Costrutti di scelta e di iterazione

Unità didattica 2
Armando Tacchella
Fondamenti di Informatica

Introduzione ai costrutti di scelta

- I programmi costruiti da semplici sequenze di istruzioni non sono in grado di modificare il loro comportamento a seconda delle scelte effettuate dall'utente
- Java supporta la possibilità di modificare il comportamento del programma durante la sua esecuzione mediante i costrutti di scelta:
  - Costrutto if-then-else (se-allora-altrimenti)
  - Costrutto switch (scelta multipla)
- Accanto ai costrutti di scelta introduciamo:
  - Le espressioni logiche (valutano a vero/falso)
  - Il tipo boolean (valori logici vero/falso)

Costrutto di scelta condizionale if-then-else

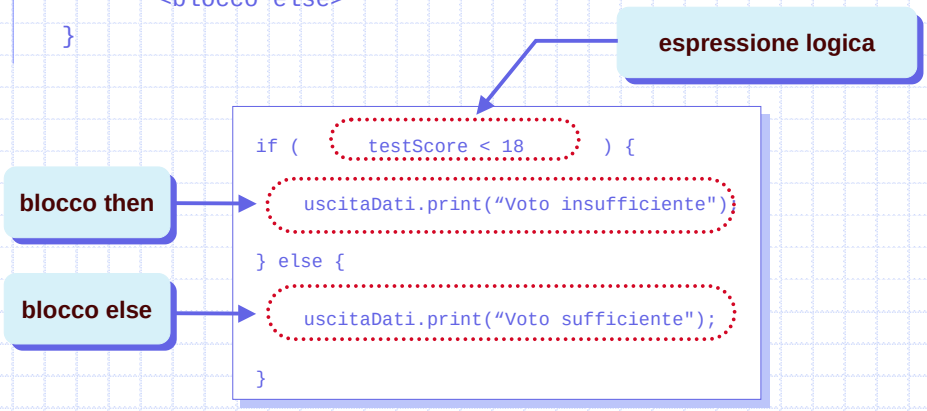
```
// uscitaDati e inDati sono stati dichiarati e creati
// testScore è dichiarato come int

testScore = inDati.getInteger("Inserire il punteggio:");

if (testScore < 18) {

    uscitaDati.print("Voto insufficiente");

} else {

    uscitaDati.print("Voto sufficiente");

}
```

Questa istruzione viene eseguita se testScore è inferiore a 18.

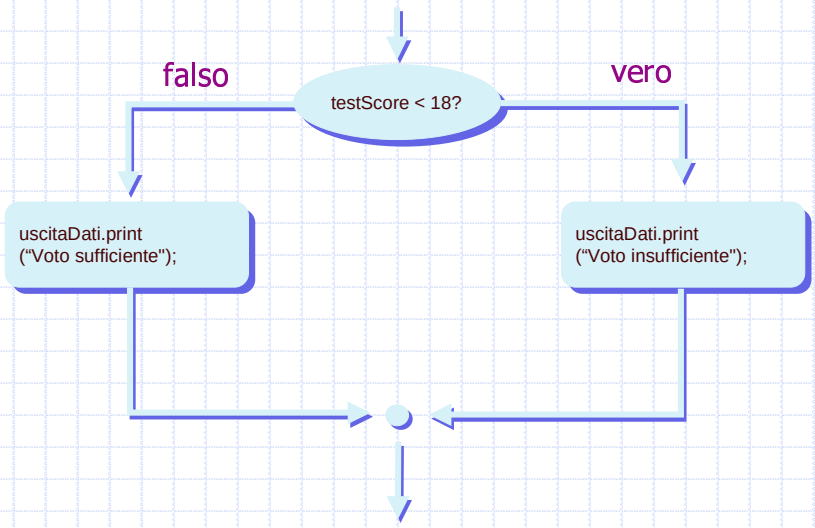
Questa istruzione viene eseguita se testScore è maggiore o uguale a 18.

