Corso di base JAVA

Mauro Donadeo
mail: mauro.donadeo@gmail.com

Le decisioni

HT Lab

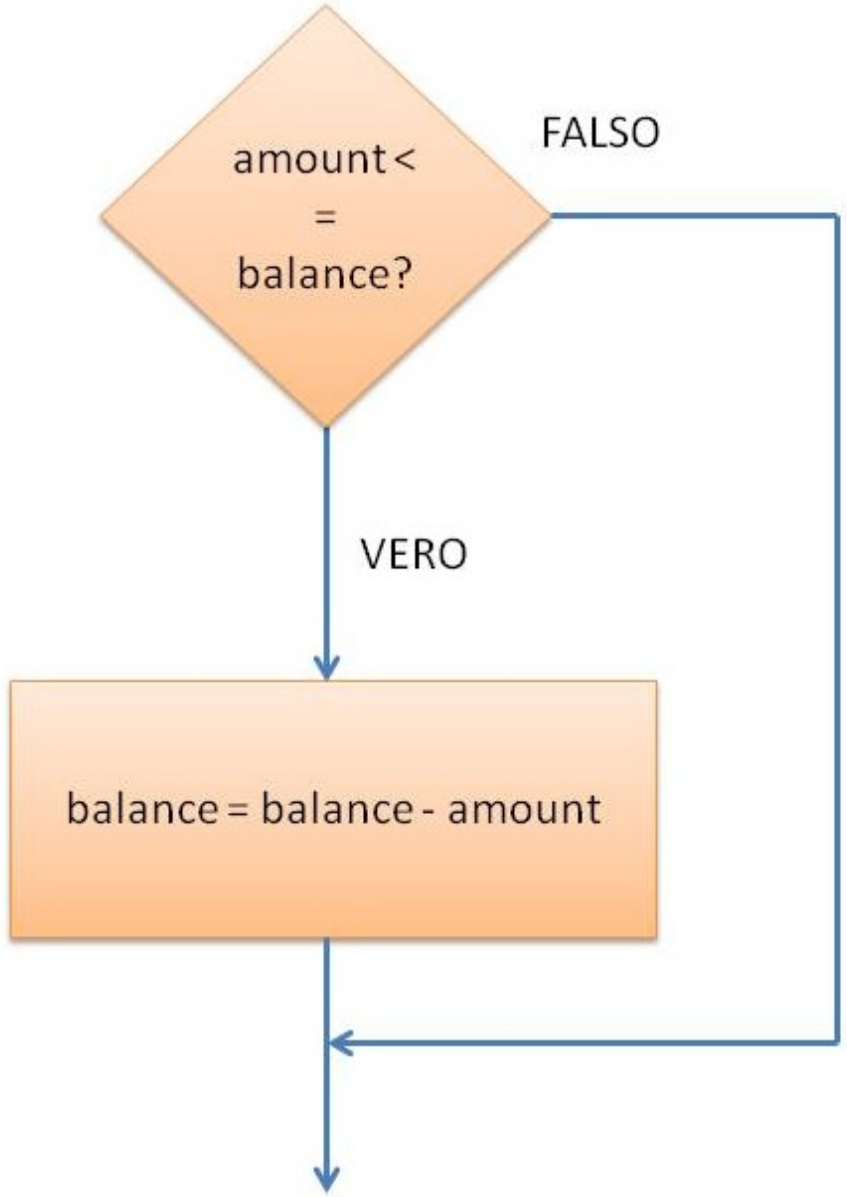
Enunciato if

Il programma **BankAccount** consente di prelevare tutto il denaro che si vuole

- il saldo `balance` può diventare **negativo**
  - **balance = balance - amount;**
- È una situazione assai poco realistica
- Quindi il programma deve **controllare** il saldo ed agire di conseguenza. *Consentire il prelievo o no.*

```
if (amount <= balance)
   balance = balance - amount;
```

- L'enunciato if si usa per realizzare una decisione ed è diviso in due parti
  - una verifica
  - un corpo
- Il corpo viene eseguito **se e solo se** **la verifica ha successo**



Tipi di enunciato in Java

- Enunciato semplice
  - **balance = balance - amount;**
- Enunciato composto
  - **if(x >= 0) x=0;**
- blocco di enunciati
  - {**zero o più enunciati di qualsiasi tipo**}

Proviamo ad emettere un messaggio d'errore in caso di prelievo non consentito:

```
if(amount <= balance)
    balance = balance - amount;
if(amount > balance)
    System.out.println("Conto scoperto");
```

Problema

Se si modifica la prima verifica, bisogna ricordarsi di modificare anche la seconda.

