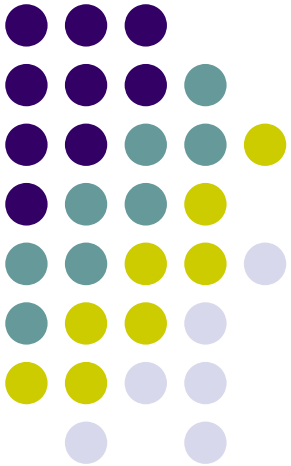
Università di Napoli Federico II - Dip. di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica

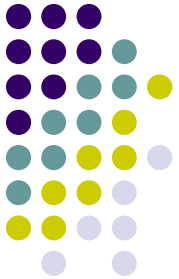
DIE TI

Corso di Programmazione

Tipi in Java



Tipi di dato in Java



- Java è un linguaggio fortemente tipizzato
- I tipi in Java appartengono a due categorie:
  - Tipi primitivi
  - Riferimenti

I tipi primitivi

C++

```
int i;
int *pi;
int &ri=i;
pi=new int;
```


Java

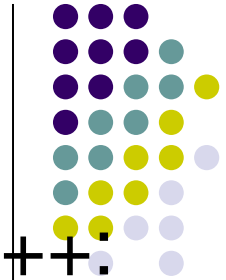
```
int i;
```

I tipi primitivi sono 8 (boolean, char, byte, short, int, long, float, double) sui quali sono disponibili le consuete operazioni

- **sono sempre allocati nell'area stack (non è possibile allocarli dinamicamente)**
- hanno una dimensione costante al variare dell'architettura sottostante
- Le variabili di istanza vengono tutte inizializzate di default (0 per i numerici/char, false per i boolean, null per le reference), le variabili locali **devono essere inizializzate o definite prima di essere utilizzata per evitare possibili errori di compilazione**.

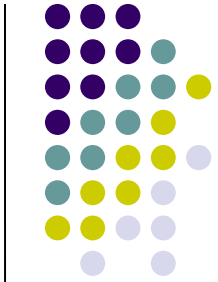
…**per l'allocazione dinamica si ricorre a delle classi Wrapper** (Boolean, Character, Byte, Short, Integer, Long, Float, Double)

Tipi primitivi



- Si "comportano" come i tipi primitivi del C/C++:
  - *una variabile di un tipo primitivo contiene direttamente il valore del dato*
  - *La dichiarazione della variabile x di tipo int alloca la memoria necessaria per contenere un int*
  - *Un assegnamento alla variabile x scrive un valore nella memoria precedentemente allocata cancellando il valore precedente*
  - *L'assegnamento di x a y copia il contenuto della variabile (il valore)*
  - *La modifica di y non modifica x*

Costanti



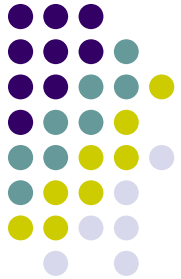
- Le costanti vengono dichiarate utilizzando la parola chiave *final*

```
final double VAL = 0.43;
```

- I nomi delle costanti sono tipicamente in maiuscolo
- E' lecito non inizializzare una costante all'atto della sua dichiarazione, ma in questo caso è possibile fare una sola assegnazione

```
final double VAL;
VAL = 0.43
//da questo momento VAL non può più essere modificata
--> errore di compilazione
```

Costanti in libreria



- Esistono molte costanti già definite nelle librerie Java
- Ad esempio: `Math.PI` in `java.lang`
- Può essere pesante dover specificare il ***nome qualificato*** della classe tutte le volte che si utilizza una costante
- In Java è possibile utilizzare per le costanti un meccanismo di importazione analogo a quello utilizzato per le classi:

```
import static java.lang.Math.PI;
```

- E' possibile in questo modo chiamare le costante soltanto *PI*

Il mondo dei riferimenti

C++

```
string c;
string *pc;
string &rc=c;
pc=new string ("Ciao");
```

Java

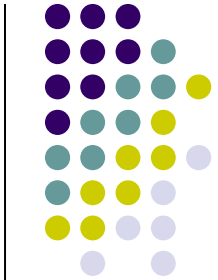
```
/*c è un riferimento ad
un tipo String*/
String c;

c=new String ("Ciao");
```

Non esistono, in Java, oggetti di tipo “complesso” ma solo riferimenti a tali oggetti. La creazione di un oggetto si articola in due fasi:

1) definizione del riferimento
2) inizializzazione ed allocazione dell’oggetto mediante **new**

...Ogni oggetto java deriva dall’oggetto java.lang.Object

Riferimenti



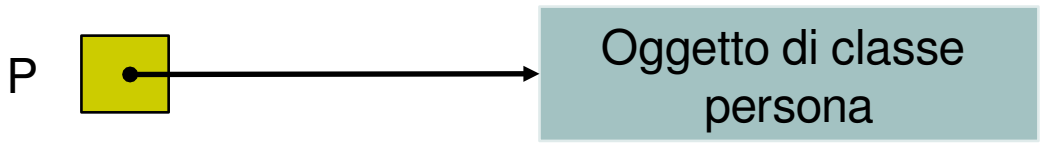
- Un riferimento si "comporta" come un puntatore, si riferisce ad un oggetto di una specifica classe. Per fissare le idee sia P un riferimento ad un oggetto di classe persona:
- *La dichiarazione della variabile P alloca la memoria necessaria per contenere un riferimento (inizializzato a **null**):*

```
persona P;
```

P | null

- *la creazione dell'oggetto di classe persona alloca un nuovo oggetto e assegna un riferimento alla variabile P:*

```
P = new persona("mario", "rossi");
```

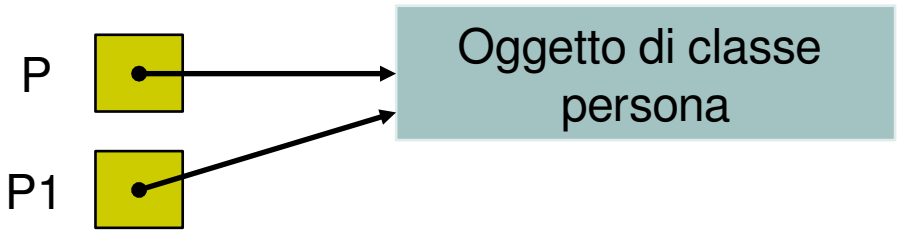


Riferimenti

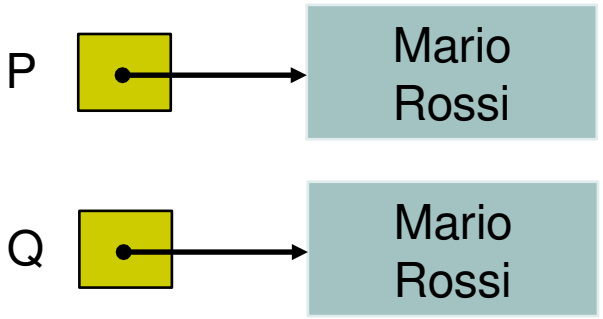
- Quindi P non ha come valore lo stato dell'oggetto persona ma si riferisce all'area di memoria dell'oggetto
- Questo ha un certo numero di conseguenze

Sia: persona P1;

1. L'assegnazione: P1=P copia P in P1, cioè copia i riferimenti!!!! Quindi l'effetto è avere due riferimenti diversi allo stesso oggetto
2. Il confronto: P==P1 confronta il valore dei riferimenti e NON gli oggetti, quindi risulta falso se P e P1 si riferiscono a due oggetti diversi anche se questi hanno lo stesso stato