Sintassi del costrutto if-then-else

```
if ( <espressione logica> ) {
        <blocco then>
} else {
        <blocco else>
}
```



Flusso di controllo if-then-else



Espressioni logiche e operatori relazionali

- Un'espressione della forma

  `<expr> <opr> <expr>`

  dove `<expr>` è un'espressione aritmetica e `<opr>` è un operatore relazionale, è un'espressione logica
- La tabella a fianco riporta gli operatori relazionali in Java
- Alcuni esempi di

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

```
testScore < 80
(testScore * 2) >= 350
30 < (w / (h * h))
(x + y) != (2 * (a + b))
(2 * Math.PI * radius) <= 359.99
```

Tipo booleano

- Le espressioni logiche possono avere solo due valori: vero o falso
- Le espressioni logiche hanno tipo boolean (primitivo)
- `true` (vero) e `false` (falso) sono costanti predefinite di tipo booleano
- Operazioni con le variabili booleane:

```
boolean x;          // x è dichiarata boolean
boolean y = false; // y è inizializzata a falso
x = w < 18;         // x è true se w<18
```

- Ogni variabile booleana è un'espressione logica

Espressioni logiche e operatori logici

- Un'espressione della forma
  `<lexpr> <opb> <lepxr>`
  dove `<lexpr>` è un'espressione logica e `<opb>` è un operatore logico, è un'espressione logica
- Gli operatori logici in Java sono tre: `&&` (congiunzione), `||` (disgiunzione), e `!` (negazione)

a	b	a && b	a || b	!a
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Valutazione delle espressioni logiche

- Si consideri la seguente espressione logica:

```
(x > y) || (x > z)
```

- L'espressione è valutata da sinistra a destra. Se x > y è vera, allora non è necessario valutare x > z perché l'espressione sarà vera indipendentemente da x > z.
- Java interrompe la valutazione delle espressioni logiche non appena il risultato è univocamente definito (valutazione short-circuit)
- Cosa accadrebbe se la seguente espressione non fosse valutata in maniera short-circuit?

```
(z == 0) || ((x / z) > 20)
```

Precedenza ed associatività

Gruppo	Operatore	Precedenza	Associatività
sottoespressione	( )	9	sinistra->destra
operatori unari	- !	8	destra->sinistra
operatori moltiplicativi	* / %	7	sinistra->destra
operatori addittivi	+ -	6	sinistra->destra
operatori relazionali	< <= > >=	5	sinistra->destra
operatori di uguaglianza	== !=	4	sinistra->destra
congiunzione	&&	3	sinistra->destra
disgiunzione	||	2	sinistra->destra
assegnamento	=	1	destra->sinistra

Blocchi then ed else con più istruzioni

- Il blocco then e il blocco else sono obbligatoriamente racchiusi tra graffe quando constano di più istruzioni.

```
if (testScore < 18)
{
    uscitaDati.print("Voto insufficiente");
    uscitaDati.print("Prova a studiare!");
}
else
{
    uscitaDati.print("Voto sufficiente");
    uscitaDati.print("Contunua così!");
}
```

blocco then

blocco else

Un po' di stile...

```
if ( <espressione logica> ) {
    …
} else {
    …
}
```

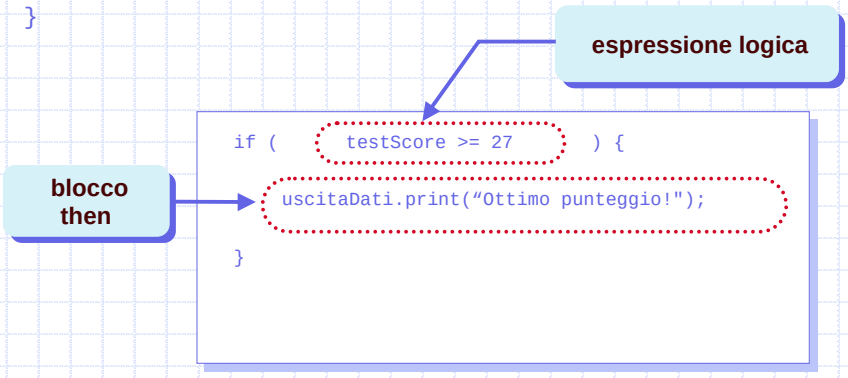
Stile 1

```
if ( <espressione logica> ) {
    …
}
else {
    …
}
```