Problema

Se il corpo del primo `if` viene eseguito, la verifica del secondo `if` usa il nuovo valore di `balance`, introducendo **errore logico**

- quando si preleva più della metà del saldo disponibile

La clausola else

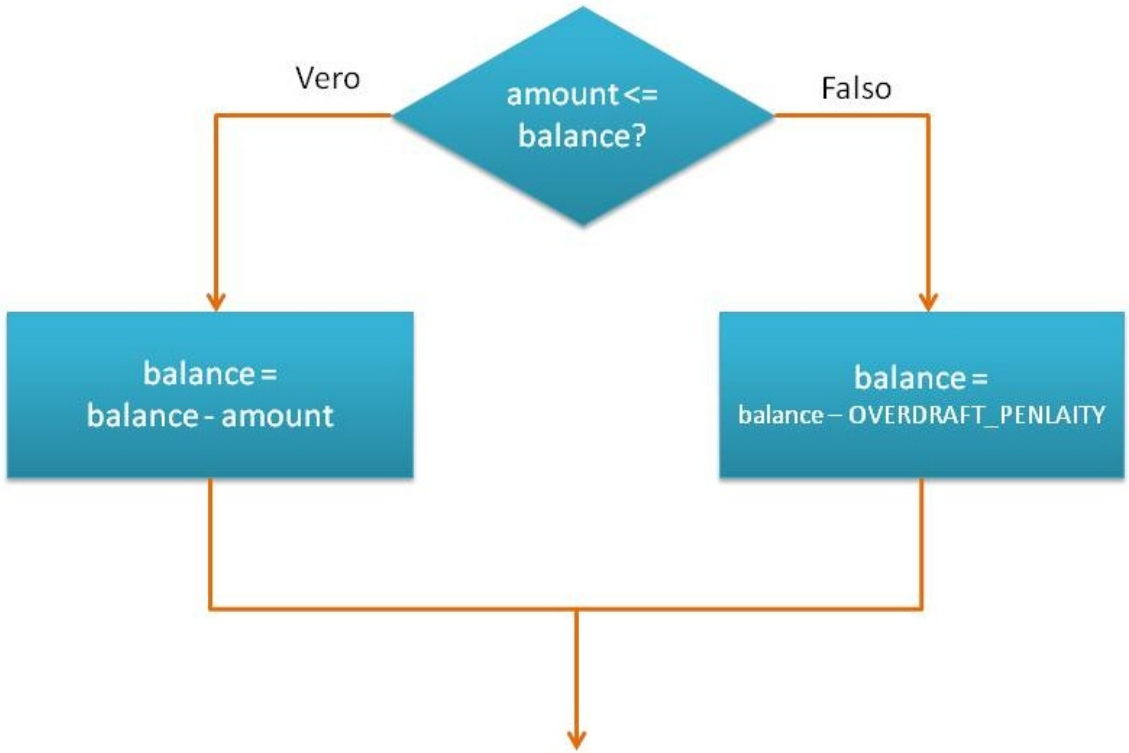
Per realizzare un'alternativa, si utilizza la clausola **else** dell'enunciato `if`

```
if(amount <= balance)
    balance = balance - amount;
else
    System.out.println("Conto aperto");
```

Vantaggio: ora c'è una sola verifica

- se la verifica ha successo, viene eseguito il primo corpo dell'enunciato `if/else`
- altrimenti, viene eseguito il secondo dopo;

```
if (amount <= balance)
    balance = balance - amount;
else{
    System.out.println("Conto scoperto");
    balance = balance - OVERDRAFT_PENALTY;
}
```



