

JAVA

Sintassi e comandi base

Il main program

L'inizio di un programma java è la funzione **main**

```
public class Prova
{
    public static void main(String[] args)
    {
        istruzione
        istruzione
        ...
    }

    funzione1()

    funzione2()
}
```

E' l'entry point
della **JVM**



Tutte le funzioni devono essere scritte all'interno della classe

Stile di codifica

- Spaziature: per separare gli elementi è necessario utilizzare almeno uno spazio
- Ogni istruzione finisce con “;”
- Si possono scrivere molte istruzioni nella stessa riga, ma è consigliabile una riga per ogni dichiarazione o istruzione.

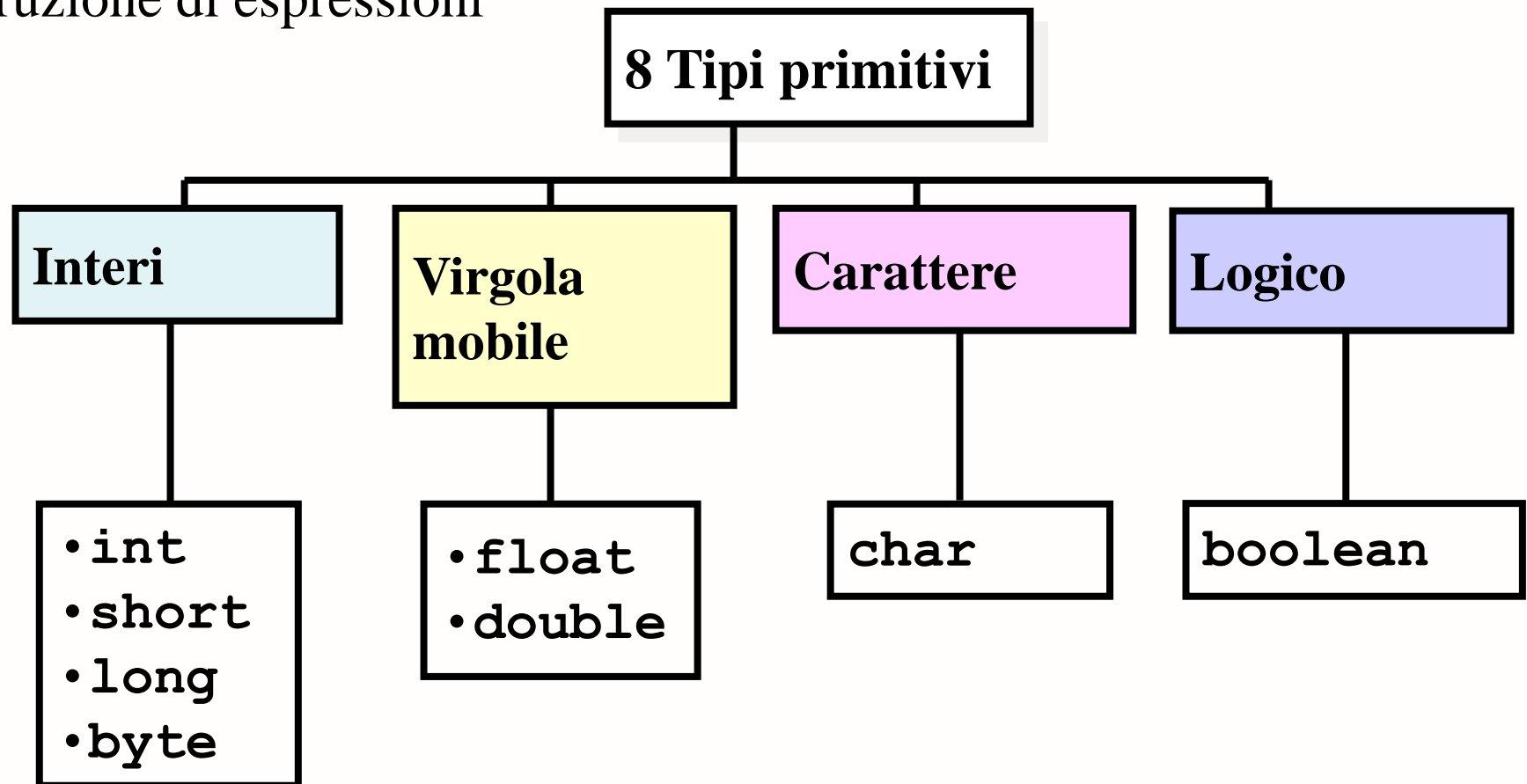
Commenti

Esistono tre diversi stili:

- `//` per commenti di una riga
- `/*` blocco di commenti
(più righe) `*/`
- `/**` Commenti javadoc:
 - * invocando uno speciale tool (javadoc.exe)
 - * sul codice commentato con questo stile
 - * viene prodotta una documentazione in HTML con
 - * formattazione standard della Sun
 - *
* Si utilizzano inoltre speciali tag per definire
 - * caratteristiche delle classi e dei metodi
 - * `/`

Tipi primitivi

Sono i tipi definiti nel linguaggio e dai quali si può partire con la costruzione di espressioni



Tipi Interi

Tipo	Requisiti di memorizzazione	Intervallo (inclusivo)
int	4 byte	Da -2.147.483.648 a + 2.147.483.647
short	2 byte	Da -32.768 a + 32.767
long	8 byte	- 9.223.372.036.854.775.808L a + 9.223.372.036.854.775.807L
byte	1 byte	Da - 128 a + 127

Tipi in virgola mobile

Tipo	Requisiti di memorizzazione	Intervallo (inclusivo)
float	4 byte	circa $\pm 3.40282347\text{E}+38$ F (6-7 cifre decimali significative)
double	8 byte	circa $\pm 1.79769313486231570\text{E}+308$ (15 cifre decimali significative)

Tipo carattere

- Si usa il delimitatore: singolo apice.

```
char c1 = 'A' ;
```

```
char c2 = 'a' ;
```

```
char nullo = '\0' ;
```

```
char nullo = '\u0000' ;
```

- E' un carattere dello schema **Unicode** (raccolge tutti i caratteri esistenti), si rappresenta su 2 bytes.
- Esistono anche delle sequenze di escape

Sequenza di escape	Nome	Valore Unicode
\b	backspace	\u0008
\t	tabulazione	\u0009
\n	nuova riga	\u000a
\r	ritorno carrello	\u000d
\"	virgolette doppie	\u0022
\'	virgolette singole	\u0027
\\	back slash	\u005c

Tipo logico

- Assume soltanto due valori: **true** o **false**
- Non si usano apici, né virgolette

Esempio:

```
boolean flag = false;
```

- Si usano nelle istruzioni di controllo di flusso (if, while, do)
- Gli operatori comparativi restituiscono valori booleani.

