va evitato perché aggira il type checking

③ Se nessuno è sottotipo dell'altro ...

Consideriamo       $A \ a = \dots$       A e B sono un *sidecast*  
 $B \ b = (B) \ a;$

Se nessuna delle due è un'interfaccia, sono di compilazione. Non ha senso a run-time.

Nessun oggetto è simultaneamente sottotipo di A e B.

Se una delle due è interfaccia, il cast è possibile in compilazione, ma a run-time può fallire

Sappiamo che C e B estendono A, effettua il type checking del segnale codice, tranne solo

- Errori di tipo
- Cast non validi (errori di compilazione)
- Cast validi ma pericolosi a run-time

```
boolean f(A a, B b) {  
    C c = (C) a; CAST VALIDO MA POTENZIALMENTE PERICOLOSO A RUN-TIME  
    A a1 = (A) b; ✓  
    Object o = a; ✓ PERCHÉ OBJECT SUPERCLASSE DI OGNI CLASSE JAVA  
    A[] arr = new A[10]; ✓  
    arr[5] = (Object) a; ERRORE DI COMPIAZIONE  
    arr[6] = b; ✓  
    return a == c; ✓  
} IN QUESTO CASO PER LA PRIMA ISTRUZIONE VERRÀ RESTITUITO TRUE
```

Altri tipi di riferimenti

Tipi wrapper → Per ogni primitive c'è una wrapper class, trattati come riferimenti

- Le classi wrapper sono:
  - Byte, Short, Integer, Long, Float, Double
  - Boolean
  - Character
  - Void

Tutte immutabili e final

Ogni wrapper class ha un metodo statico value of. Prende come argomento un valore del tipo di base corrispondente e restituisce un oggetto di quel wrapper che rappresenta il valore passato come argomento.

Utile per usare oggetti già creati.

Le wrapper class numeriche estendono la classe astratta Number

Perciò nei metodi:



Estraggo il valore contenuto,  
permettendo il cast in un  
Tipo base

public byte	byteValue()
public short	shortValue()
public int	intValue()
public long	longValue()
public float	floatValue()
public double	doubleValue()

Anche in questo caso è possibile perdere dati  
(da double a int)