Confrontare valori

Le condizioni dell'enunciato if sono molto spesso dei confronti tra due valori

if(x >= 0)

Gli operatori di confronto si chiamano operatori relazionali

>	Maggiore
>=	Maggiore o uguale
<	Minore
<=	Minore o uguale
==	Uguale
! =	Diverso

Attenzione: negli gli operatori da due caratteri **non** vanno inseriti spazi intermedi

Operatori relazionali

Fare **molta** attenzione nella differenza tra l'operatore relazionale == e l'operatore assegnazione =

```
a = 5; //assegno ad a il valore 5

if(a == 5) //esegue enunciato
    //enunciato
```

Confrontare numeri in virgola mobile

I numeri in virgola mobile hanno una precisione limitata ed i calcoli possono introdurre errori di arrotondamento e troncamento

Tali errori sono inevitabili e bisogna fare molta attenzione nella formulazione di verifiche che coinvolgono numeri con in virgola mobile.

Affinché gli errori di arrotondamento non influenzino la logica del programma, i confronti tra numeri in virgola mobile devono avere una tolleranza

- Verifica di uguaglianza tra x ed y (di tipo double):

$$|x - y| < \epsilon \textbf{ con } \epsilon = \textbf{1E-14}$$

- Scelta migliore se x,y sono molto grandi o molto piccoli

$$|x - y| < \epsilon * max(|x|, |y|) \textbf{ con } \epsilon = \textbf{1E-14}$$

Esercizio

Calcolare la radice quadrata di 2 e confrontare il risultato con 2. Se il risultato è uguale a 2 Stampare "ok!" altrimenti "Errore"

- Per confrontare stringhe si usa il metodo **equals**

  if(s1.equals(s2))
- Per confrontare stringhe ignorando la differenza tra maiuscole e minuscole si usa **equalsIgnorecase**

  if(s1.equalsIgnorecase(s2))
- Non usare **mai** l'operatore di ugualianza per confrontare stringhe. **Usare sempre equals**
  - Attenzione perché la Virtual Machine **NON** segnalerà alcun errore di sintassi

Confronto di stringhe

Confrontando con l'operatore di uguaglianza due riferimenti a stringhe si verifica se i riferimenti puntano allo stesso oggetto stringa

```
String s1 = "Stringa";
String s2 = s1;
String s3 = "String";
s3 = s3 + "a"; // s3 contiene "Stringa"
```

Confronto di stringhe

Confrontando con l'operatore di uguaglianza due riferimenti a stringhe si verifica se i riferimenti puntano allo stesso oggetto stringa

```
String s1 = "Stringa";
String s2 = s1;
String s3 = "String";
s3 = s3 + "a"; // s3 contiene "Stringa"
```

- Il confronto **s1 == s2 è vero**, perché puntano allo stesso oggetto stringa
- Il confronto **s1 == s3 è falso**, perché puntano ad oggetti diversi, anche se tali oggetti hanno lo stesso contenuto (sono "identici")

Ordinamento lessicografico

- Se due stringhe sono diverse, è possibile conoscere la relazione che intercorre tra loro secondo **l'ordinamento lessicografico**, simile al comune ordinamento alfabetico.
- Il confronto lessicografico tra stringhe si esegue con il metodo **compareTo**

  if(s1.compareTo(s2) $<$ 0)
- Il metodo compareTo restituisce in valore `int`
  - **negativo** se s1 precede s2 nell'ordinamento;
  - **positivo** se s1 segue s2 nell'ordinamento;
  - **zero** se s1 e s2 sono identiche.

