



Capitolo 3

Selezione e iterazione

Sommario:

L'istruzione `if-else`

Esercizio `ConfrontoFrazioni`

`if-else` innestati

Il tipo primitivo `boolean`

Confronto fra riferimenti

Operatori booleani

Il ciclo `do...while`

Il ciclo `while`

Il ciclo `for`

Le istruzioni `break` e `continue`

L'istruzione if-else

Sintassi

```
if (condizione)  
    istruzione1  
else  
    istruzione2
```

Sintassi

```
if (condizione)  
    istruzione1  
else  
    istruzione2
```

► *condizione*

È una qualunque espressione di tipo **boolean** scritta *obbligatoriamente* tra parentesi tonde

Sintassi

```
if (condizione)  
    istruzione1  
else  
    istruzione2
```

- ▶ *condizione*

È una qualunque espressione di tipo **boolean** scritta *obbligatoriamente* tra parentesi tonde

- ▶ *istruzione1*, *istruzione2*

sono *istruzioni singole* oppure *blocchi di istruzioni*, cioè sequenze di istruzioni racchiuse tra parentesi graffe

L'istruzione if-else: esecuzione

Sintassi

```
if (condizione)  
    istruzione1  
else  
    istruzione2
```

- (1) Viene valutata *condizione* (il valore è **true** o **false**)

L'istruzione if-else: esecuzione

Sintassi

```
if (condizione)  
    istruzione1  
else  
    istruzione2
```

- (1) Viene valutata *condizione* (il valore è **true** o **false**)
 - se la condizione è **vera**, viene eseguita *istruzione1*

Sintassi

```
if (condizione)  
    istruzione1  
else  
    istruzione2
```

(1) Viene valutata *condizione* (il valore è **true** o **false**)

- se la condizione è **vera**, viene eseguita *istruzione1*
- se la condizione è **falsa**, viene eseguita *istruzione2*

Sintassi

```
if (condizione)  
    istruzione1  
else  
    istruzione2
```

- (1) Viene valutata *condizione* (il valore è **true** o **false**)
 - se la condizione è **vera**, viene eseguita *istruzione1*
 - se la condizione è **falsa**, viene eseguita *istruzione2*
- (2) L'esecuzione riprende dalla prima istruzione che segue l'istruzione *if-else*

L'istruzione if: esecuzione

Sintassi

```
if (condizione)  
    istruzione
```

- Viene valutata la condizione

Sintassi

```
if (condizione)  
    istruzione
```

- ▶ Viene valutata la condizione
 - se la condizione è **vera**, viene eseguita *istruzione*

Sintassi

```
if (condizione)  
    istruzione
```

- ▶ Viene valutata la condizione
 - se la condizione è **vera**, viene eseguita *istruzione*
 - se la condizione è **falsa**, l'esecuzione riprende dalla prima istruzione che segue l'istruzione **if**.

ConfrontoFrazioni

```
int num, den;  
  
//lettura e costruzione della prima frazione  
num = in.readInt("Numeratore prima frazione> ");  
den = in.readInt("Denominatore prima frazione> ");  
Frazione f1 = new Frazione(num, den);
```

ConfrontoFrazioni

```
int num, den;

//lettura e costruzione della prima frazione
num = in.readInt("Numeratore prima frazione> ");
den = in.readInt("Denominatore prima frazione> ");
Frazione f1 = new Frazione(num, den);

//lettura e costruzione della seconda frazione
num = in.readInt("Numeratore seconda frazione> ");
den = in.readInt("Denominatore seconda frazione> ");
Frazione f2 = new Frazione(num, den);
```

ConfrontoFrazioni

```
int num, den;

//lettura e costruzione della prima frazione
num = in.readInt("Numeratore prima frazione> ");
den = in.readInt("Denominatore prima frazione> ");
Frazione f1 = new Frazione(num, den);

//lettura e costruzione della seconda frazione
num = in.readInt("Numeratore seconda frazione> ");
den = in.readInt("Denominatore seconda frazione> ");
Frazione f2 = new Frazione(num, den);

//confronto ...
if (f1.equals(f2))
    out.println("Le due frazioni sono uguali");
else
    out.println("Le due frazioni sono diverse");
```