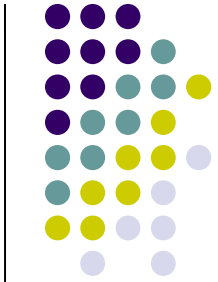


Situazione 1



Situazione 2

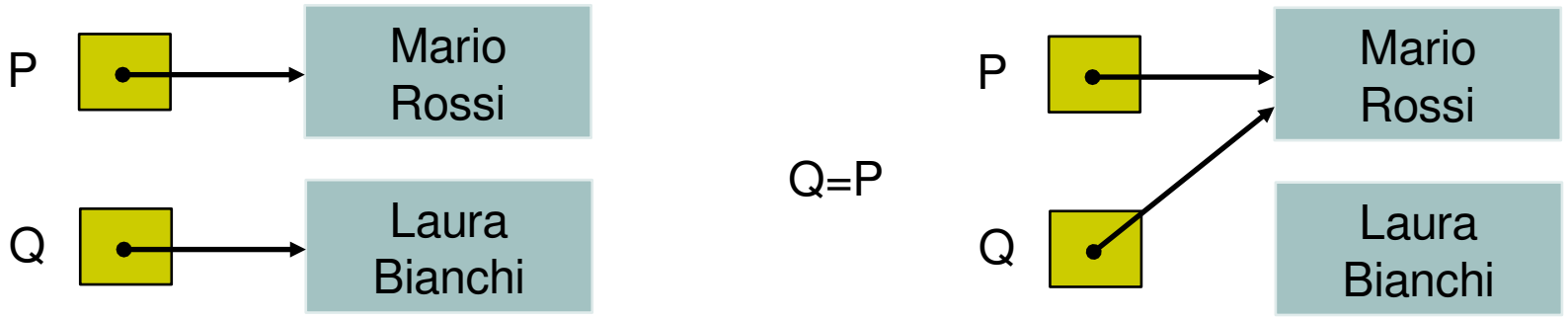
Riferimenti



- Le stesse considerazioni valgono se si passa un "oggetto" come parametro (argomento) ad una funzione (metodo)
- Si passa in realtà il valore di un riferimento all'oggetto
- *Lo stesso discorso vale per gli array e per le stringhe (gli array e le stringhe sono oggetti!)*

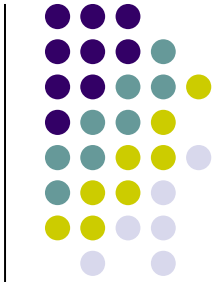
Riferimenti

- Un'altra conseguenza dell'uso dei riferimenti è che si possono ottenere «oggetti orfani», cioè privi di qualunque riferimento



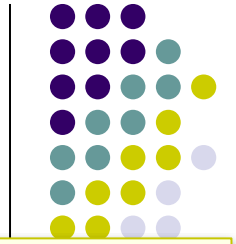
- Come arrivo più a Laura Bianchi? Questo oggetto è «orfano»….. E quindi non è più utilizzabile!!!! Ma occupa un'area di memoria….

Riferimenti



- Un oggetto non più utilizzabile è *garbage, (spazzatura)*
- *Il linguaggio Java (come molti linguaggi moderni) prevede un meccanismo di rimozione degli oggetti privi di riferimenti detto Garbage Collector*
- *Il garbage collector viene eseguito periodicamente dalla Java Virtual Machine. Interrompe per un attimo l'esecuzione del programma e pulisce la memoria dagli oggetti privi di riferimenti*

Esempio-Esercizio

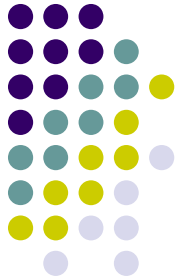


```
// RIFERIMENTI AD OGGETTI E
// CONFRONTO
class Prova {
 public static void main(String args[]) {
       String str1, str2;
       str1= new String("JAVA");
       //str2=str1;

       // Qual e' l'output se sostituiamo la seguente
       // linea con la precedente ?
       str2=new String("JAVA");

       System.out.println("Stesso valore ?"+str1.equals(str2));
       System.out.println("Stesso oggetto ?"+(str1==str2));
  }
}
```

Riferimento *null*

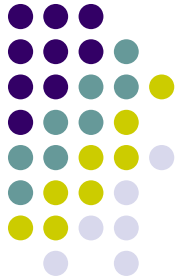
- Un riferimento potrebbe anche non riferire alcun oggetto. Si utilizza un valore speciale, indicato dalla parola riservata **`null`**.
- **`null`** non vale -1: il suo valore non importa!
- Questo valore non rappresenta alcun oggetto, ed è il valore di inizializzazione di default per le variabili riferimento.
- Assegnare il valore **`null`** a un riferimento significa rilasciare l'oggetto corrispondente (interviene il *garbage collector*)
- Se si cerca di utilizzare un oggetto mediante un riferimento pari a null, il compilatore ritorna un errore (NullPointerException).

Tipi primitivi in Java



Tipo	Dimensione in bit	Valori	Standard
boolean		`true` o `false`	
char	16	da `'\u0000'` a `'\uFFFF'` (da 0 a 65535	Insieme dei caratteri ISO Unicode
byte	8	da -128 a +127 [da $-(2^7)$ a 2^7-1]	
short	16	da $-(2^{15})$ a $2^{15}-1$	
int	32	da $-(2^{31})$ a $2^{31}-1$	
long	64	da $-(2^{63})$ a $2^{63}-1$	
float	32	Numeri negativi: da -3.40…E+38 a -1.40…E-45 Numeri Positivi: da 1.40…E-45 a 3.40…E+38	IEEE 754 (numeri in virgola mobile)
double	64	Numeri negativi: da -1.79…E+308 a -4.94…E-324 Numeri Positivi: da 4.94…E-324 a 1.79…E+308	IEEE 754 (numeri in virgola mobile)

