

SUPSI

# Istruzioni di Controllo

Loris Grossi, Fabio Landoni, Andrea Baldassari

Contenuto realizzato in collaborazione con: T. Leidi, A.E. Rizzoli, S. Pedrazzini

Fondamenti di Informatica

Bachelor in Ingegneria Informatica

# Obiettivo

Essere in grado di utilizzare in maniera corretta, all'interno di programmi, le varie istruzioni di controllo messe a disposizione da Java.

Obiettivi della lezione:

- Conoscere il concetto di espressione.
- Conoscere e saper usare i blocchi di codice.
- Conoscere e saper usare le istruzioni di selezione `if ... else if ... else` e `switch`.
- Conoscere e saper usare l'operatore ternario.
- Conoscere e saper usare le istruzioni di ripetizione `while`, `do ... while` e `for`.
- Conoscere e saper usare istruzioni di controllo annidate.
- Conoscere e saper usare le istruzioni `continue` e `break`.

## Sintassi, semantica e prammatica

Per poter utilizzare un **linguaggio di programmazione** bisogna conoscere la sua **sintassi** (insieme delle regole grammaticali):

- parole chiave riservate (for, int, true, ...),
- struttura dei moduli di un programma (uso delle parentesi),
- utilizzo dei vari separatori.

Inoltre, la scrittura di un programma per computer richiede la conoscenza di:

- **semantica**: significato delle operazioni che si vogliono far eseguire al computer,
- **prammatica**: il modo corretto di fare le cose, secondo consuetudine.

## Codice sorgente

Il codice sorgente di un programma viene realizzato combinando gli elementi:

- variabili, operatori, letterali, separatori, invocazioni di procedure e funzioni, ...
- ... in espressioni ...
- ... e le espressioni in istruzioni.

# Espressioni

Un'**espressione** è una **porzione di codice che rappresenta o calcola un valore**.

Un'**espressione semplice** può essere:

- un letterale,
- una variabile,
- una chiamata ad una procedura o funzione.

```
1573489
1000f
valore
eseguiCalcolo(10, 3)
richiediInput()
"buongiorno"
```

# Espressioni

Un'**espressione** viene **valutata al momento dell'esecuzione** e il suo valore può essere:

- assegnato ad una variabile,
- usato come parametro in una chiamata ad una procedura o funzione,
- combinato con altri valori mediante operatori rispettando le regole di precedenza (come già introdotto nelle scorse lezioni).

Le **espressioni complesse** si costruiscono combinando le espressioni semplici.

```
x++  
nuovoValore = valore + 10  
eseguiCalcolo(x + y, 10.0f)  
int risultato = calcolaValore(valorePrecedente)  
String input = richiediInput()
```

## Istruzioni

L'istruzione è l'**unità di esecuzione** in Java.

Ogni istruzione **deve essere completata con il punto e virgola** (';'), fatta eccezione per le istruzioni di controllo del flusso di codice.

Le istruzioni si suddividono in:

- **Istruzioni di dichiarazione**: utilizzate per dichiarare le variabili.
- **Istruzioni d'espressione**: espressioni d'assegnazione, invocazione di procedure o funzioni, espressioni di incremento o di decremento (++ o --), espressioni per la creazione d'oggetti.
- **Istruzioni per il controllo del flusso del codice**: utilizzate per controllare l'esecuzione del programma (ad esempio if e while).

## Istruzioni: esempio

```
/**  
 * Esempio di programma con varie istruzioni  
 */  
public class Istruzioni {  
    public static void main(String[] args) {  
        int valore = 0, j = 0;  
        while (j < 30) {  
            if (j == 9) {  
                valore++;  
                System.out.println("10");  
            } else if (j == 19) {  
                valore++;  
                System.out.println("20");  
            }  
            j++;  
        }  
        System.out.println("Valore finale: " + valore);  
    }  
}
```

# Programma

Quindi, in un programma dobbiamo:

- **dichiarare le variabili** che useremo,
- **scrivere le istruzioni** che specificano le operazioni da eseguire sulle variabili.

```
import java.util.Scanner;

public class TriangoloRettangolo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Base: ");
        double base = input.nextDouble();
        System.out.print("Altezza: ");
        double altezza = input.nextDouble();
        double area = base * altezza * .5;
        System.out.println("Area: " + area);
        input.close();
    }
}
```

## Blocchi, diramazioni e cicli

Le **strutture di controllo** si dividono in

- **blocchi**: servono per **organizzare** logicamente sequenze di istruzioni,
- **istruzioni di selezione** (diramazioni): servono per **indirizzare** il flusso di esecuzione in parti diverse del programma,
- **istruzioni di ripetizione** (cicli): servono per rappresentare in forma compatta l'**esecuzione ripetuta** di particolari sequenze di istruzioni.

Tramite queste strutture di controllo si possono costruire programmi di una discreta complessità.

## Blocchi di codice

Un blocco è una struttura per **organizzare una sequenza d'istruzioni**.

```
{  
    int nuovoValore = 2;  
    nuovoValore++;  
    System.out.println(nuovoValore);  
}
```

Un blocco può essere **vuoto**.

In un blocco si possono **dichiarare variabili locali**.

Una **variabile locale** è **visibile solo all'interno del blocco in cui è dichiarata**. Essa è accessibile, utilizzando l'identificatore, dal momento in cui viene dichiarata e fino alla fine del blocco in cui essa è contenuta.

## Blocchi di codice

Un blocco di codice può essere utilizzato ovunque si può collocare una singola istruzione.

In alcuni casi i blocchi di codice **sono obbligatori**, in altri casi sono opzionali.

All'interno dei blocchi è abitudine usare l'indentazione del codice.