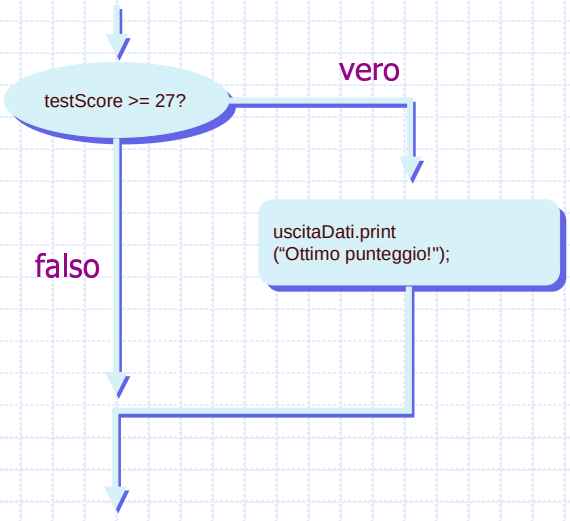
Stile 2

Costrutto di scelta condizionale if-then

```
if ( <espressione logica> ) {
        <blocco then>
}
```



Flusso di controllo if-then



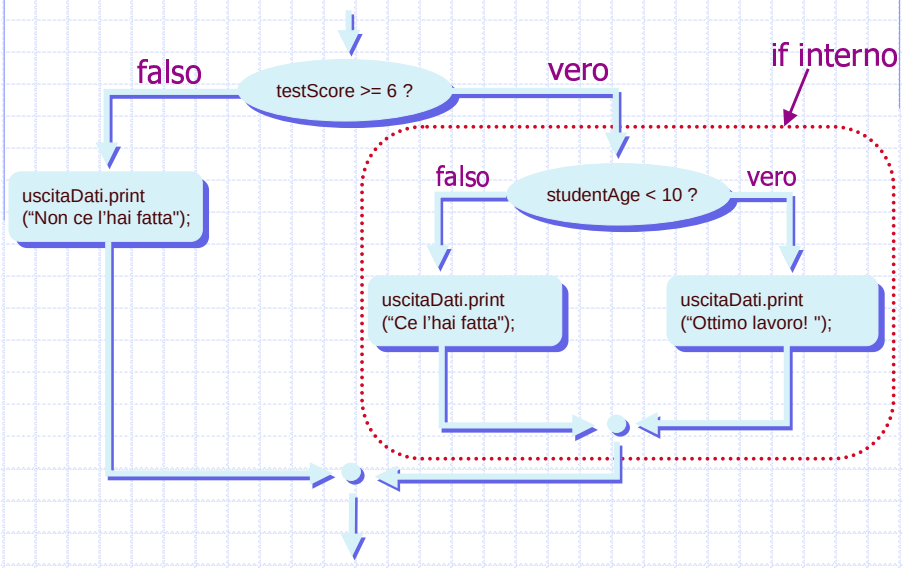
Costrutti if annidati

- Il blocco then e il blocco else possono contenere qualsiasi istruzione valida, inclusi altri costrutti if. Questi ultimi si dicono costrutti if annidati (nested).

costrutto if interno

```
if (testScore >= 6) {
    if (studentAge < 10) {
        uscitaDati.print("Ottimo lavoro!");
    } else {
        uscitaDati.print("Ce l'hai fatta"); //test score >= 6
    }                                       //and age >= 10
} else { // test score < 6
    uscitaDati.print("Non ce l'hai fatta");
}
```

Flusso di controllo di if annidati



Costrutti condizionali alla prova

- Supponiamo di avere due variabili intere `num1` e `num2`
- `negativeCount` conta il numero di valori negativi

```
int negativeCount;
if (num1 < 0) {
   if (num2 < 0) {
      negativeCount = 2;
   } else {
      negativeCount = 1;
   }
} else {
   if (num2 < 0) {
      negativeCount = 1;
   } else {
      negativeCount = 0;
   }
}
```

```
int negativeCount = 0;
if (num1 < 0) {
   negativeCount =
      negativeCount + 1;
}
if (num2 < 0) {
   negativeCount =
      negativeCount + 1;
}
```