Dichiarazione variabili

- Tutte le variabili devono essere dichiarate
- La variabile viene definita da:
 - un identificatore
 - da un tipo

Regole degli identificatori

- sono case-sensitive
- possono contenere lettere, numeri, _, \$
- non possono iniziare con un numero

Convenzioni degli identificatori

- Inizia sempre con la lettera minuscola
- Se ci sono più parole si alterna minuscolo-maiuscolo, in questo modo:
Es, `int maxValoreCalcolato;`

Blocchi di istruzioni

- Sono contenuti tra

```
{  
    statement 1  
    statement 2  
}
```

- Racchiudono istruzioni arbitrarie

Non è possibile utilizzare lo stesso nome per variabili in blocchi annidati:

NB: In altri linguaggi di programmazione è permesso


```
{  
    int a =10;  
    {  
        int a =20; //NO!  
    }  
}
```

Scope di una variabile

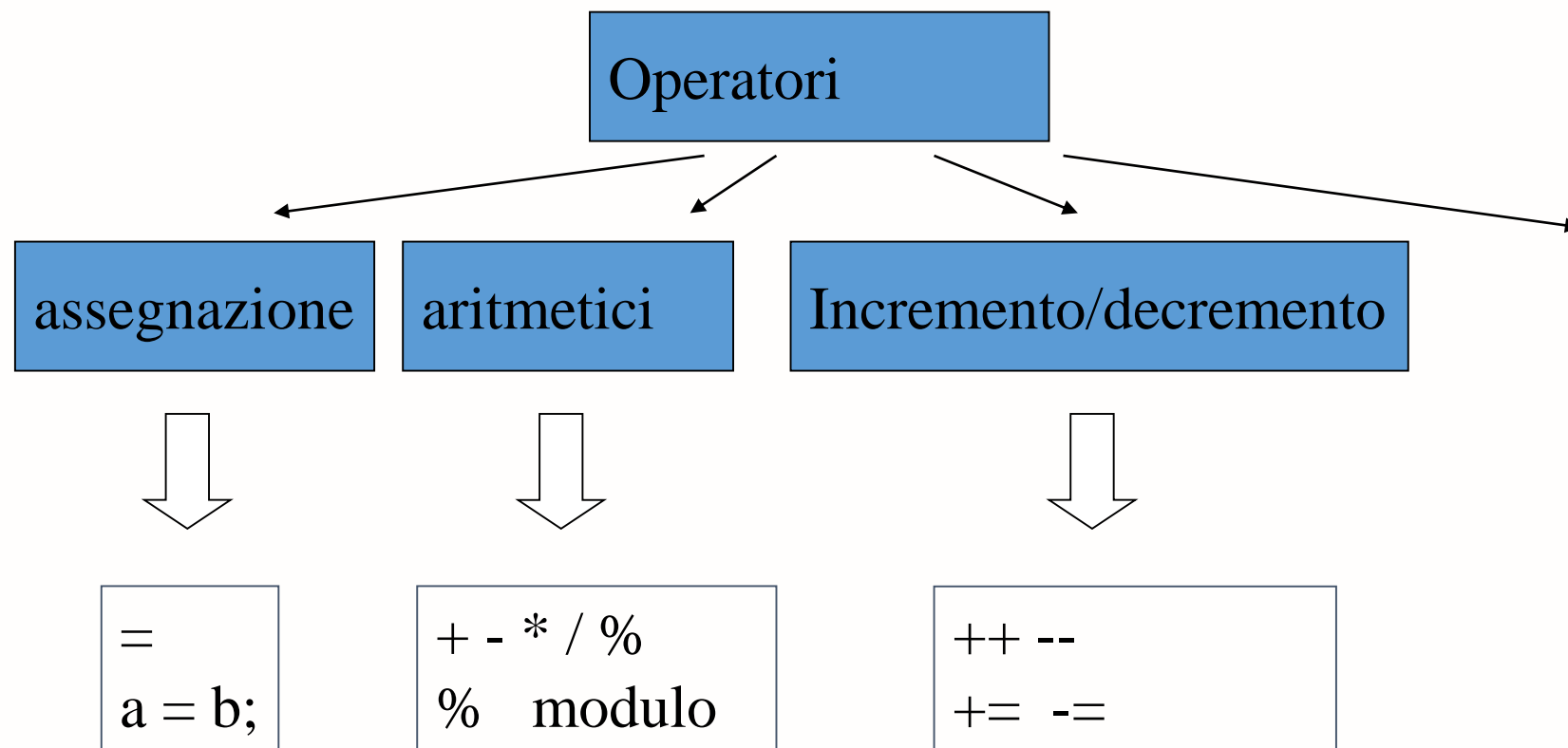
- Lo spazio di esistenza (e di visibilità) di una variabile è il suo **SCOPE**
- Oltre lo scope di una variabile, la memoria viene deallocata.

```
{  
    statement 1;  
    statement 2;  
    int a;  
    {  
        statement 3;  
        statement 4;  
    }  
    statement 5;  
    statement 6;  
    a = 10; //OK!  
}
```

Scope di a



Operatori



Operatori

Attenzione alla divisione!