## Blocchi di codice: esempi

```
{  
    System.out.print("La risposta é: ");  
    String risposta = "Nessuna idea!";  
    System.out.println(risposta);  
}
```

```
{  
    // Questo blocco scambia i valori di x e y.  
    int temp; // Variabile temporanea utilizzata nel blocco.  
    temp = x; // Memorizza il valore di x in temp.  
    x = y; // Copia il valore di y in x.  
    y = temp; // Copia il valore di temp in y.  
}
```

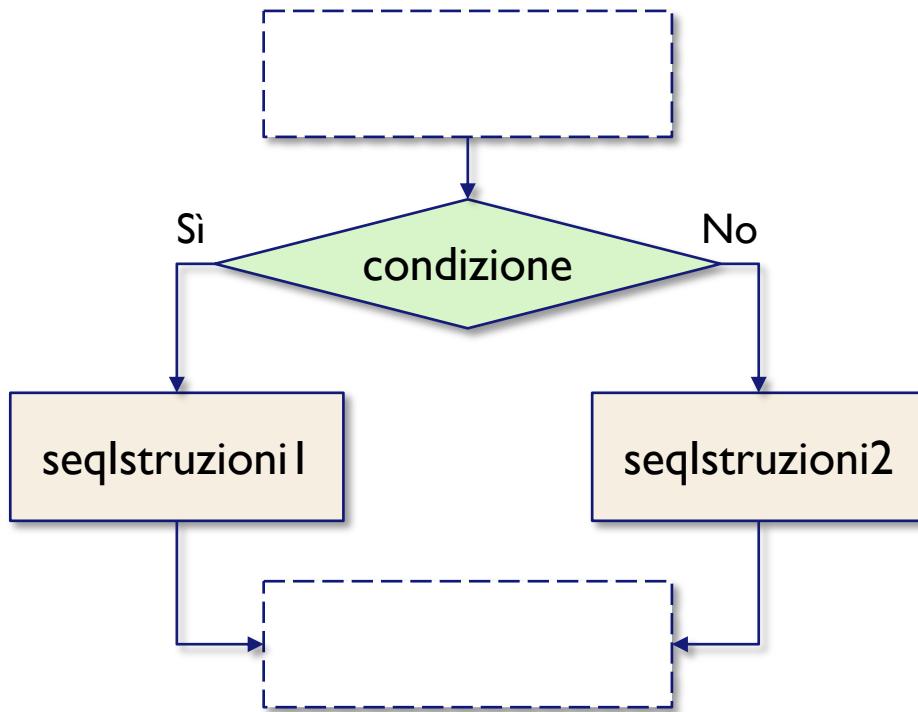
# Istruzioni di selezione in Java

```
if (condizione1) {  
    sequenzaIstruzioni1;  
} else if (condizione2) {  
    sequenzaIstruzioni2;  
} else if (condizione3) {  
    sequenzaIstruzioni3;  
} else {  
    sequenzaIstruzioni4;  
}
```

```
switch (espressione) {  
case valore1:  
    sequenzaIstruzioni10;  
    break;  
case valore2:  
    sequenzaIstruzioni11;  
    break;  
case valore3:  
    sequenzaIstruzioni12;  
    break;  
default:  
    sequenzaIstruzioni13;  
    break;  
}
```



## L'istruzione if ... else



```
if (condizione) {  
    seqIstruzioni1;  
} else {  
    seqIstruzioni2;  
}
```

## L'istruzione if ... else

L'istruzione if ... else permette di **eseguire parti di codice condizionalmente.**

L'istruzione if ... else **valuta espressioni che restituiscono valori di tipo boolean.**

La parte **else** è **opzionale**.

I blocchi vanno usati per aggregare più istruzioni.



## Istruzione condizionale semplice: esempio

```
import java.util.Scanner;

public class LimitaValore {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Inserisci un intero: ");
        int valore = scanner.nextInt();
        scanner.close();

        if (valore >= 10) {
            valore = 0;
        }
        System.out.println("Valore: " + valore);
    }
}
```