- Dato che **if-else** è un'istruzione, può comparire nel corpo di un'istruzione **if-else...**

```
int x, y, z;  
...  
if (x == 1)  
    if (y == 1)  
        z = x + y;  
    else  
        z = x * y;  
else  
    z = x - y;
```



```
if (x == 1) if (y == 1) z = x + y; else z = x - y;
```

```
if (x == 1) if (y == 1) z = x + y; else z = x - y;
```

- Per il compilatore è equivalente a:

```
if (x == 1)
    if (y == 1)
        z = x + y;
    else
        z = x - y;
```

```
if (x == 1)
    if (y == 1)
        z = x + y;
else
    z = x - y;
```

```
if (x == 1) if (y == 1) z = x + y; else z = x - y;
```

- Per il compilatore è equivalente a:

```
if (x == 1)
    if (y == 1)
        z = x + y;
    else
        z = x - y;
```

```
if (x == 1)
    if (y == 1)
        z = x + y;
else
    z = x - y;
```

- In una sequenza di **if innestati**, un **else** è associato
 - al primo **if** che lo precede, ...
 - per il quale non sia stato ancora individuato un **else**

```
if (x == 1)
  if (y == 1)
    z = x + y;
  else
    z = x - y;
```

```
if (x == 1)
  if (y == 1)
    z = x + y;
  else
    z = x - y;
```

- Per associare l'unico **else** al primo **if** dobbiamo utilizzare le parentesi graffe:

```
if (x == 1) {
  if (y == 1)
    z = x + y;
} else
  z = x - y;
```

- ▶ Due valori possibili, denotati dai letterali `false` e `true`

- ▶ Due valori possibili, denotati dai letterali `false` e `true`
- ▶ **Condizioni**
Le *espressioni booleane*, cioè le espressioni che restituiscono un valore di tipo `boolean`

- ▶ Due valori possibili, denotati dai letterali `false` e `true`

- ▶ **Condizioni**

Le *espressioni booleane*, cioè le espressioni che restituiscono un valore di tipo `boolean`

- ▶ **Condizioni semplici**

Confronti fra espressioni di tipo primitivo mediante un *operatore relazionale*:

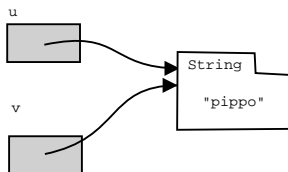
- > maggiore di
- <= minore o uguale a
- >= maggiore o uguale a
- == uguale a
- < minore di
- != diverso da

Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = u;
```

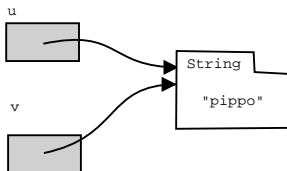
Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = u;
```



Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = u;
```

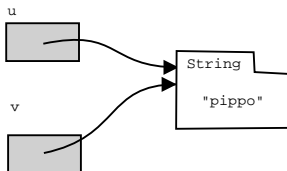


`u == v`

- ▶ viene valutata **true**
- ▶ `u` e `v` fanno riferimento allo stesso oggetto

Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = u;
```



`u == v`

- ▶ viene valutata **true**
- ▶ `u` e `v` fanno riferimento allo stesso oggetto

`u.equals(v)`

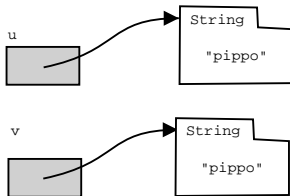
- ▶ viene valutata **true**
- ▶ un oggetto è uguale a se stesso

Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = new String("pippo");
```

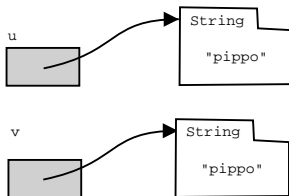
Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = new String("pippo");
```



Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = new String("pippo");
```

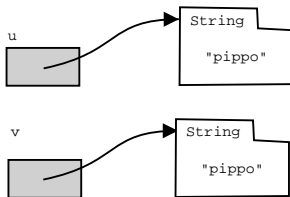


```
u == v
```

- ▶ viene valutata **false**
- ▶ `u` e `v` fanno riferimento a oggetti distinti

Confronto fra riferimenti: == e equals

```
String u, v;  
u = new String("pippo");  
v = new String("pippo");
```



`u == v`

- ▶ viene valutata **false**
- ▶ u e v fanno riferimento a oggetti distinti

`u.equals(v)`

- ▶ viene valutata **true**
- ▶ i due oggetti rappresentano la medesima stringa

- Un tipo è caratterizzato dai suoi valori e dalle operazioni che si possono compiere su di essi.

- ▶ Un tipo è caratterizzato dai suoi valori e dalle operazioni che si possono compiere su di essi.

Il tipo `boolean` dispone di:

- ▶ due operatori binari ($\text{boolean} \times \text{boolean} \rightarrow \text{boolean}$)

`&&` and (*congiunzione*)

`||` or (*disgiunzione*)

- ▶ Un tipo è caratterizzato dai suoi valori e dalle operazioni che si possono compiere su di essi.

Il tipo `boolean` dispone di:

- ▶ due operatori binari ($\text{boolean} \times \text{boolean} \rightarrow \text{boolean}$)

`&&` and (*congiunzione*)

`||` or (*disgiunzione*)

- ▶ un operatore unario ($\text{boolean} \rightarrow \text{boolean}$)

`!` not (*negazione*)

Tavole di verità

x	y	x && y
false	false	false
false	true	false
true	false	false
true	true	true

x	y	x y
false	false	false
false	true	true
true	false	true
true	true	true

x	!x
false	true
true	false

Precedenze degli operatori booleani

- L'operatore `!` ha la massima precedenza e `||` la minima

Precedenze degli operatori booleani

- L'operatore **!** ha la massima precedenza e **||** la minima

Esempio

`a && b || a && c` equivalente a `(a && b) || (a && c)`

Precedenze degli operatori booleani

- L'operatore **!** ha la massima precedenza e **||** la minima

Esempio

`a && b || a && c` equivalente a `(a && b) || (a && c)`

`!a && b || a && !c` equivalente a `((!a) && b) || (a && (!c))`

Esempio di tavola di verità

`! (a && b) || (a && c)`

a	b	c	!	(a && b)		(a && c)
false	false	false	true	false	true	false
false	false	true	true	false	true	false
false	true	false	true	false	true	false
false	true	true	true	false	true	false
true	false	false	true	false	true	false
true	false	true	true	false	true	true
true	true	false	false	true	false	false
true	true	true	false	true	true	true

`! (x && y)` equivalente a `!x || !y`

Leggi di De Morgan

$\neg (x \ \&\& \ y)$ equivalente a $\neg x \ || \ \neg y$

$\neg (x \ || \ y)$ equivalente a $\neg x \ \&\& \ \neg y$

$\neg (x \ \&\& \ y)$ equivalente a $\neg x \ || \ \neg y$

$\neg (x \ || \ y)$ equivalente a $\neg x \ \&\& \ \neg y$

Esercizio

Dimostrare le leggi di De Morgan costruendo e confrontando le tavole di verità delle espressioni coinvolte.

L'istruzione do...while

Sintassi

```
do  
    istruzione  
while (condizione)
```

Sintassi

```
do  
    istruzione  
while (condizione)
```

► *condizione*

È un'espressione di tipo `boolean` scritta *obbligatoriamente* tra parentesi tonde

L'istruzione do...while

Sintassi

```
do  
    istruzione  
while (condizione)
```

► *condizione*

È un'espressione di tipo **boolean** scritta *obbligatoriamente* tra parentesi tonde

► *istruzione*

È l'istruzione che dev'essere ripetuta: può essere un'istruzione singola oppure un blocco di istruzioni

L'istruzione do...while: esecuzione