Costrutto di scelta condizionale if – else if

- Supponiamo di dover prendere una decisione sulla base di una gamma di valori
- Nell'esempio a fianco, ad ogni punteggio numerico corrisponde un voto alfabetico

Punteggio	Voto
90 ≤ score	A
80 ≤ score < 90	B
70 ≤ score < 80	C
60 ≤ score < 70	D
score < 60	F

```
if (score >= 90) {
   uscitaDati.print("Il tuo voto è A");
} else if (score >= 80) {
    uscitaDati.print("Il tuo voto è B");
} else if (score >= 70) {
    uscitaDati.print("Il tuo voto è C");
} else if (score >= 60) {
    uscitaDati.print("Il tuo voto è D");
} else {
    uscitaDati.print("Il tuo voto è F");
}
```

Nota: associazione tra if ed else

A e B sono differenti?

A

```
if (x < y)
   if (x < z)
    uscitaDati.print("Hello!");
else
    uscitaDati.print("Ciao!");
```

B

```
if (x < y)
   if (x < z)
       uscitaDati.print("Hello!");
   else
      uscitaDati.print("Ciao!");
```

Sia A che B significano

```
if (x < y) {
   if (x < z) {
      uscitaDati.print("Hello");
   } else {
      uscitaDati.print("Ciao!");
   }
}
```

Istruzione di scelta condizionale switch

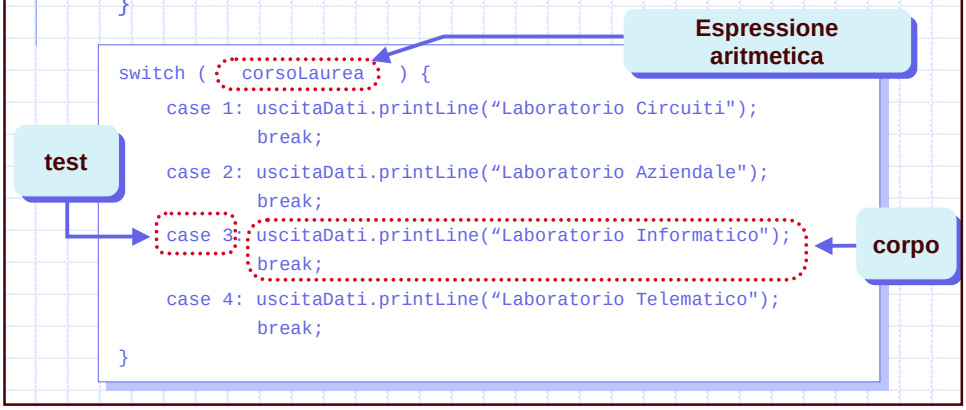
```
int corsoLaurea;
corsoLaurea =
    inDati.getInteger("Corso 1=ELE, 2=GEST, 3=INFO, 4=TLC:" );

switch (corsoLaurea) {

    case 1: uscitaDati.printLine("Laboratorio Circuiti");
            break;

    case 2: uscitaDati.printLine("Laboratorio Aziendale");
            break;

    case 3: uscitaDati.printLine("Laboratorio Informatico");
            break;

    case 4: uscitaDati.printLine("Laboratorio Telematico");
            break;
}
```

Viene eseguita se corsoLaurea è uguale a 1.

Viene eseguita se corsoLaurea è uguale a 4.