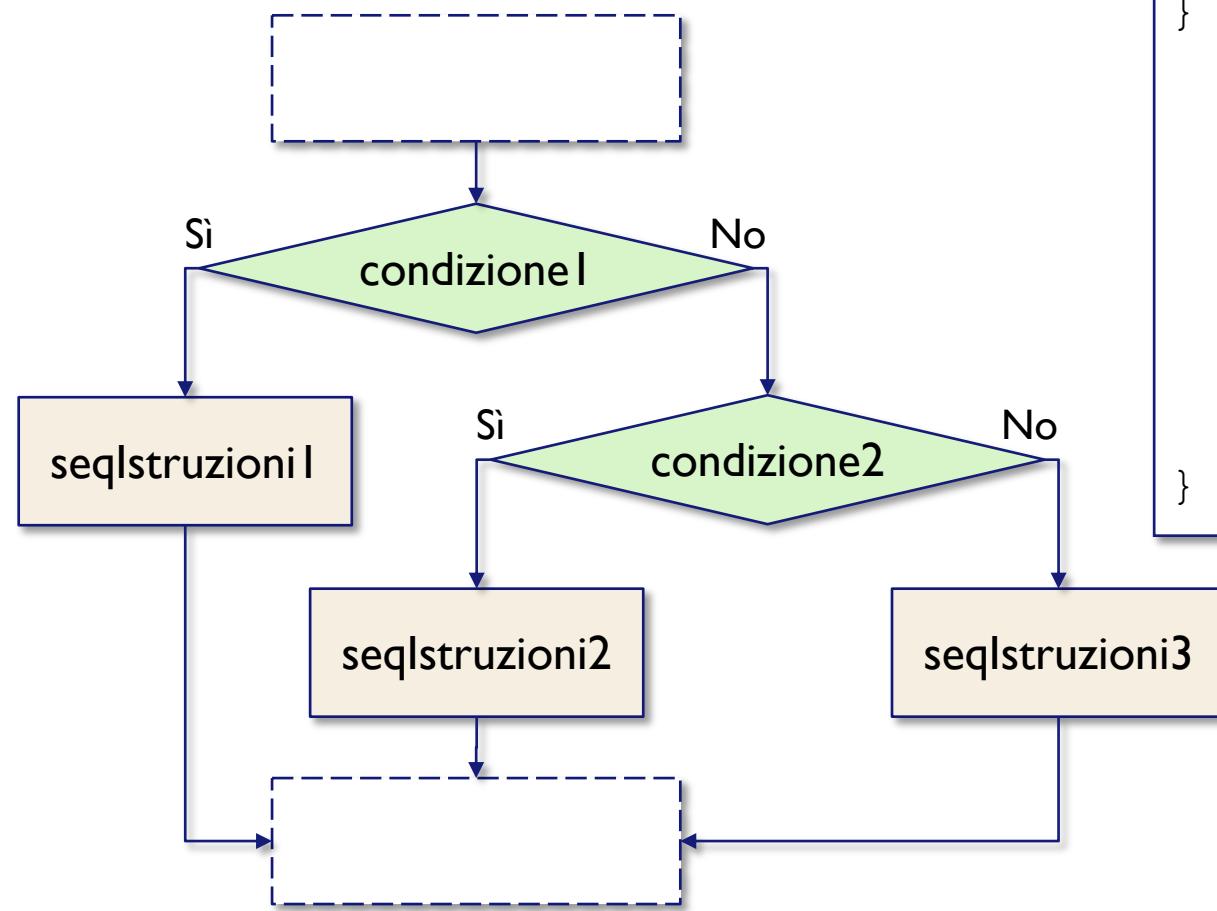
## Istruzione condizionale con alternativa: esempio

```
import java.util.Scanner;

public class MinMax {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Inserisci due interi: ");
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        scanner.close();
        int min, max;
        if (a > b) {
            min = b;
            max = a;
        } else {
            min = a;
            max = b;
        }
        System.out.println("Min: " + min + ", Max: " + max);
    }
}
```



## Istruzione if ... else ... if ... else



```
if (condizione1) {  
    seqIstruzioni1;  
} else {  
    if (condizione2) {  
        seqIstruzioni2;  
    } else {  
        seqIstruzioni3;  
    }  
}
```



## Istruzione if ... else if ... else

```
if (condizione1) {  
    sequenzaIstruzioni1;  
} else {  
    if (condizione2) {  
        sequenzaIstruzioni2;  
    } else {  
        sequenzaIstruzioni3;  
    }  
}
```

```
if (condizione1) {  
    sequenzaIstruzioni1;  
} else if (condizione2) {  
    sequenzaIstruzioni2;  
} else {  
    sequenzaIstruzioni3;  
}
```

I due modi di scrivere l'istruzione condizionale sono **esattamente equivalenti!**

La notazione di destra è da preferirsi in quanto più leggibile.



## Istruzione if ... else if ... else: esempio

```
import java.util.Scanner;

public class Temperatura {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Inserisci la temperatura: ");
        int temperatura = scanner.nextInt();
        scanner.close();

        if (temperatura < 18) {
            System.out.println("Fa freddo!");
        } else if (temperatura < 25) {
            System.out.println("Si sta bene.");
        } else {
            System.out.println("Fa caldo!");
        }
    }
}
```

## Sono la stessa cosa?

```
int x = 10, y = 15, z = 20;
if (x < y) {
    System.out.println("x minore di y");
} else if (x < z) {
    System.out.println("x minore di z");
} else if (x > z) {
    System.out.println("x maggiore di z");
}
```

```
int x = 10, y = 15, z = 20;
if (x < y) {
    System.out.println("x minore di y");
}
if (x < z) {
    System.out.println("x minore di z");
}
if (x > z) {
    System.out.println("x maggiore di z");
}
```

## Istruzione condizionale con &&: esempio

```
Scanner scanner = new Scanner(System.in);
System.out.print("Inserisci giorno, mese e anno: ");
int giorno = scanner.nextInt();
int mese = scanner.nextInt();
int anno = scanner.nextInt();
scanner.close();

// Calcola il giorno seguente

// ...

// Gestisci il caso dell'ultimo giorno dell'anno
if (giorno == 31 && mese == 12) {
    giorno = 1;
    mese = 1;
    anno++;
}

// ...
```