Se gli operandi sono interi
il risultato è comunque
intero

Se almeno un operando è
decimale il risultato è
decimale

```
{  
int a=2;  
int b=3;  
int c=0;  
double e=2;  
double f=3;  
double d =0;  
  
c=b/a;  
System.out.println("c= "+c); // c = 1  
d=b/a;  
System.out.println("d= "+d); // d = 1.0  
d=f/e;  
System.out.println("f/e= "+d); // d=1.5  
d=b/e;  
System.out.println("b/e= "+d); // d=1.5  
}
```

Cast

```
int a;  
int b = 10;  
a = b;
```

```
int a = 4;  
short b = 10;  
a = b;
```

```
byte a;  
int b = 10;
```

```
a = b; // NO!!
```

```
a = (byte)b; // SI! Ma i rischi  
// sono consapevoli
```

Quanto vale a in questi esempi?

Questa istruzione è potenzialmente pericolosa.
Infatti il compilatore dà errore

Java impone al programmatore
di fare scelte consapevoli.



Cast

- Le conversioni vengono eseguite mediante cast (attribuzioni forzate di tipo).
- La sintassi del cast consiste nell'indicare il tipo di destinazione tra ()

Promozione automatica

Nelle operazioni tra tipi diversi, il risultato è del tipo più alto

`int + byte = int`

`int + short = int`

`int + double = double`

.....

Tranne nel caso della ***promozione automatica***

`byte + byte = int`

`float + float = double`

cioè si effettua la promozione al tipo di riferimento

Esempio di promozione:

`byte a = 18; byte b = 10;`

`byte c = a + b; // errato perché a+b viene di tipo int`

`int c = a + b; // giusto oppure byte c = (byte)(a+b);`

Tipi di riferimento:
int per gli interi
double per i decimali

Operatori relazionali

- Permettono il confronto tra due valori che stabiliscono relazioni di uguaglianza e d'ordine.

Operatore	Risultato
$=$	uguale a
\neq	diverso da
$>$	maggiore
$<$	minore
\geq	maggiore o uguale
\leq	minore o uguale

Operatori relazionali

- Java adotta simboli diversi per le assegnazioni e per la verifica dell'uguaglianza.

`(a == 7)` darà **true** se `a` vale 7 e **false** diversamente

`a = 7` assegna 7 alla variabile `a`

NOTA:

`if (a=7) // NON compila!!`

Operatori booleani

Operatore	Significato
 	OR sc
&&	AND sc
!	NOT
^	XOR
 	OR
&	AND

Esempio:

Supponiamo che

`a = 7;`

`b = 10;`

Allora

`(a == 7) && (b > 0) → true`

`(a == 7) || (b > 0) → true`

`!(a > 4) → false`

`(a == 7) ^ (b > 0) → false`

`sc → ShortCircuited`,
 cioè ottimizza controllo delle
 condizioni in OR e in AND

Operatori booleani

Esempio senza ShortCircuited:

```
int denom =0;  
int num=7;  
if ( denom != 0 & num / denom >10) a=5;  
System.out.println("num = "+num);
```

Risultato:

Exception in thread "main" java.lang.ArithmeticException: / by zero

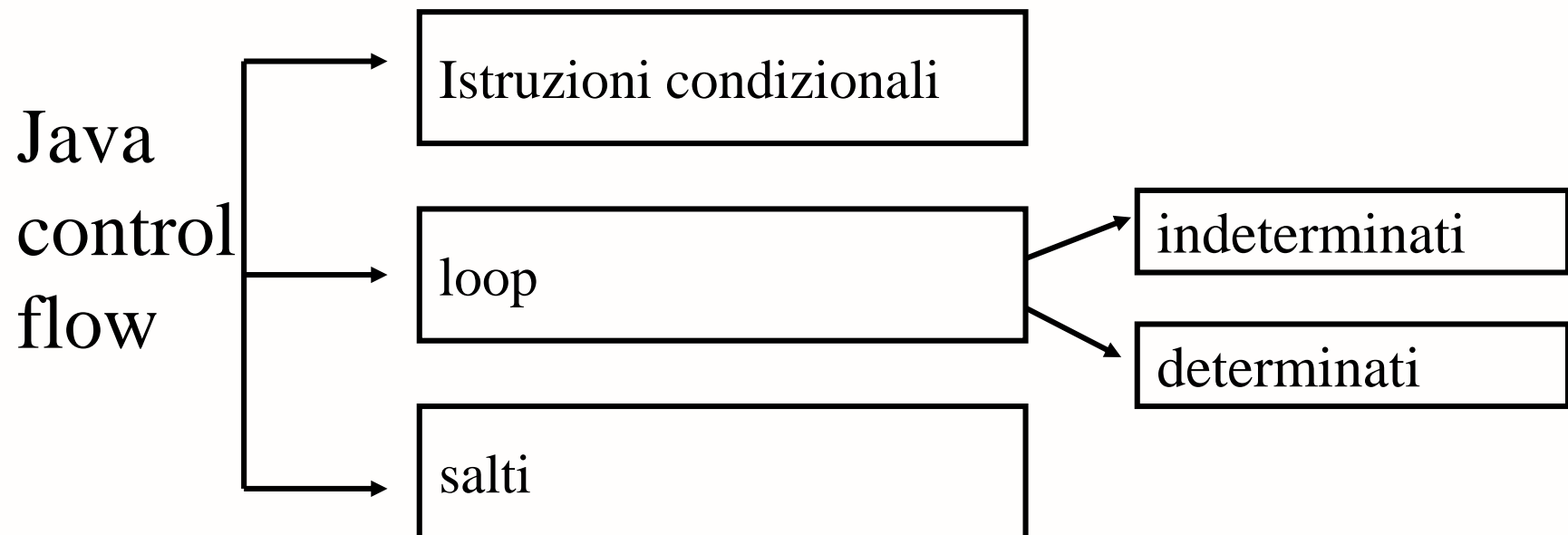
Esempio con ShortCircuited:

```
int denom =0;  
int num=7;  
if ( denom != 0 && num / denom >10) a=5;  
System.out.println("num = "+num);
```

Risultato:

num = 7

Controllo di flusso



Istruzioni condizionali

```
if (condizione)
{
    istruzione1;
    istruzione2;
    ...
}
```

oppure

```
if (condizione)
    istruzione;
```

La condizione deve essere sempre tra parentesi.