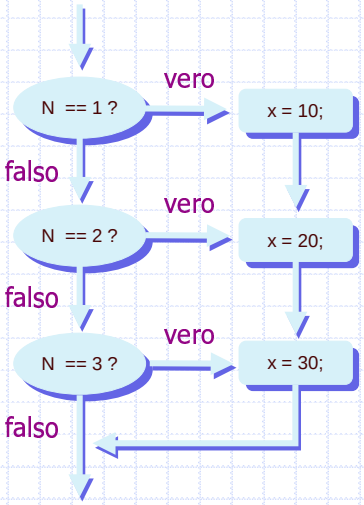
Sintassi del costrutto switch

```
switch ( <espressione aritmetica> ) {
        <test 1> : <corpo 1>
        …
        <test n> : <corpo n>
}
```



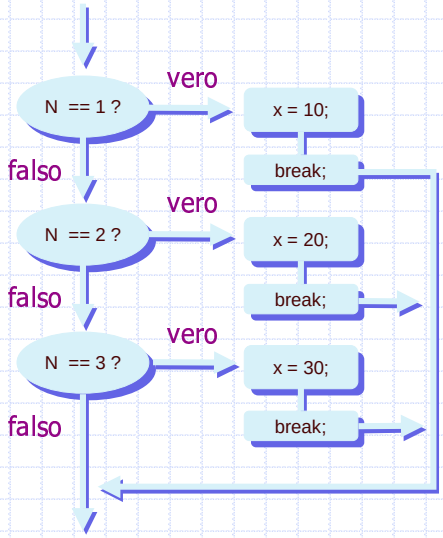
Flusso di controllo di switch senza break

```
switch ( N ) {
   case 1: x = 10;
   case 2: x = 20;
   case 3: x = 30;
}
```



Flusso di controllo di switch con istruzioni break

```
switch ( N ) {
   case 1: x = 10;
           break;
   case 2: x = 20;
           break;
   case 3: x = 30;
           break;
}
```



Blocco default per l'istruzione switch

```
switch (esperienza) {

    case 10:
    case  9:
    case  8: uscitaDati.print("Maestro");
             break;

    case  7:
    case  6: uscitaDati.print("Anziano");
             break;

    case  5:
    case  4: uscitaDati.print("Apprendista");
             break;

    default: uscitaDati.print("Errore: dato non valido");
             break;
}
```

Introduzione ai costrutti di iterazione

- I programmi costruiti da:
  - sequenze di istruzioni
  - scelte condizionali

  non sono in grado di ripetere una certa funzione a meno che il numero di iterazioni non sia fissato a priori
- Java supporta la possibilità di ripetere l'esecuzione di parti di un programma mediante i costrutti di iterazione:
  - Costrutti while-do e do-while (iterazione condizionale)
  - Costrutto for (iterazione semplice)

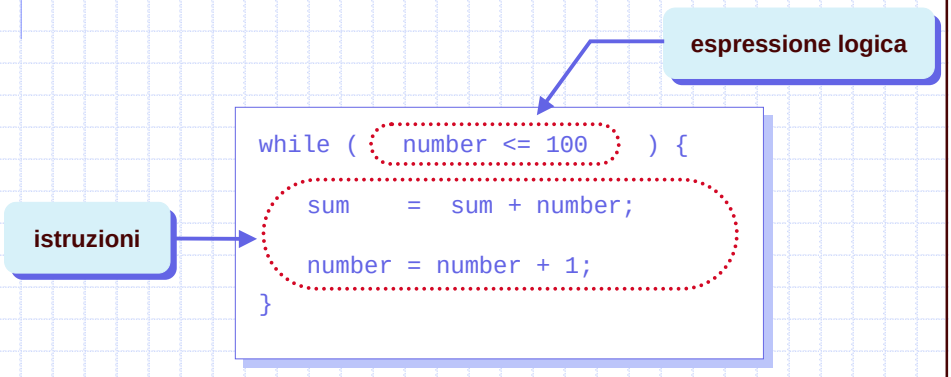
Costrutto di iterazione condizionale while

```
int sum = 0, number = 1;

while ( number <= 100 ) {

   sum    =  sum + number;

   number = number + 1;

}
```