```
do  
    istruzione  
while (condizione)
```

- (1) Viene eseguito il corpo del ciclo, cioè *istruzione*

L'istruzione do...while: esecuzione

```
do  
    istruzione  
while (condizione)
```

- (1) Viene eseguito il corpo del ciclo, cioè *istruzione*
- (2) Viene valutata l'espressione *condizione*

L'istruzione do...while: esecuzione

```
do  
    istruzione  
while (condizione)
```

- (1) Viene eseguito il corpo del ciclo, cioè *istruzione*
- (2) Viene valutata l'espressione *condizione*
 - se la condizione è **vera**, l'esecuzione prosegue dal **Punto (1)**

L'istruzione `do...while`: esecuzione

```
do  
    istruzione  
while (condizione)
```

- (1) Viene eseguito il corpo del ciclo, cioè *istruzione*
- (2) Viene valutata l'espressione *condizione*
 - se la condizione è **vera**, l'esecuzione prosegue dal **Punto (1)**
 - se la condizione è **falsa**, il ciclo termina e l'esecuzione riprende dalla prima istruzione che segue l'istruzione `do...while`.

L'istruzione do...while

```
do  
    istruzione  
while (condizione)
```

Osservazioni

- L'esecuzione del ciclo termina quando la condizione risulta **falsa**

L'istruzione do...while

```
do  
    istruzione  
while (condizione)
```

Osservazioni

- ▶ L'esecuzione del ciclo termina quando la condizione risulta **falsa**
- ▶ Il corpo del ciclo è sempre eseguito almeno una volta

```
...  
int num, den;  
Frazione somma = new Frazione(0);  
boolean continua;
```

```
...  
int num, den;  
Frazione somma = new Frazione(0);  
boolean continua;  
  
do {  
    num = in.readInt("Numeratore? ");  
    den = in.readInt("Denominatore? ");  
    somma = somma.piu(new Frazione(num, den));  
    continua = in.readSiNo("Vuoi inserire un'altra frazione (s/n)? ");  
} while (continua);
```

```
...  
int num, den;  
Frazione somma = new Frazione(0);  
boolean continua;  
  
do {  
    num = in.readInt("Numeratore? ");  
    den = in.readInt("Denominatore? ");  
    somma = somma.piu(new Frazione(num, den));  
    continua = in.readSiNo("Vuoi inserire un'altra frazione (s/n)? ");  
} while (continua);  
  
out.println("La somma e' " + somma.toString());  
...
```

L'istruzione while

Sintassi

```
while (condizione)  
    istruzione
```


Sintassi

```
while (condizione)  
    istruzione
```

► *condizione*

È un'espressione booleana scritta **obbligatoriamente** tra parentesi tonde

Sintassi

```
while (condizione)  
    istruzione
```

► *condizione*

È un'espressione booleana scritta **obbligatoriamente** tra parentesi tonde

► *istruzione*

È l'istruzione che dev'essere ripetuta; può essere un'istruzione singola oppure un blocco di istruzioni

L'istruzione while: esecuzione

```
while (condizione)  
    istruzione
```

(1) Viene valutata l'espressione *condizione*

L'istruzione while: esecuzione

```
while (condizione)  
    istruzione
```

(1) Viene valutata l'espressione *condizione*

- Se la condizione è **vera**:
 - viene eseguita *istruzione* (il corpo del ciclo)
 - l'esecuzione continua dal **Punto (1)**

L'istruzione while: esecuzione

```
while (condizione)  
    istruzione
```

(1) Viene valutata l'espressione *condizione*

- Se la condizione è **vera**:
 - viene eseguita *istruzione* (il corpo del ciclo)
 - l'esecuzione continua dal **Punto (1)**
- Se la condizione è **falsa**, il ciclo termina e l'esecuzione continua dalla prima istruzione che segue il ciclo **while**

L'istruzione while