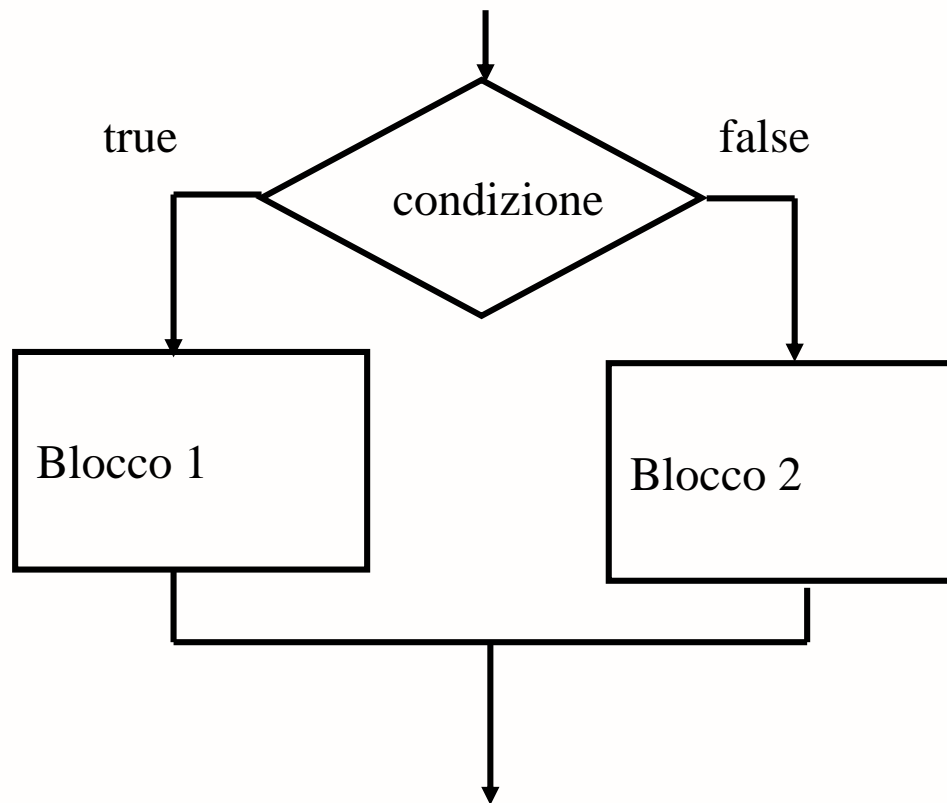
Una sola istruzione → le parentesi graffe possono essere omesse

In generale:

```
if (condizione)
{
    blocco 1;
}
else
{
    blocco 2;
}
```

NOTA:
Il blocco **else** è
opzionale

Istruzioni condizionali



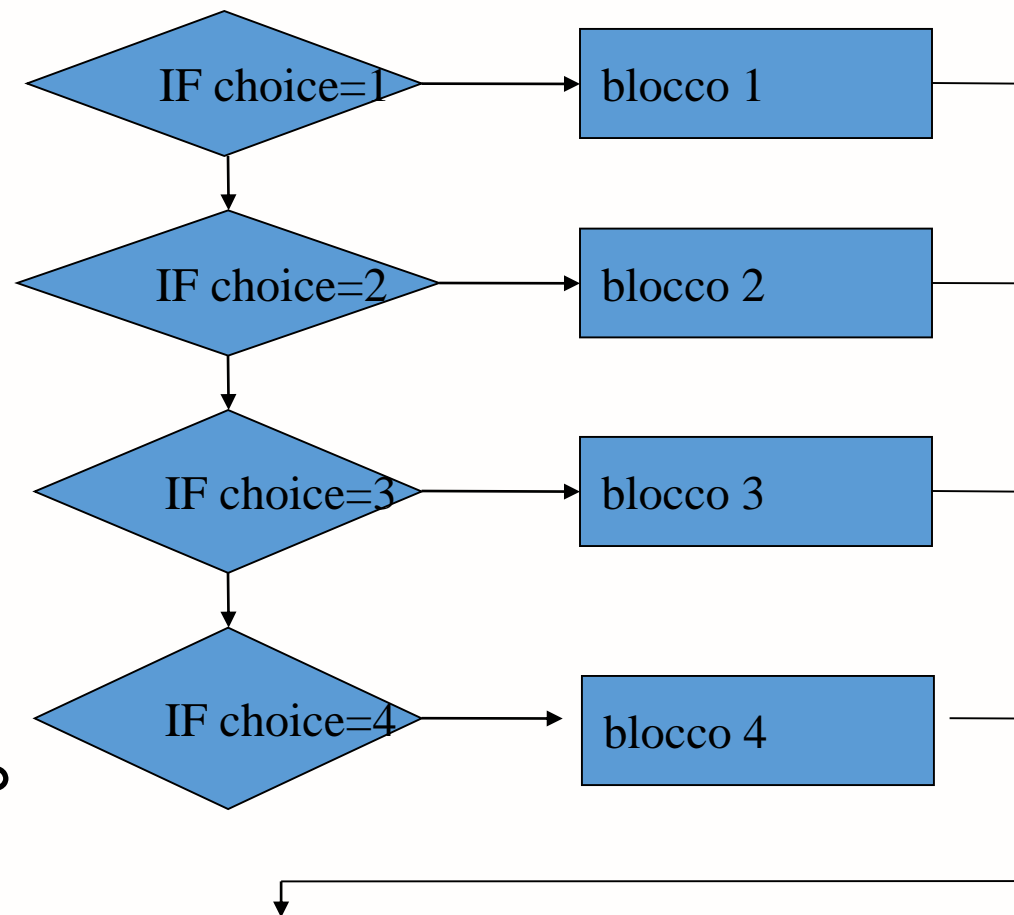
Istruzione switch

- Dispositivo di condizione multipla
- Utilizzabile solo con
 - valori di tipo **char**
 - valori di tipi numeri **int**
 - valori di tipi **enumeration**
- **No** tipi numeri long
- **No** intervalli di valori
- **No** Stringhe -/ **si da java 7 in poi !!**