## Istruzione condizionale con ||: esempio

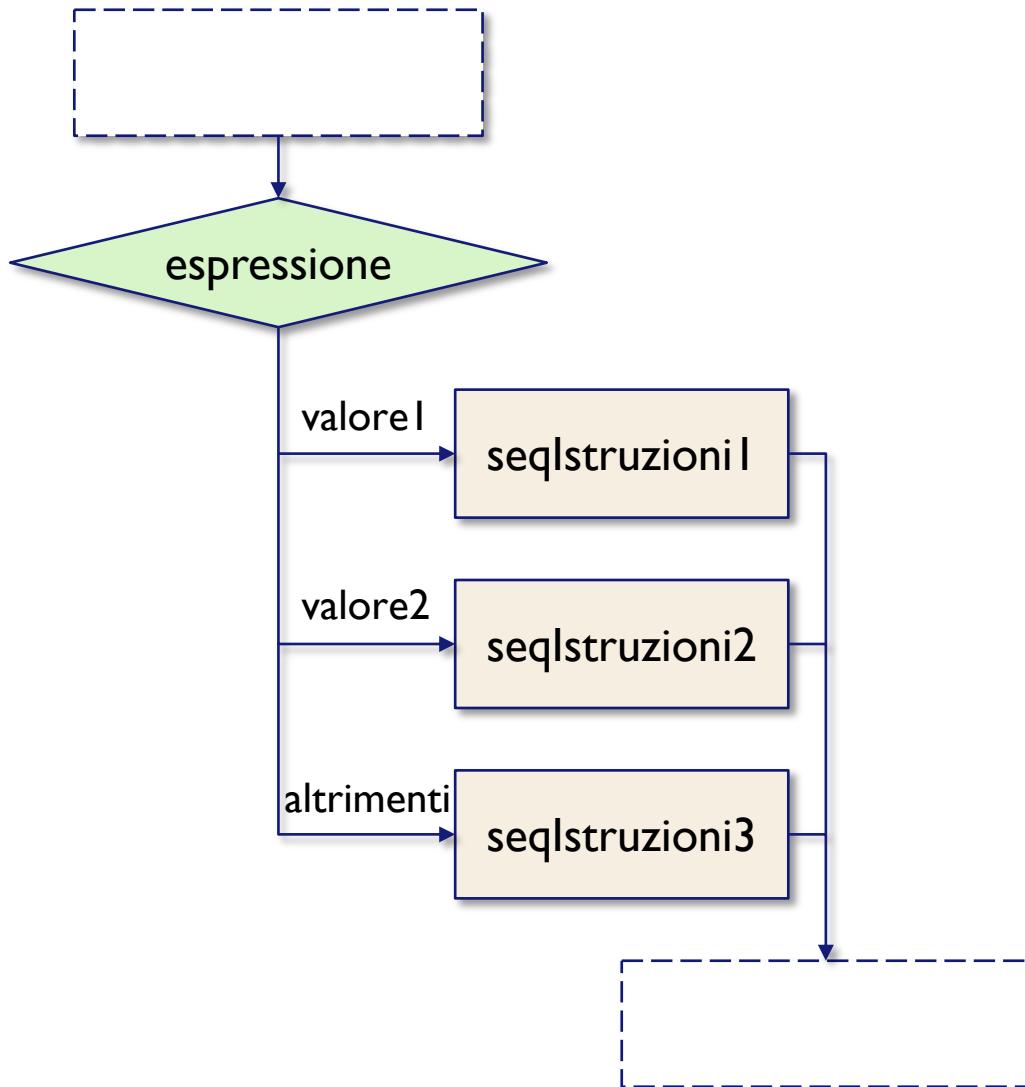
```
Scanner scanner = new Scanner(System.in);
System.out.print("Inserisci la durata: ");
int anni = scanner.nextInt();
System.out.print("Inserisci il valore: ");
double valore = scanner.nextDouble();
scanner.close();

// Eseguito in ogni caso
System.out.print("Il valore dell'investimento dopo ");

if (anni == 0 || anni > 1) {
    System.out.print(anni + " anni");
} else {
    // Gestisce il caso per 1 anno
    System.out.print("1 anno");
}

// Eseguito in ogni caso
System.out.println(" è di " + valore + " CHF.");
```

## L'istruzione switch tradizionale



```
switch (espressione) {  
    case valore1:  
        seqIstruzioni1;  
        break;  
    case valore2:  
        seqIstruzioni2;  
        break;  
    default:  
        seqIstruzioni3;  
        break;  
}
```

## L'istruzione switch tradizionale

L'istruzione `switch` permette di **gestire**, in maniera semplificata, **differenti piste di esecuzione** del codice (**selezione a n-vie**).

La selezione avviene sulla base del **valore assunto da un'espressione**.

L'istruzione `switch` funziona con i seguenti tipi di dato:

- `char`, `byte`, `short` e `int`,
- tipi enumerativi,
- `String` (da Java 7).

La sezione **`default`** è **opzionale**. Se presente, gestisce tutti i casi non gestiti nei vari `case`.

Un'istruzione **`if ... else`** può funzionare sulla base di **ranges di valori o condizioni**. Un'istruzione **`switch`** invece è basata esclusivamente su di un **singolo valore**.

## L'istruzione switch tradizionale: esempio

```
Scanner scanner = new Scanner(System.in);
System.out.print("Inserire un valore: ");
int c = scanner.nextInt();
scanner.close();
switch (c) {
case 1:
    System.out.print("1 ");
    break;
case 2:
    System.out.print("2 ");
    break;
case 3:
    System.out.print("3 ");
    break;
case 4:
    System.out.print("4 ");
    break;
default:
    System.out.print("4+ ");
}
```

Valore di c	Output
1	1
2	2
3	3
4	4
5	4+

## L'istruzione switch tradizionale: esempio con fall-through

```
Scanner scanner = new Scanner(System.in);
System.out.print("Inserire un valore: ");
int c = scanner.nextInt();
scanner.close();
switch (c) {
case 1:
    System.out.print("1 ");
case 2:
    System.out.print("2 ");
    break;
case 3:
    System.out.print("3 ");
case 4:
    System.out.print("4 ");
default:
    System.out.print("4+ ");
}
```

Valore di c	Output
1	1 2
2	2
3	3 4 4+
4	4 4+
5	4+

## L'istruzione switch tradizionale: esempio

```
Scanner input = new Scanner(System.in);
System.out.print("Inserisci un mese: ");
int mese = input.nextInt();
input.close();

int giorni = 0;
switch (mese) {
case 4: case 6: case 9: case 11:
    giorni = 30;
    break;
case 2:
    giorni = 28; // Ignorando gli anni bisestili
    break;
default:
    if (mese >= 1 && mese <= 12) {
        giorni = 31;
    } else {
        System.out.println("Mese non valido!");
    }
}
System.out.println("Il mese " + mese + " ha " + giorni + " giorni.");
```

## Il nuovo switch

Con il rilascio di Java 14 l'istruzione `switch` è stata aggiornata, introducendo alcune nuove caratteristiche, fra le quali troviamo:

- possibilità di usare la “notazione freccia” (->),
- `switch expression`.