```
while (condizione)  
    istruzione
```

Osservazioni

- L'esecuzione del ciclo termina quando *condizione* risulta *falsa*

L'istruzione while

```
while (condizione)  
    istruzione
```

Osservazioni

- ▶ L'esecuzione del ciclo termina quando *condizione* risulta **falsa**
- ▶ *istruzione* può essere eseguita anche zero volte

```
int num, den;  
Frazione somma = new Frazione(0);  
int contaFrazioni = 0;
```



```
int num, den;  
Frazione somma = new Frazione(0);  
int contaFrazioni = 0;  
  
// inserimento e calcolo della somma  
boolean continua = in.readSiNo("Vuoi inserire una " +  
                                "frazione (s/n)? ");
```

```
int num, den;
Frazione somma = new Frazione(0);
int contaFrazioni = 0;

// inserimento e calcolo della somma
boolean continua = in.readSiNo("Vuoi inserire una " +
                                "frazione (s/n)? ");

while (continua) {
    num = in.readInt("Numeratore? ");
    den = in.readInt("Denominatore? ");
    somma = somma.piu(new Frazione(num, den));
    contaFrazioni = contaFrazioni + 1;
    continua = in.readSiNo("Vuoi inserire un'altra " +
                            "frazione (s/n)? ");
}

//comunicazione dei risultati
...
```

```
int num, den;
Frazione somma = new Frazione(0);
int contaFrazioni = 0;

// inserimento e calcolo della somma
...

//comunicazione dei risultati
if (contaFrazioni == 0)
    out.println("Non e' stata inserita alcuna frazione");
else {
    Frazione media = somma.diviso(new Frazione(contaFrazioni));
    out.println("La somma e' " + somma.toString()+
                "; la media e' " + media.toString());
}
```

L'istruzione for

Sintassi

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

Sintassi

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

- *espr_inizializzazione*

È una lista di espressioni, separate virgola (,)

Sintassi

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

- ▶ *espr_inizializzazione*

È una lista di espressioni, separate virgola (,)

- ▶ *condizione*

È una qualunque espressione booleana

Sintassi

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

- ▶ *espr_inizializzazione*

È una lista di espressioni, separate virgola (,)

- ▶ *condizione*

È una qualunque espressione booleana

- ▶ *espr_incremento*

È una lista di espressioni

Sintassi

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

► *espr_inizializzazione*

È una lista di espressioni, separate virgola (,)

► *condizione*

È una qualunque espressione booleana

► *espr_incremento*

È una lista di espressioni

► *istruzione*

È una singola istruzione oppure un blocco di istruzioni

L'istruzione for: esecuzione

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

- (1) Vengono valutate le espressioni che compaiono in *espr_inizializzazione*

L'istruzione for: esecuzione

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

- (1) Vengono valutate le espressioni che compaiono in *espr_inizializzazione*
- (2) Viene valutata l'espressione *condizione*

L'istruzione for: esecuzione

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

- (1) Vengono valutate le espressioni che compaiono in *espr_inizializzazione*
- (2) Viene valutata l'espressione *condizione*
 - Se *condizione* è **true**:
 - viene eseguito il blocco di istruzioni nel corpo del ciclo
 - vengono valutate le espressioni che compaiono in *espr_incremento*
 - l'esecuzione prosegue dal **Punto (2)**

L'istruzione for: esecuzione

```
for (espr_inizializzazione; condizione; espr_incremento)  
    istruzione
```

- (1) Vengono valutate le espressioni che compaiono in *espr_inizializzazione*
- (2) Viene valutata l'espressione *condizione*
 - Se *condizione* è **true**:
 - viene eseguito il blocco di istruzioni nel corpo del ciclo
 - vengono valutate le espressioni che compaiono in *espr_incremento*
 - l'esecuzione prosegue dal **Punto (2)**
 - Se la condizione è **false**, l'esecuzione riprende dalla prima istruzione che segue l'istruzione **for**

```
int cont;  
...  
for (cont = 1; cont <= 10; cont = cont + 1)  
    out.println(cont);
```