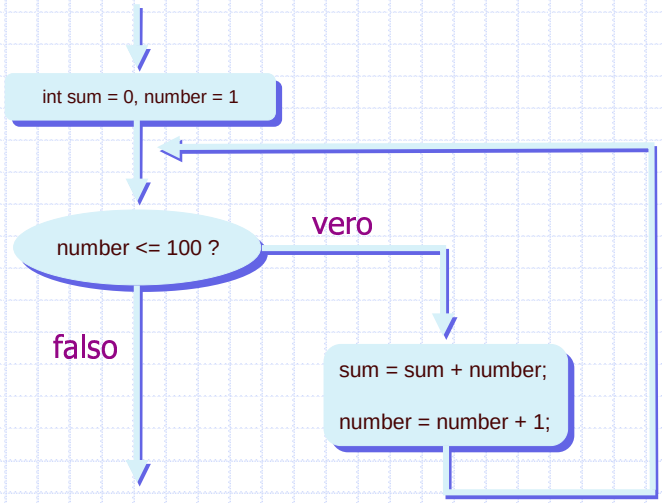
Queste istruzioni vengono eseguite fintanto che number è minore o uguale a 100.

Sintassi del costrutto while

```
while ( <espressione logica> ) {
        <istruzioni>
}
```



Flusso di controllo while



Utilizzo di while: ripetizione di calcoli

1

```
int sum = 0, number = 1;

while ( sum <= 1000000 ) {
     sum    =  sum + number;
     number = number + 1;
}
```

Continua ad aggiungere i numeri 1, 2, 3, … sino a che la somma diventa maggiore di 1.000.000.

2

```
int product =  1, number = 1,
    count   = 20, lastNumber;

lastNumber = 2 * count - 1;

while (number <= lastNumber) {
     product = product * number;
     number  = number + 2;
}
```

Calcola il prodotto dei primi 20 interi dispari.

Utilizzo di while: controllo dei valori in ingresso

- Un esempio realistico di utilizzo del costrutto while per filtrare i valori in ingresso non validi.
- Accetta età tra 0 e 130, estremi esclusi.

Questa istruzione deve essere ripetuta.

```
age = inDati.getInteger("Età (tra 0 e 130):");

while (age < 0 || age > 130) {
    uscitaDati.printLine("Età non valida. Si prega riprovare.");

    age = inDati.getInteger( "Età (tra 0 e 130):" );

}
```

Possibili errori con costrutti while - 1

1
```
int product = 0;

while ( product < 500000 ) {
    product = product * 5;
}
```

2
```
int count = 0;

while ( count != 11 ) {
    count = count + 2;
}
```

Cicli Infiniti
Entrambe i cicli non termineranno perchè le espressioni logiche non diventeranno mai false.

Possibili errori con il costrutto while - 2

- Obiettivo: eseguire il corpo del ciclo 10 volte.

1
```
count = 1;
while ( count < 10 )
{
    . . .
    count = count+1;
}
```
✗

2
```
count = 1;
while ( count <= 10 )
{
    . . .
    count = count+1;
}
```
✓

3
```
count = 0;
while ( count <= 10 )
{
    . . .
    count = count+1;
}
```
✗

4
```
count = 0;
while ( count < 10 )
{
    . . .
    count = count+1;
}
```
✓

1 e 3 contengono l'errore off-by-one (fuori di uno).

Controlli sui costrutti iterativi

1. Attenzione all'errore off-by-one.
2. Assicurarsi che il ciclo contenga un'istruzione che consentirà al ciclo stesso di terminare.
3. Assicurarsi che il ciclo sia ripetuto il numero corretto di volte. In particolare, volendo eseguire il corpo di un ciclo N volte è necessario:
   1. inizializzare una variabile `contatore` a `0` e utilizzare il test `(contatore < N)`, oppure
   2. inizializzare una variabile `contatore` a `1` e utilizzare il test `(contatore <= N)`.
4. Assicurarsi che le istruzioni necessarie nel ciclo siano contenute entro le parentesi graffe.

Abbreviazione degli operatori aritmetici

```
sum = sum + number;
```

equivale a

```
sum += number;
```

Operatore	Utilizzo	Significato
+=	a += b;	a = a + b;
-=	a -= b;	a = a – b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Operatori di pre- e post-incremento

Operatore	Utilizzo	Denota	Significato
++	a++	a	a = a + 1
	++a	a+1	
--	a--	a	a = a - 1
	--a	a-1	

Costrutto di iterazione condizionale do-while

```
int sum = 0, number = 1;

do {
   sum += number;
   number++;
} while ( sum <= 1000000 );
```

Queste istruzioni sono eseguite fintanto che la somma è minore uguale a 1.000.000.