Esercizio

Scrivere un programma che chiede all'utente di inserire tre stringhe (una per riga) visualizza le stringhe in ordine lessicografico crescente (una per riga)

Confronto di oggetti

- Come per le stringhe, l'operatore == tra due variabili oggetto verifica se i due riferimenti puntano allo stesso oggetto, e non verifica l'uguaglianza tra oggetti
- Il metodo **equals** può essere applicato a qualsiasi oggetto, perché è definito nella classe **Object**, da cui derivano tutte le classi
- È compito di ciascuna classe **ridefinire** il metodo **equals**, come per la classe **String**
  - altrimenti il metodo **equals** di **Object** usa semplicemente l'operatore di uguaglianza.
- il metodo **equals** di ciascuna classe deve effettuare il **confronto delle caratteristiche** (variabili di esemplare) degli oggetti di tale classe

Sequenza di confronti

Se si hanno più di due alternative, si usa **sequenza di confronti**

```
if (voto >= 8)
    System.out.println("Compito Eccellente");
else if (voto >= 6)
    System.out.println("Compito sufficiente");
else if (voto >= 4)
    System.out.println("Compito quasi sufficiente");
else if (voto >= 2)
    System.out.println("Compito insufficiente");
else if (voto >= 0)
    System.out.println("Compito totalmente ↩
        insufficiente");
else
    System.out.println("Numeri negativi non validi");
```

```
if (voto >= 0)
    System.out.println("Totalmente insufficiente");
else if (voto >= 2)
    System.out.println("Compito insufficiente");
else if (voto >= 4)
    System.out.println("Compito quasi insufficiente");
else if (voto >= 6)
    System.out.println("Compito sufficiente");
else
    System.out.println("Compito eccellente");
```

```
if (voto >= 0)
    System.out.println("Totalmente insufficiente");
else if (voto >= 2)
    System.out.println("Compito insufficiente");
else if (voto >= 4)
    System.out.println("Compito quasi insufficiente");
else if (voto >= 6)
    System.out.println("Compito sufficiente");
else
    System.out.println("Compito eccellente");
```

- Il codice seguente non funziona, perché stampa sempre **Totalmente insufficiente** per qualsiasi valore di voto.
- Se si fanno confronti di tipo **maggiore di** si devono scrivere prima i valori più alti e viceversa.

```
if (voto >= 8)
    System.out.println("Compito Eccellente");
if (voto >= 6)
    System.out.println("Compito sufficiente");
if (voto >= 4)
    System.out.println("Compito quasi sufficiente");
if (voto >= 2)
    System.out.println("Compito insufficiente");
if (voto >= 0)
    System.out.println("Compito totalmente ↩
       insufficiente");
```

```
if (voto >= 8)
    System.out.println("Compito Eccellente");
if (voto >= 6)
    System.out.println("Compito sufficiente");
if (voto >= 4)
    System.out.println("Compito quasi sufficiente");
if (voto >= 2)
    System.out.println("Compito insufficiente");
if (voto >= 0)
    System.out.println("Compito totalmente ↩
        insufficiente");
```

Se non si rendono **mutuamente esclusive** le alternative, usando le clausole **else**, non funziona

- se voto vale "3", stampa **sia** "Compito insufficiente" che **sia** "Compito totalmente insufficiente"