```
int cont;  
...  
for (cont = 1; cont <= 10; cont = cont + 1)  
    out.println(cont);
```

```
for ( ; x != 0; ) {  
    x = in.readInt("Inserisci un numero ");  
    out.println(x);  
}
```

- L'espressione di inizializzazione può contenere direttamente la dichiarazione della variabile di controllo:

- L'espressione di inizializzazione può contenere direttamente la dichiarazione della variabile di controllo:

```
for (int cont = 1; cont <= 10; cont = cont + 1)  
    out.println(cont);
```


- L'espressione di inizializzazione può contenere direttamente la dichiarazione della variabile di controllo:

```
for (int cont = 1; cont <= 10; cont = cont + 1)  
    out.println(cont);
```

- In questo caso la variabile `cont` non è definita al di fuori del ciclo.

- L'espressione di inizializzazione può contenere direttamente la dichiarazione della variabile di controllo:

```
for (int cont = 1; cont <= 10; cont = cont + 1)  
    out.println(cont);
```

- In questo caso la variabile `cont` non è definita al di fuori del ciclo.
- È possibile dichiarare più variabili ma devono essere tutte dello stesso tipo:

- ▶ L'espressione di inizializzazione può contenere direttamente la dichiarazione della variabile di controllo:

```
for (int cont = 1; cont <= 10; cont = cont + 1)
    out.println(cont);
```

- ▶ In questo caso la variabile `cont` non è definita al di fuori del ciclo.
- ▶ È possibile dichiarare più variabili ma devono essere tutte dello stesso tipo:

```
for (int i = 1, j = 0; i + j <= 20; i = i + 1, j = j + 1)
    out.println(i + j);
```

L'istruzione break

`break`

Termina l'esecuzione del blocco dell'iterazione in cui compare.

L'istruzione break

break

Termina l'esecuzione del blocco dell'iterazione in cui compare.

```
for (int sx = 0, dx = s.length() - 1; sx < dx; sx = sx + 1,
    dx = dx - 1)
    if (s.charAt(sx) != s.charAt(dx)) {
        palindroma = false;
        break;
    }
```

PappagalloStanco.java

```
import prog.io.*;

class PappagalloStanco {
    public static void main(String[] args) {
        //predisposizione dei canali di comunicazione
        ...

        String messaggio;
        String risposta;

        do {
            messaggio = in.readLine();
            risposta = messaggio.toUpperCase();
            out.println(risposta);
            if (messaggio.equals("stanco"))
                break;
        } while (true);
    }
}
```

L'istruzione continue

`continue`

Provoca l'interruzione dell'esecuzione del blocco di istruzioni interne al ciclo e il passaggio all'iterazione successiva.

continue

Provoca l'interruzione dell'esecuzione del blocco di istruzioni interne al ciclo e il passaggio all'iterazione successiva.

- ▶ Nel caso dei cicli `while` o `do...while` si saltano le restanti istruzioni nel corpo del ciclo e si passa immediatamente alla valutazione della condizione.

continue

Provoca l'interruzione dell'esecuzione del blocco di istruzioni interne al ciclo e il passaggio all'iterazione successiva.

- ▶ Nel caso dei cicli **while** o **do...while** si saltano le restanti istruzioni nel corpo del ciclo e si passa immediatamente alla valutazione della condizione.
- ▶ Nel caso dei cicli **for** si passa a valutare le espressioni di incremento del ciclo e poi alla valutazione della condizione.

```
int x, somma = 0;
do {
    x = in.readInt();
    if (x == 0)
        break;
    if (x % 2 == 1)
        continue;
    somma += x;
} while (true);
```

```
int x, somma = 0;
do {
    x = in.readInt();
    if (x == 0)
        break;
    if (x % 2 == 1)
        continue;
    somma += x;
} while (true);
```

```
int x, somma = 0;
boolean continua = true;
do {
    x = in.readInt();
    if (x == 0)
        continua = false;
    else if (x % 2 == 0)
        somma += x;
} while (continua);
```