Sintassi del costrutto do-while

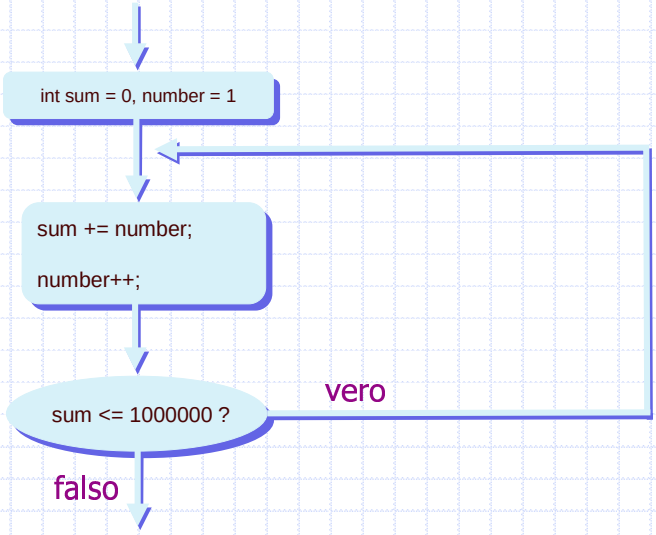
```
do {
        <istruzioni>
} while ( <espressione logica> );
```

```
do   {
    sum += number;
    number++;
}   while (   sum <= 1000000   );
```

istruzioni

espressione logica

Flusso di controllo do-while



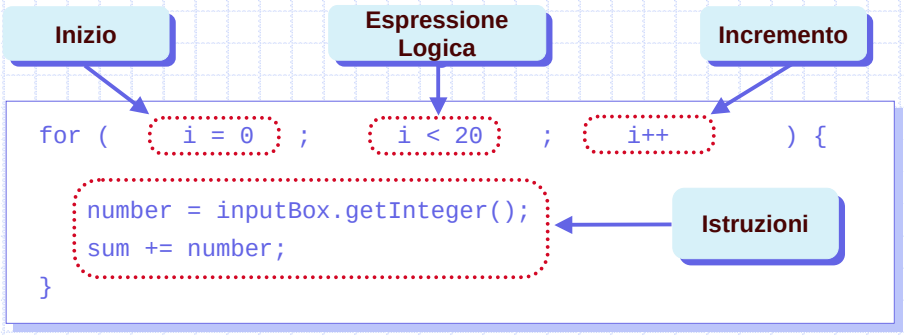
Il costrutto di iterazione semplice for

```
int i, sum = 0, number;

for (i = 0; i < 20; i++) {
    number = inDati.getInteger();
    sum += number;
}
```

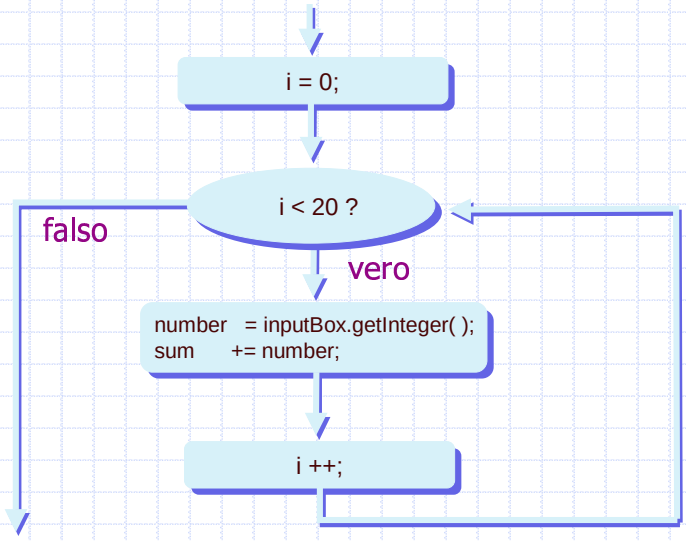
Queste istruzioni sono eseguite 20 volte (i = 0, 1, 2, … , 19).