Non tutti i linguaggi di programmazione supportano queste nuove caratteristiche. Per questo motivo è importante conoscere anche la sintassi ed il funzionamento dello `switch` “tradizionale”.

Per una maggiore chiarezza ci riferiamo con il termine di “nuovo” `switch` quando si vuole enfatizzare l'uso delle nuove caratteristiche.

## Il nuovo switch: la “notazione freccia” (->)

Uso della “notazione freccia” (->), ponendola al posto del simbolo “:”.

- non è più necessario usare la keyword break
- questa sintassi impedisce l'uso del fall-through
- ogni case può dichiarare più valori separati da virgole.

**Non è possibile mischiare le due notazioni** (“notazione freccia” e “notazione due punti”).

## Il nuovo switch: la “notazione freccia” (->)

### switch tradizionale

```
switch (espressione) {  
    case valore1:  
        seqIstruzioni1;  
        break;  
    case valore2:  
        seqIstruzioni2;  
        break;  
    default:  
        seqIstruzioni3;  
        break;  
}
```

### nuovo switch

```
switch (espressione) {  
    case valore1 -> {  
        seqIstruzioni1;  
    }  
    case valore2 -> {  
        seqIstruzioni2;  
    }  
    default -> {  
        seqIstruzioni3;  
    }  
}
```

## switch tradizionale: esempio

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a month: ");
int month = input.nextInt();
input.close();

String season;
switch (month) {
case 12: case 1: case 2:
    season = "winter";
    break;
case 3: case 4: case 5:
    season = "spring";
    break;
case 6: case 7: case 8:
    season = "summer";
    break;
case 9: case 10: case 11:
    season = "autumn";
    break;
default:
    season = "not identifiable";
    break;
}

System.out.println("The season is " + season + ".");
```

## Nuovo switch: esempio “notazione freccia”

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a month: ");
int month = input.nextInt();
input.close();

String season;

switch (month) {
case 12, 1, 2 -> season = "winter";
case 3, 4, 5 -> season = "spring";
case 6, 7, 8 -> season = "summer";
case 9, 10, 11 -> season = "autumn";
default -> season = "not identifiable";
}

System.out.println("The season is " + season + ".");
```

## Il nuovo switch come espressione

Il **nuovo switch** può essere usato come **espressione**, ossia un costrutto che **ritorna un valore**.

- La clausola **default** è **obbligatoria** (esaustività dello switch).
- Non dimenticare il **punto e virgola dopo la parentesi di chiusura dello switch**.

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a month: ");
int month = input.nextInt();
input.close();

String season = switch (month) {
    case 12, 1, 2 -> "winter";
    case 3, 4, 5 -> "spring";
    case 6, 7, 8 -> "summer";
    case 9, 10, 11 -> "autumn";
    default -> "not identifiable";
};

System.out.println("The season is " + season + ".") ;
```



## Il nuovo switch come espressione: yield

Se la “**notazione freccia**” punta ad un blocco di codice contenente **diverse istruzioni**, per ritornare il valore si usa la keyword **yield** e non la keyword return.

```
String season = switch (month) {  
    case 12, 1, 2 -> "winter";  
    case 3, 4, 5 -> "spring";  
    case 6, 7, 8 -> "summer";  
    case 9, 10, 11 -> "autumn";  
    default -> {  
        System.out.println("Provided number is not a valid month!");  
        yield "not identifiable";  
    }  
};
```



## Il nuovo switch come espressione: yield

Con la “**notazione freccia**”, la keyword **yield** può essere usata **solo all'interno di blocchi di codice**.

```
String season = switch (month) {  
    case 12, 1, 2 -> yield "winter";  
    case 3, 4, 5 -> yield "spring";  
    case 6, 7, 8 -> yield "summer";  
    case 9, 10, 11 -> yield "autumn";  
    default -> yield "not identifiable";  
};
```

```
String season = switch (month) {  
    case 12, 1, 2 -> { yield "winter"; }  
    case 3, 4, 5 -> { yield "spring"; }  
    case 6, 7, 8 -> { yield "summer"; }  
    case 9, 10, 11 -> { yield "autumn"; }  
    default -> { yield "not identifiable"; }  
};
```

## Operatore ternario

In Java esiste l'operatore ternario “?”. È un operatore condizionale, detto anche **inline if**.

```
condizione ? valoreSeVero : valoreSeFalso
```

L'operatore restituisce il primo valore (`valoreSeVero`) se la condizione è vera. Altrimenti restituisce il secondo valore (`valoreSeFalso`).

Permette di scrivere, in maniera compatta, espressioni del tipo: **“se ... allora ... altrimenti”**.

# Operatore ternario

```
prossimo = (n % 2 == 0) ? (n / 2) : (3 * n + 1);
```

Equivale a:

```
if (n % 2 == 0) {
    prossimo = n / 2;
} else {
    prossimo = 3 * n + 1;
}
```



## Operatore ternario: esempio

```
import java.util.Scanner;

public class ValoreAssoluto {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Inserisci un valore: ");
        int valore = input.nextInt();
        input.close();

        // Calcola il valore assoluto
        int abs = valore < 0 ? -valore : valore;
        System.out.print("Il valore assoluto di ");
        System.out.println(valore + " è " + abs);
    }
}
```

# Istruzioni di ripetizione in Java

```
while (condizione1) {  
    sequenzaIstruzioni1;  
}
```

Eseguito **zero o più** volte.