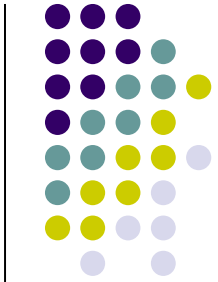
Tipi primitivi in Java



- La rappresentazione di un boolean è specifica per la JVM di ogni piattaforma.
- È possibile utilizzare i trattini bassi per rendere più leggibili i valori letterali numerici.
  - Per esempio 1_000_000 è equivalente a 1000000
- Una variabile boolean può assumere solo il valore `true` o `false`. Il seguente codice genera errore:

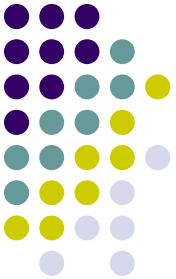
```
if (12) {
      System.out.println("ramo True");
}
```

Type casting dei tipi primitivi in Java



- Il type casting avviene quando si assegna un valore di un tipo di dati primitivo a un altro tipo di dato sempre primitivo.
- In Java ci sono due tipi di casting
  - **Widening Casting (automatico)** - converte un tipo più piccolo in un tipo di dimensioni più grandi
    - byte -> short -> char -> int -> long -> float -> double
  - **Narrowing Casting (manuale)** - converte un tipo più largo in un tipo di dimensioni più piccole
    - double -> float -> long -> int -> char -> short -> byte

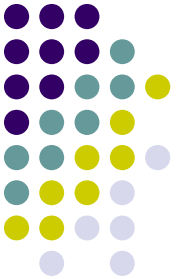
Esempio Widening Casting



```
public static void main(String[] args) {
    char mioCarattere = '{';
    //Widening Casting
    int mioIntero = mioCarattere;
    System.out.println( x: mioIntero);

}
```

```
Output - MioRunning (run)
   run:
   123
   BUILD SUCCESSFUL (total time: 0 seconds)
```

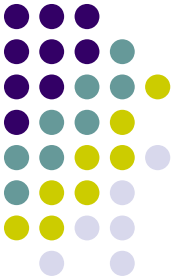
Esempio Narrowing Casting



```
public static void main(String[] args) {
    int mioIntero = 126;
    //Narrowing Casting
    char mioCarattere = (char) mioIntero;
    System.out.println( x: mioCarattere);

}
```

```
Output - MioRunning (run)
run:
~
BUILD SUCCESSFUL (total time: 0 seconds)
```

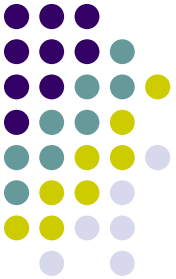
Esempi di uso di caratteri Unicode in Java



```
public static void main(String[] args) {
    char carattere1 = '\u0041';
    char carattere2 = '\u007B';
    System.out.println( x: carattere1);
    System.out.println( x: carattere2);
}
```

```
Output - MioRunning (run)
run:
A
{
BUILD SUCCESSFUL (total time: 0 seconds)
```

Esempio di operazioni tra variabili

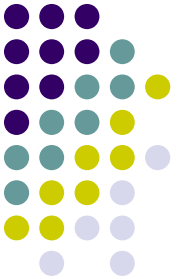


```
public static void main(String[] args) {
    int x=4;
    int y=3;
    float ris=x/y;
    System.out.println( x: ris);
}
```

Qual è il risultato?

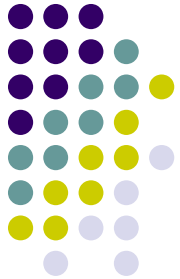
```
run:
1.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Una possibile soluzione



```
public static void main(String[] args) {
    int x=4;
    int y=3;
    float ris=(float)x / (float) y;
    System.out.println( x: ris);
}
```

Altro casting da intero a char



```
public static void main(String[] args) {
    int a = 2000000000;
    char c = (char) a;
    System.out.println( x: c);
}
```

Qual è il risultato?

```
run:
鐨
BUILD SUCCESSFUL (total time: 0 seconds)
```

Perché?

2000000000%65536 = 37888 che in esadecimale vale 9400 ossia la codifica Unicode dell'ideogramma

Riferimenti