Istruzione switch

Se **choice** è un **int** letto da tastiera

```
switch (choice)
{
    case 1:
        ...
        break;
    case 2:
        ...
        break;
    case 3:
        ...
        break;
    case 4 :
        ...
        break;
    default:
        // input errato
        ...
}
```



Loop indeterminati

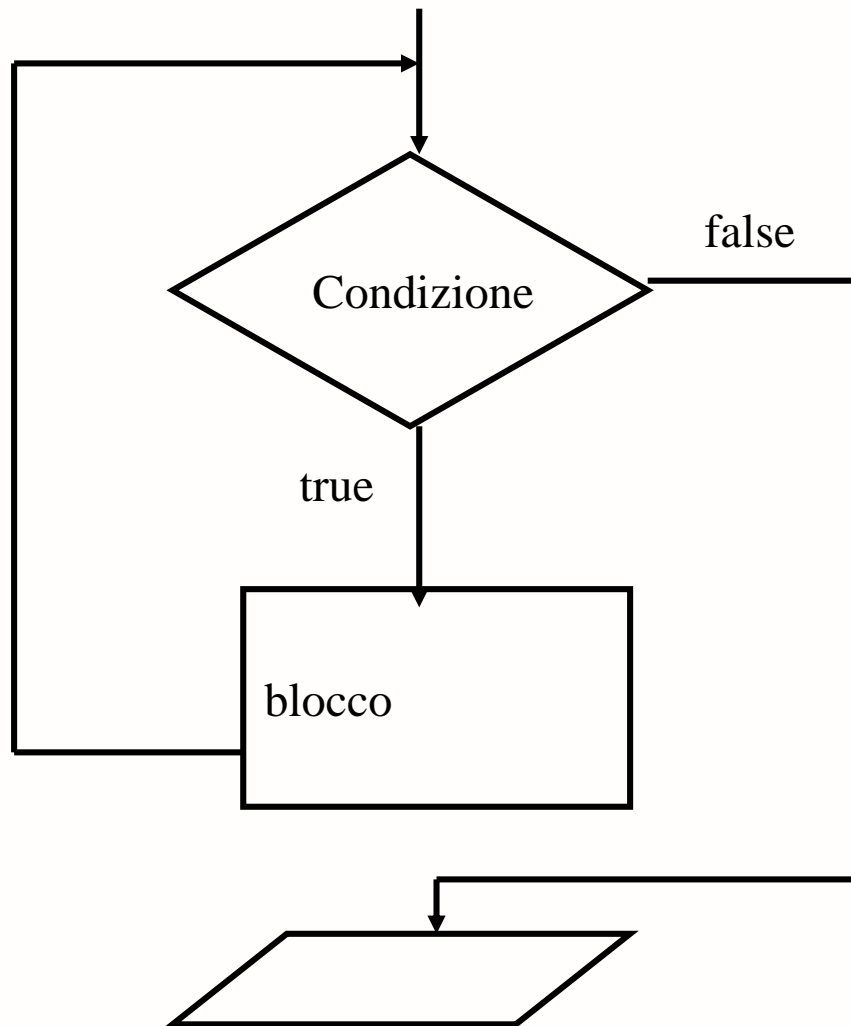
Il loop **while** esegue una verifica iniziale.

```
while (condizione)
{
    ...    blocco istruzioni
}
```

- Se la condizione iniziale è **false**, non viene eseguito
- Il loop while esegue il blocco finché la condizione iniziale è **true**
- Per eseguire almeno una volta il blocco occorre usare il **do-while** :

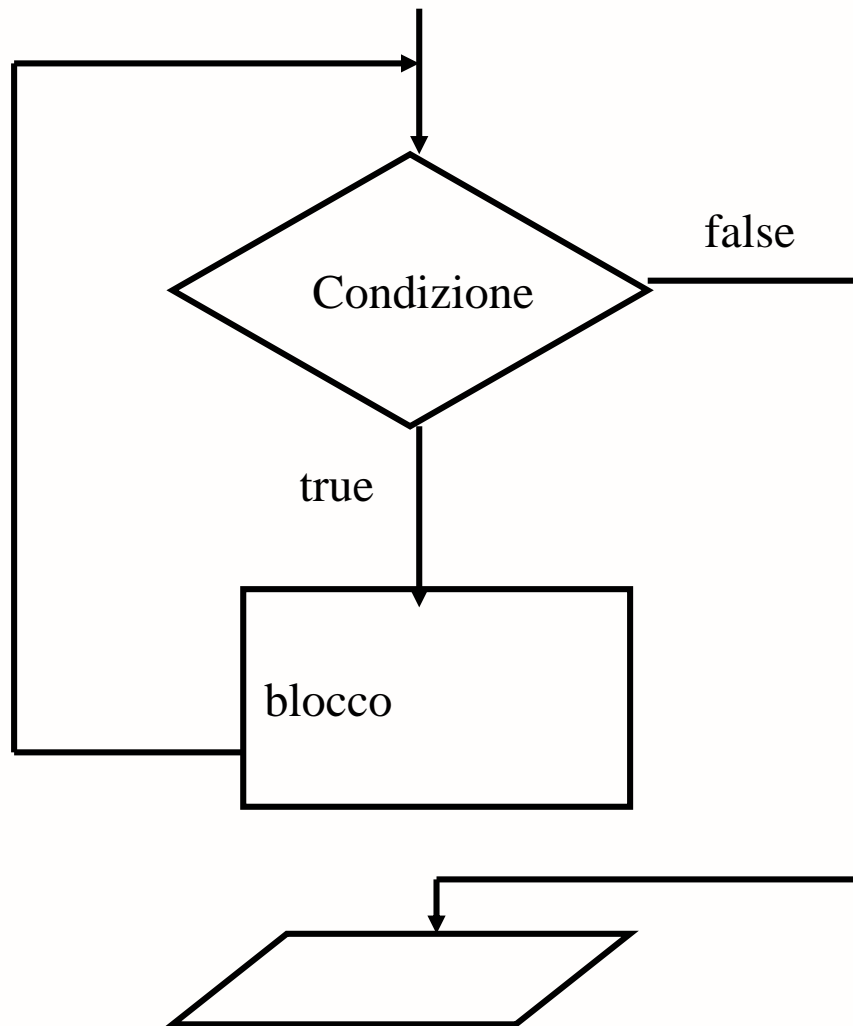
```
do
{
    ...    blocco istruzioni
}
while (condizione);
```