```
do {  
    sequenzaIstruzioni2;  
} while (condizione2);
```

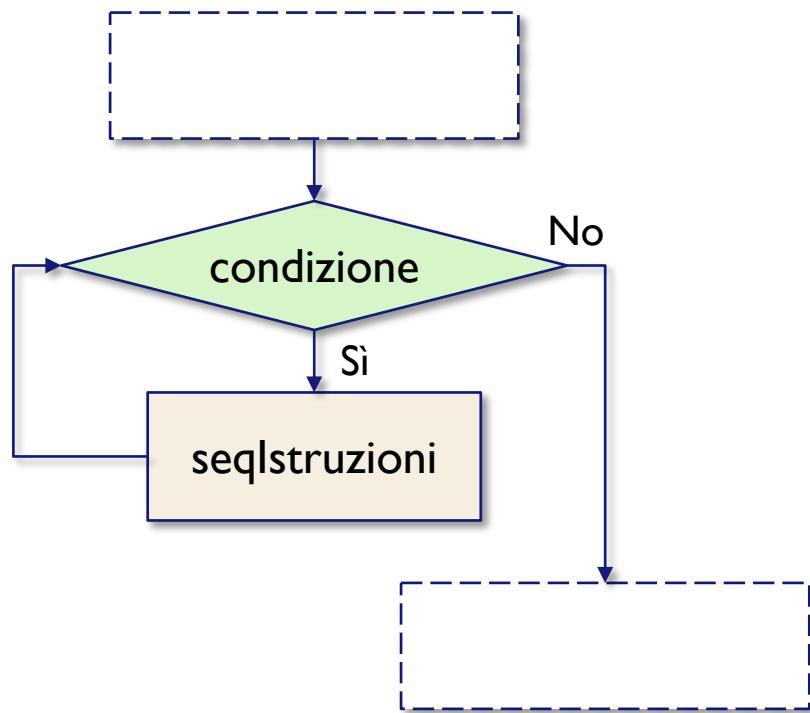
Eseguito **una o più** volte.

```
for (int i = valoreIniziale; i < valoreFinale; i++) {  
    sequenzaIstruzioni3;  
}
```

Eseguito un **numero fisso** di volte.



# L'istruzione while



```
while (condizione) {  
    seqIstruzioni;  
}
```

## L'istruzione while

L'istruzione `while` permette di **eseguire una o più istruzioni ripetutamente**.

L'esecuzione ripetuta prosegue fino a quando una determinata **condizione rimane vera**.

L'istruzione `while` valuta ripetutamente un'**espressione** che restituisce un valore di tipo **boolean**.

Può eseguire anche **zero volte**.



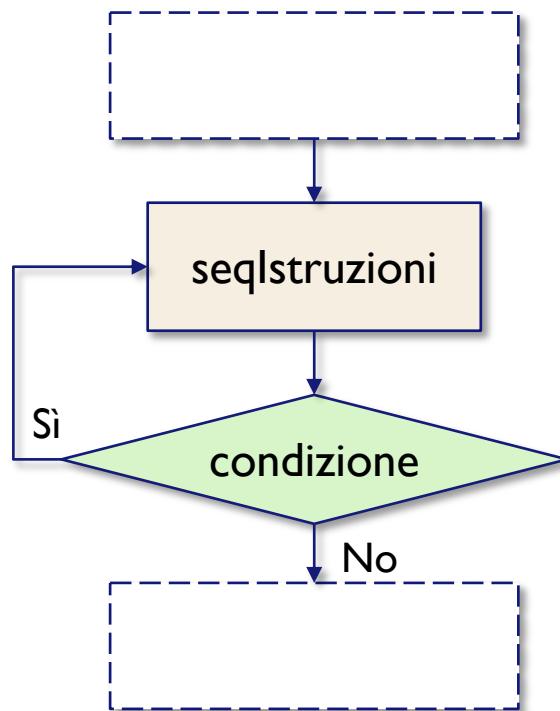
## L'istruzione while: esempio

```
import java.util.Scanner;

public class Sequenza {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Limite massimo: ");
        int limiteMax = scanner.nextInt();
        scanner.close();

        int numero = 0;
        while (numero <= limiteMax) {
            System.out.print(numero + " ");
            numero++;
        }
        System.out.println("\nFine");
    }
}
```

## L'istruzione do ... while



```
do {  
    seqIstruzioni;  
} while (condizione);
```

## L'istruzione do ... while

L'istruzione do ... while permette di **eseguire una o più istruzioni ripetutamente.**

L'esecuzione ripetuta prosegue fino a quando una determinata **condizione rimane vera.**

L'istruzione do ... while valuta ripetutamente un'**espressione** che restituisce un valore di tipo **boolean**.

Esegue **almeno una volta**.

## L'istruzione do ... while: esempio

```
import java.util.Scanner;

public class Sequenza {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Limite massimo: ");
        int limiteMax = scanner.nextInt();
        scanner.close();

        int numero = 0;
        do {
            System.out.print(numero + " ");
            numero++;
        } while (numero <= limiteMax);
        System.out.println("\nFine");
    }
}
```

## Istruzione while e do ... while: corrispondenze