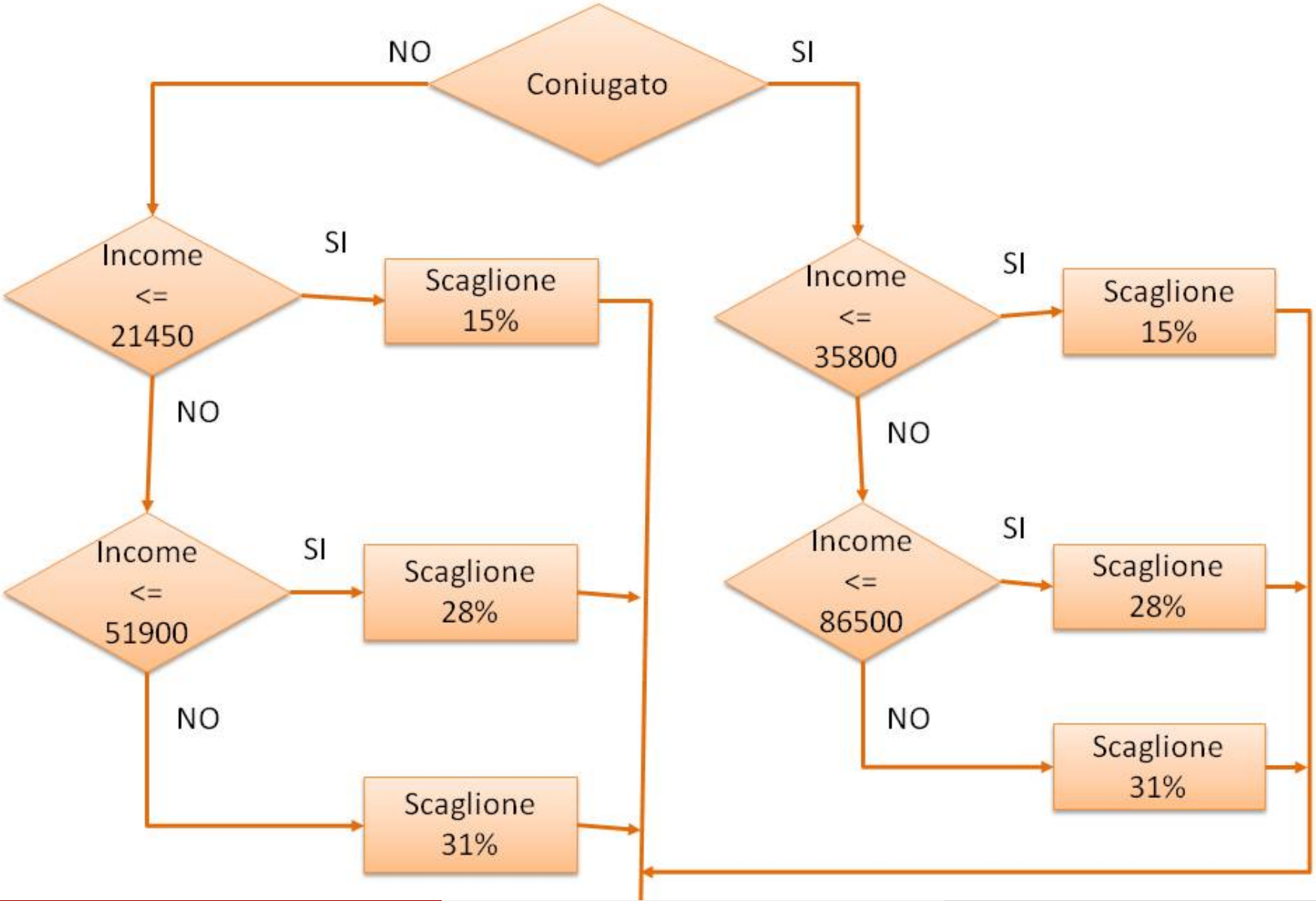
Diramazioni annidate

Se il vostro stato civile è non coniugato		Se il vostro stato civile coniugato	
Scaglione fiscale	**Aliquota**	**Scaglione fiscale**	**Aliquota**
\$0 . . . \$21 450	15%	\$0 . . . \$35 800	15%
\$21 450 fino a \$51 900	28%	\$38 500 . . . 86 500	28%
Superiore a \$51 900	31%	Superiore \$86 500	31%

Ci sono **due livelli** nel processo decisionale

- **Prima**, dobbiamo scegliere lo stato civile
- **Poi**, per ciascuno stato civile, dobbiamo scegliere lo scaglione di reddito.

Esercitazione



Nell'esempio precedente abbiamo notato che, se ben indentato il codice, la clausola else si riferisce al primo enunciato **if** La regola sintattica è che **una clausula else appartiene sempre all'enunciato if più vicino**

```
double cost = 5; // prezzo per USA
if (country.equals("USA"))
    if (state.equals("HI"))
        cost = 10; // Hawaii piu' costoso
else
   cost = 20; // estero ancora piu' costoso
```

L'esempio precedente svolge la funzione seguente con la giusta indentazione

```
double cost = 5; // prezzo per estero
if (country.equals("USA"))
    if (state.equals("HI"))
        cost = 10; // Hawaii piu' costoso
    else
        cost = 20; // USA ancora piu' costoso
```

Il risultato è che gli stati esteri ottengono il prezzo più basso, e gli USA continentali il più alto! Il **contrario** di ciò che si voleva...