Il while



```
class WhileEsempio
{
    public static void main(String[] args)
    {
        int i = 1;
        while (i <= 11)
        {
            System.out.println("Contatore = " + i);
            i++;
        }
    }
}
```

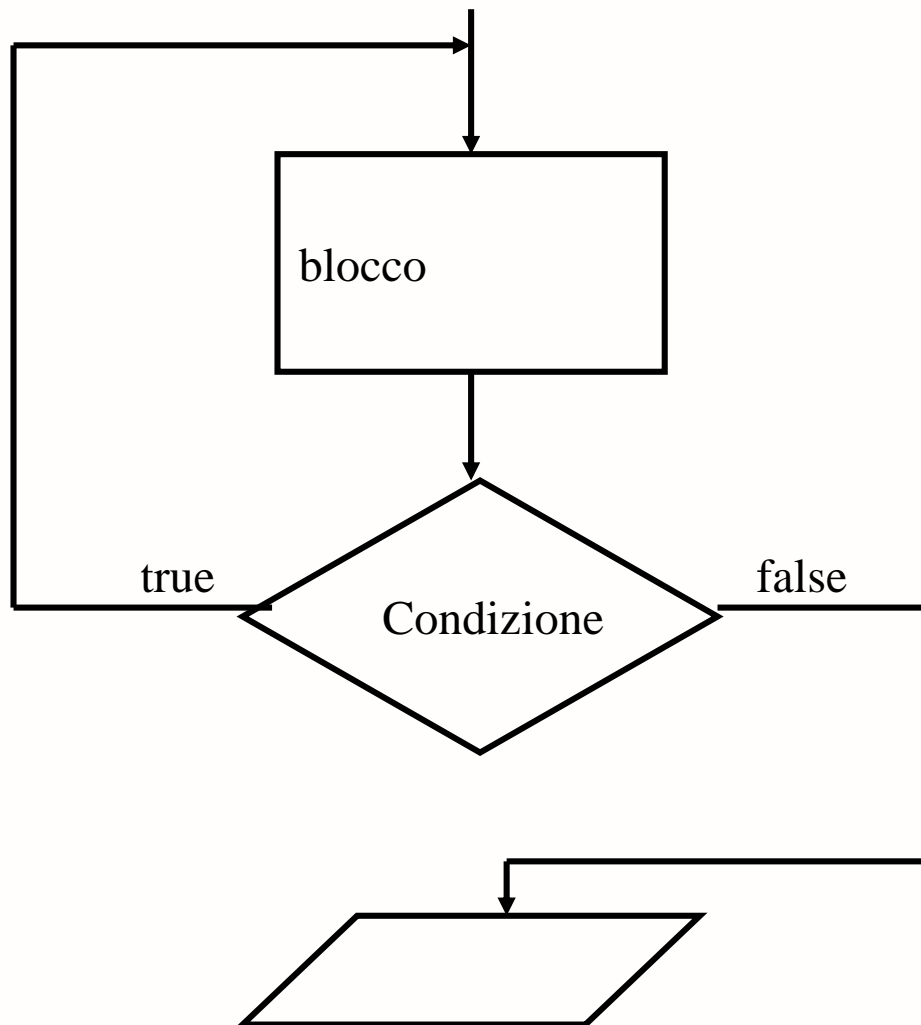
Il while



```
class WhileEsempio
{
    public static void main(String[] args)
    {
        int i = 1;
        while (i <= 11)
        {
            System.out.println("Contatore = " + i);
            i++;
        }
    }
}
```

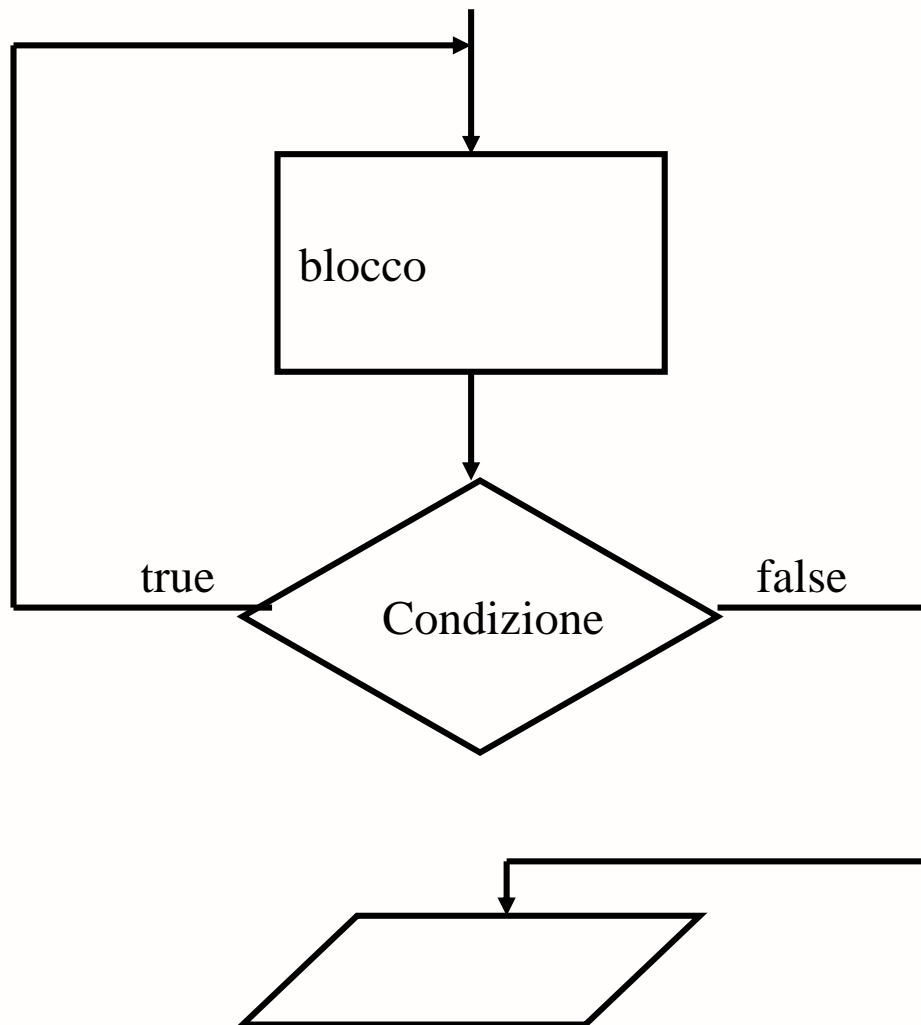
Contatore= 1
Contatore= 2
Contatore= 3
Contatore= 4
Contatore= 5
Contatore= 6
Contatore= 7
Contatore= 8
Contatore= 9
Contatore= 10
Contatore= 11