```
do {  
    seqIstruzioni1;  
} while (condizione1);
```

```
seqIstruzioni1;  
while (condizione1) {  
    seqIstruzioni1;  
}
```

```
while (condizione2) {  
    seqIstruzioni2;  
}
```

```
if (condizione2) {  
    do {  
        seqIstruzioni2;  
    } while (condizione2);  
}
```

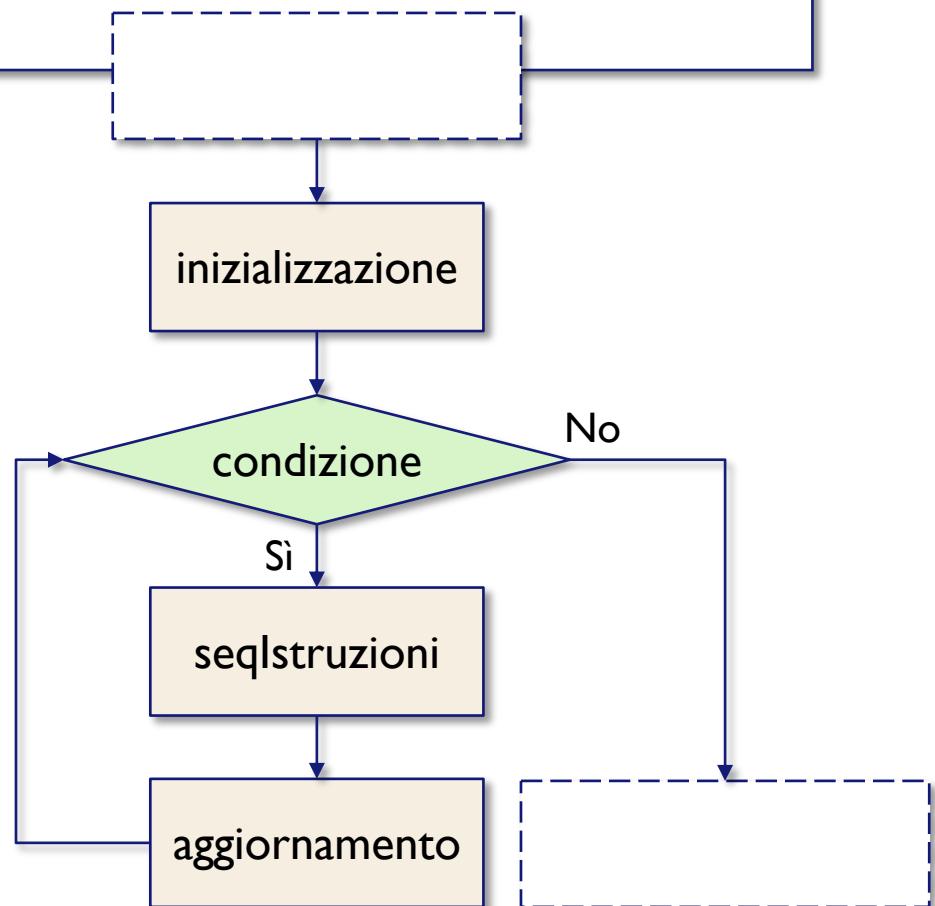
## Esempio

```
Scanner scanner = new Scanner(System.in);
boolean continua = true;
while (continua) {
    //...
    System.out.println("Vuoi continuare? (true/false)");
    continua = scanner.nextBoolean();
}
scanner.close();
```

```
Scanner scanner = new Scanner(System.in);
boolean continua = true;
do {
    //...
    System.out.println("Vuoi continuare? (true/false)");
    continua = scanner.nextBoolean();
} while (continua);
scanner.close();
```

## L'istruzione for

```
for (inizializzazione; condizione; aggiornamento) {  
    seqIstruzioni;  
}
```



## L'istruzione for

L'istruzione `for` mette a disposizione una **soluzione compatta per iterare all'interno di un range di valori.**

Per ogni istruzione `for` è possibile specificare:

- L'**espressione di inizializzazione**: viene eseguita una sola volta quando il ciclo comincia.
- L'**espressione di proseguimento / terminazione**: viene valutata all'inizio di ogni iterazione per decidere quando terminare il ciclo.
- L'**espressione di incremento / aggiornamento**: viene eseguita alla fine di ogni iterazione.

L'esecuzione ripetuta prosegue fino a quando la **condizione di proseguimento rimane vera**.

## L'istruzione for: esempio

```
import java.util.Scanner;

public class Interessi {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Numero di anni: ");
        int anni = scanner.nextInt();
        System.out.print("Saldo iniziale: ");
        double saldo = scanner.nextDouble();
        System.out.print("Tasso d'interesse: ");
        double tassoInteresse = scanner.nextDouble();
        scanner.close();

        for (int i = 0; i < anni; i++) {
            saldo += saldo * tassoInteresse;
            System.out.print("Anno: " + (i + 1));
            System.out.println(" -> Saldo: " + saldo);
        }
    }
}
```

## L'istruzione `for`: cicli di conteggio

La forma tipica d'utilizzo dell'istruzione `for` è quella del **ciclo di conteggio con incremento o decremento**:

```
for (tipo contatore = min; contatore < max; contatore++) {  
    sequenzaIstruzioni;  
}
```

La variabile `contatore` assume tutti i valori tra `min` (**compreso**) e `max` (**escluso**).

Ad esempio:

```
for (int i = 0; i < 10; i++) {  
    System.out.println(i);  
}
```

```
for (int i = 9; i >= 0; i--) {  
    System.out.println(i);  
}
```

# L'istruzione for: esempi cicli di conteggio

Calcolo del fattoriale:

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

Conteggio delle doppie:

```
String frase = "Se oggi seren non è, doman seren sarà,"
        + " se non sarà seren si rassurererà";
int cnt = 0;
for (int i = 1; i < frase.length(); i++) {
    if (frase.charAt(i) == frase.charAt(i - 1)) {
        cnt++;
    }
}
System.out.println("Conteggio = " + cnt);
```

## L'istruzione for: alcune note

L'espressione di aggiornamento **non deve necessariamente essere un incremento unitario.**

```
for (int i = 0; i <= 20; i = i + 2) {  
    System.out.print(i + " ");  
}
```

Le espressioni d'inizializzazione, di proseguimento e d'aggiornamento sono **opzionali**.