Per ottenere il risultato voluto, bisogna "nascondere" il secondo enunciato if all'interno di un blocco di enunciati, inserendo una coppia di parentesi graffe.

- per evitare problemi con **else** sospeso, è meglio **racchiudere sempre** il corpo di un enunciato if tra parentesi graffe, anche quando sono inutili.

```
double cost = 5; // prezzo per USA
if (country.equals("USA")){
    if (state.equals("HI"))
    cost = 10; // Hawaii piu' costoso
}else
    cost = 20; // estero ancora piu' costoso
```

Espressioni booleane

Ogni espressione in Java ha un **valore**

- $x + 10$ espressione aritmetica, valore numerico;
- $x < 10$ espressione relazionale, valore booleano;

Una espressione relazionale può avere solo due valori **true** o **false**

- Il tipo di dati **boolean**, come tutti gli altri tipi di dati, consente la definizione di variabili
- A volte è comodo utilizzare **variabili booleane** per memorizzare valori di passaggi intermedi in cui è opportuno scomporre verifiche troppo complesse
- Altre volte l'uso di una variabile booleana rende più leggibile il codice

Metodi predicativi

Così vengono chiamati i metodi che restituiscono valori di tipo **booleano**.

- Solitamente **verificano** una condizione sullo stato di oggetto
- Solitamente iniziano con **is** oppure **has**

Metodi predicativi possono essere usati come condizioni di enunciati **if**

Gli operatori booleani o logici

Gli operatori booleani o logici servono a svolgere **operazioni su valori booleani**

if($x > 10$ **&&** $x < 20$**)**

- L'operatore **&&** (and) combina due o più condizioni in una sola, che risulta vera **se e solo se sono tutte vere**.
- L'operatore || (or) combina due o più condizioni in una sola, se risulta vera **se e solo se almeno una è vera**
- L'operatore **!** (not) **inverte** il valore di una espressione booleana

Più operatori booleani possono essere usati in un'unica espressione

$$\textbf{if}((x > 10 \ \textbf{\&\&} \ x > 20) \ \| \ x > 30)$$

La valutazione di un'espressione con operatori booleani viene effettuata con una strategia detta cortocircuito (o **valutazione pigra**)

AND

A	B	A&&B
true	true	true
true	false	false
false	qualsiasi	false

OR

A	B	A || B
true	qualsiasi	true
false	true	true
false	false	false

NOT

A	!A
true	false
false	true

- In un'espressione booleana con più operatori, la valutazione viene fatta da sinistra a destra, dando la precedenza all'operatore **not**, poi all'operatore **and**, infine all'operatore **or**
- L'ordine di valutazione può comunque essere alterato dalle parentesi tonde

```
if (!(x < 0 || x > 10)){
    // esegue se x e' compreso tra 0 e 10,
    // estremi inclusi
}
```

```
if (!x < 0 || x > 10)
  // esegue se x e' maggiore o uguale a 0
```

ESERCITAZIONE

Scrivere un programma che segnala all'utente se il numero intero positivo che ha introdotto corrisponde ad un anno bisestile oppure no.

Suggerimento: un anno bisestile è divisibile per 4. Fanno eccezione gli anni divisibili per 100, che non sono bisestili, e gli anni divisibili per 400, che invece sono bisestili: tali eccezioni esistono però solo dopo l'adozione del calendario gregoriano, che avvenne nel 1582.

Si scriva la classe Triangolo, che descrive un triangolo

```
public class Triangolo{
    /*
       Costruttore della classe Triangolo
    */
    public Triangolo (double la, double lb, double lc){
//... completare
    }

    /*
       restituisce informazioni sul triangolo. le ↩
          informazioni sono relative
       ai lati:    equilatero,  isoscele,    scaleno
       Esempio: per il triangolo di lati 3, 4, 5 ↩
          restituisce la stringa
       "scaleno rettangolo".
    */
    public String info(){
```