Il do - while



```
class DoWhileEsempio
{
public static void main(String[] args)
{
    int i = 1;
    do
    {
        System.out.println("Contatore = " + i);
        i++;
    }
    while (i <= 11);
}
```

Il do - while



```
class DoWhileEsempio
{
    public static void main(String[] args)
    {
        int i = 1;
        do
        {
            System.out.println("Contatore = " + i);
            i++;
        }
        while (i <= 11);
    }
}
```

Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
Count is: 11

Loop determinati

```
for ( inizio; iterazione; passo)
{
    istruzione1;
    istruzione2;
    istruzione3;
}
```

Il ciclo si ripete un **numero prefissato** di volte, secondo i parametri dati da:

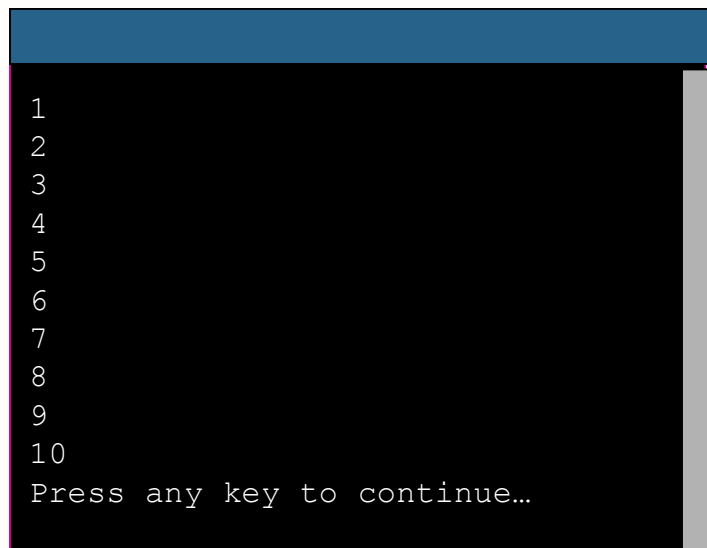
- **inizio ciclo**
- **iterazione del ciclo**
- **passo**

La variabile per ciclare si può dichiarare dentro le parentesi tonde

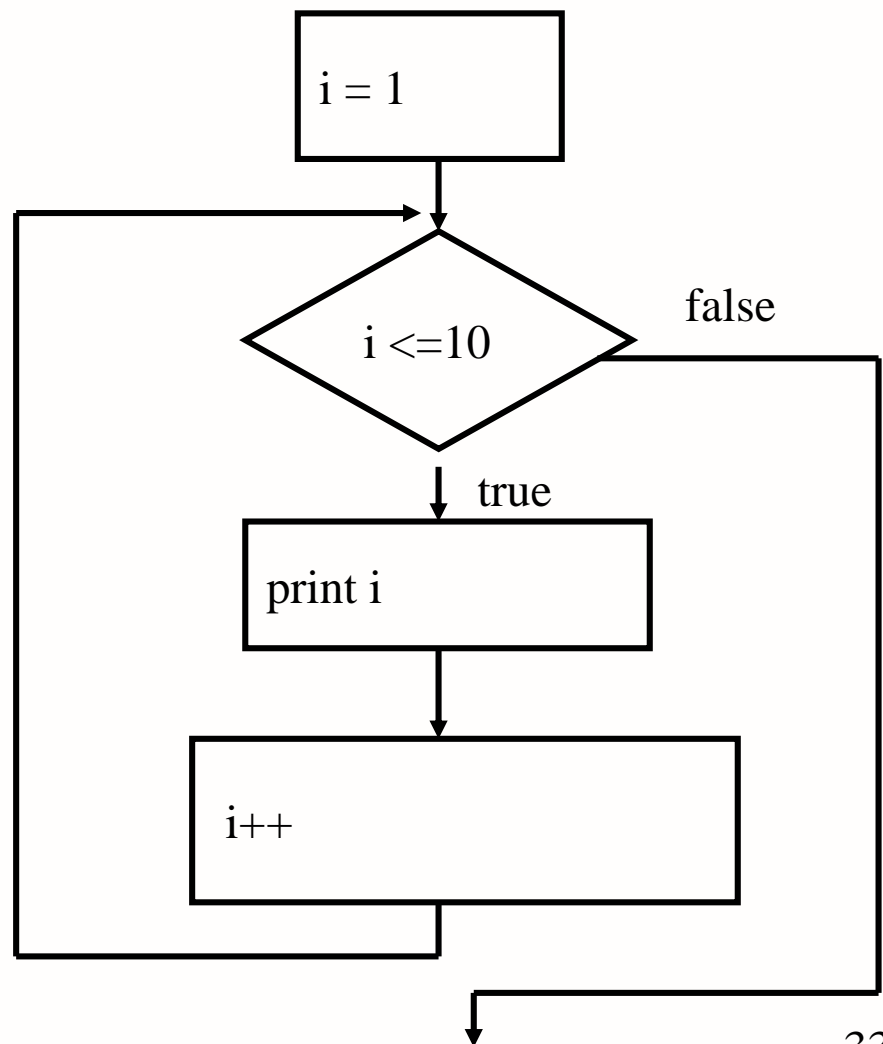
Loop determinati

```
for (int i = 1; i <=10; i++)  
{  
    System.out.println(i);  
}
```