Sintassi del costrutto for

```
for ( <inizio>; <espressione logica>; <incremento>  ){
      <istruzioni>
}
```



Flusso di controllo del ciclo for



Costrutto for: alcuni esempi di impostazione

1
```
for (int i = 0; i < 100; i += 5)
```
i = 0, 5, 10, … , 95

2
```
for (int j = 2; j < 40; j *= 2)
```
j = 2, 4, 8, 16, 32

3
```
for (int k = 100; k > 0; k--) )
```
k = 100, 99, 98, 97, ..., 1

Costrutti di iterazione annidati

- I cicli annidati sono un costrutto comunemente utilizzato in programmazione.
- Si supponga di voler generare la seguente tabella...

Carpet Price Table

	5	10	15	20	25
11	1045	2090	3135	4180	5225
12	1140	2280	3420	4560	5700
13	1235	2470	3705	4940	6175
14	1330	2660	3990	5320	6650
15	1425	2850	4275	5700	7125
16	1520	3040	4560	6080	7600
17	1615	3230	4845	6460	8075
18	1710	3420	5130	6840	8550
19	1805	3610	5415	7220	9025
20	1900	3800	5700	7600	9500

Generazione di una tabella

```
int          price;
MainWindow   finestra   = new MainWindow();
OutputBox    uscitaDati = new OutputBox(finestra);

finestra.show();
uscitaDati.setTitle("Carpet Price Table");
uscitaDati.show();

for (int width = 11; width <= 20; width++) {

    for (int length = 5; length <= 25; length += 5) {
            price = width * length * 19; //19 euro per mq
            uscitaDati.print(" ");
            uscitaDati.print(price);
    }

    uscitaDati.skipLine(1); // un spazio al termine di ogni riga
}
```

ESTERNO

INTERNO

Esercitazione – Soluzione di equazioni (1)

- Scrivere un'applicazione Java per il calcolo delle soluzioni di un'equazione di secondo grado $ax^2 + bx + c = 0$ non degenere ($a \neq 0$ e $b \neq 0$)
- L'applicazione deve richiedere i tre coefficienti e calcolare le soluzioni nel caso in cui siano reali
- Lo schema del programma è:
  - Richiedere i coefficienti a,b,c
  - Verificare che a e b siano diversi da 0 e, in caso contrario, non proseguire nell'esecuzione
  - Calcolare il discriminante
  - Sulla base del valore del discriminante calcolare le soluzioni se reali, o se reali e coincidenti.
  - Visualizzare le soluzioni

Esercitazione – Soluzione di equazioni (2)

- Modificare il programma (1) per fare in modo che non prosegua fintanto che l'utente non ha inserito valori corretti per i coefficienti a e b
- Modificare il programma (1) per chiedere all'utente se, una volta terminati i calcoli, desidera calcolare le soluzioni di un'altra equazione
- Modificare il programma (1) per chiedere in anticipo quante sono le equazioni di cui si desidera calcolare le soluzioni
- Scrivere un programma che unisca le estensioni proposte al primo e al terzo punto.

Esercitazione – Soluzione di problemi con cicli

- Dato un numero N, chiedere N dati numerici all'utente e calcolare somma e media.
- Dato un numero N, calcolare il fattoriale di N. Il fattoriale, denotato N!, è definito come:
  - N! = 0 se N = 0 - N! = 1 se N = 1
  - N! = N(N-1)! se N > 1
- Dato un numero N, visualizzare un triangolo rettangolo con base N e altezza N.
- Come sopra, ma per un triangolo isoscele

Supponendo N=3, vogliamo visualizzare:

Unità didattica 2 – Argomenti svolti

- Scelte condizionali utilizzando if e switch
- Espressioni logiche, operatori relazionali e logici
- Scelte condizionali annidate
- Iterazioni condizionali utilizzando while-do e do-while
- Iterazioni con l'istruzione for
- Costrutti iterativi annidati