Stampa sullo schermo i numeri da 1 a 10

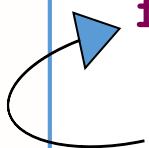


```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
Press any key to continue...
```



continue

- Consente di saltare all'esterno del ciclo dove compare
- Per effettuare un salto di 2 cicli, è necessario il costrutto `label-continue`

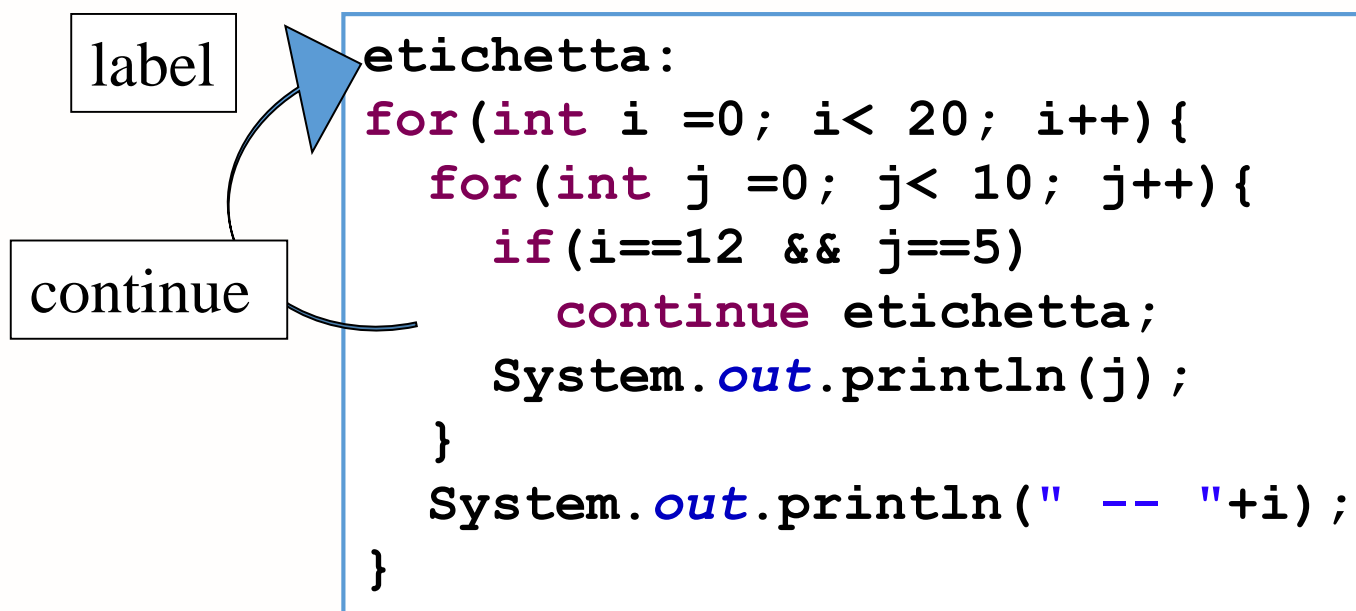


```
for(int i =0; i< 20; i++){  
    for(int j =0; j< 10; j++){  
        if(i==12 && j==5)  
            continue;  
        System.out.println(j);  
    }  
    System.out.println("  --  "+i);  
}
```

In corrispondenza di $i=12$ e $j=5$ non viene effettuata la stampa di $j = 5$

label

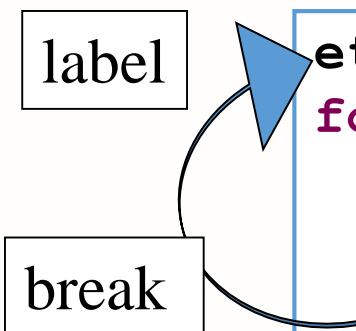
- Consente di etichettare un ciclo e di saltare al punto dell'etichetta con l'istruzione **continue**
- L'etichetta deve avere un nome composto da una sola parola e deve essere seguito da ":"



In corrispondenza di $i=12$ e $j=5$ si salta all'iterazione con $i=13$ e non viene effettuata la stampa di j 5...9

break

- Consente di etichettare un ciclo e di interromperlo al punto dell'etichetta con l'istruzione **break**
- L'etichetta deve avere un nome composto da una sola parola e deve essere seguito da ":"



```
etichetta:
for(int i =0; i< 20; i++){
    for(int j =0; j< 10; j++){
        if(i==12 && j==5)
            break etichetta;
        System.out.println(j) ;
    }
    System.out.println("  --  "+i) ;
}
```

In corrispondenza di $i=12$ e $j=5$ il ciclo su i (e dunque anche su j) viene terminato!!!

Passaggio parametri

- Per tutti i tipi **primitivi** il passaggio dei parametri è **per valore**.
- Le modifiche su tali variabili non sono visibili al di fuori delle funzioni alle quali vengono passate.

Esempio

```
public class PassaggioPerValore {  
    public static void main(String[] args) {  
        int n=30;  
        System.out.println("n = "+n);  
        nonMod(n);  
        System.out.println("adesso n = "+n);  
    }  
  
    public static void nonMod(int i){  
        System.out.println("    i = "+i);  
        i=0;  
        System.out.println("    adesso i = "+i);  
    }  
}
```

Tutorial della Oracle



Il sito della Oracle mette a disposizione una guida pratica per programmatori, ricca di esempi e discussioni su specifici argomenti.



<http://download.oracle.com/javase/tutorial/>