```
int i = 0;  
// Alcune istruzioni  
for (; i <= 20;) {  
    i = i + 2;  
    System.out.print(i + " ");  
}
```

```
int i = 0;  
// Ciclo infinito  
for (;;) {  
    i = i + 2;  
    System.out.print(i + " ");  
}
```

## L'istruzione for: particolarità

Le variabili ‘contatore’ possono essere di **qualsiasi tipo (deve permettere l'operazione somma)**.

```
for (char ch = 'A'; ch <= 'Z'; ch++) {  
    System.out.print(ch);  
}
```

L'istruzione for permette di definire ed operare su **più di una variabile ‘contatore’** alla volta. Si consiglia, però, di fare parecchia attenzione. In particolare per evitare dei cicli che non terminano mai, come ad esempio:

```
for (double i = 0.5, j = 39; i + j < 40; i++, j--) {  
    // ...  
}
```

## L'istruzione for: particolarità

Inoltre, all'interno del blocco di codice di un'istruzione for è possibile modificare i valori delle variabili 'contatore'.

Si consiglia di evitare questo tipo di soluzione perché rende il codice **poco comprensibile e poco robusto**.

```
for (int i = 0; i < 40; i += 2) {  
    // ...  
  
    i--;  
  
    // ...  
}
```

## Istruzione for e while: corrispondenze

```
for (inizializzazione; condizione; aggiornamento) {  
    seqIstruzioni;  
}
```

```
inizializzazione;  
while (condizione) {  
    seqIstruzioni;  
    aggiornamento;  
}
```

Il comportamento dei due codici è **equivalente**.

## Istruzioni e blocchi

All'interno di un'istruzione per il controllo del flusso del codice, se viene eseguita **una sola istruzione**, non è necessario specificare un nuovo blocco di codice.

```
if (valore < 15) {  
    System.out.println(valore);  
} else if (valore < 20) {  
    System.out.println("grande");  
} else {  
    System.out.println("gigante");  
}
```

```
int x = 0;  
while (x < 5) {  
    System.out.println(x++);  
}
```

```
if (valore < 15)  
    System.out.println(valore);  
else if (valore < 20)  
    System.out.println("grande");  
else  
    System.out.println("gigante");
```

```
int x = 0;  
while (x < 5)  
    System.out.println(x++);
```

## Istruzioni di controllo annidate

Se necessario, è possibile **innestare più di un'istruzione per il controllo del flusso del codice nell'altra.**

```
int k = 0;
for (int i = 0; i < 40; i++) {
    for (int j = 0; j < 20; j++) {
        k += i - j;
    }
    for (int j = 0; j < 20; j++) {
        k -= j + i;
    }
}
```

```
for (int i = 0; i < 40; i++) {
    if (i < 10) {
        // ...
    } else if (i < 20) {
        // ...
    } else {
        // ...
    }
}
```

## Istruzioni di controllo annidate: esempio

Si vuole visualizzare a schermo la seguente sequenza di valori:

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	14	16	18	20	22	24
3	6	9	12	15	18	21	24	27	30	33	36
4	8	12	16	20	24	28	32	36	40	44	48
5	10	15	20	25	30	35	40	45	50	55	60
6	12	18	24	30	36	42	48	54	60	66	72
7	14	21	28	35	42	49	56	63	70	77	84
8	16	24	32	40	48	56	64	72	80	88	96
9	18	27	36	45	54	63	72	81	90	99	108
10	20	30	40	50	60	70	80	90	100	110	120
11	22	33	44	55	66	77	88	99	110	121	132
12	24	36	48	60	72	84	96	108	120	132	144

## Istruzioni di controllo annidate: esempio

Possibile soluzione:

```
// Percorri tutte le righe
for (int riga = 1; riga <= 12; riga++) {
    // Percorri tutte le colonne di ogni riga
    for (int colonna = 1; colonna <= 12; colonna++) {
        // Mostra il risultato corrente
        System.out.print(riga * colonna + "\t");
    }
    // Inizia una nuova riga
    System.out.println();
}
```

## L'istruzione continue

L'istruzione **continue** permette di **interrompere l'iterazione corrente** di un ciclo for, while o do ... while ma senza interrompere il ciclo. **L'esecuzione prosegue con l'iterazione successiva.**

```
for (int i = 1; i < 10; i++) {  
    if (i == 5) {  
        continue;  
        // Codice mai raggiunto  
    }  
    System.out.println("Valore di i: " + i);  
}
```

## L'istruzione continue: esempio

```
double numero, somma = 0.0;
int cnt = 0;
Scanner input = new Scanner(System.in);

for (int i = 1; i < 6; ++i) {
    System.out.print("Inserire un numero: ");
    numero = input.nextDouble();
    if (numero == 0.0)
        continue;

    somma += numero;
    cnt++;
}

if (cnt != 0)
    System.out.println("Media = " + somma / cnt);

input.close();
```

## L'istruzione break

Si può utilizzare l'istruzione **break** per **terminare un ciclo** for, while o do ... while. **L'esecuzione continua dalla prima istruzione che segue il loop.**

```
String frase = "Lasciate ogni speranza, voi ch'entrate.";
char c = '\'';
int cnt = 0;
for (int i = 0; i < frase.length(); i++) {
    if (frase.charAt(i) == c)
        break;
    cnt++;
}
if (cnt == frase.length())
    System.out.println("Carattere non trovato!");
else {
    System.out.print("Ci sono " + cnt);
    System.out.println(" caratteri prima di '" + c + "'.");
}
```

## L'istruzione break: esempio

```
Scanner scanner = new Scanner(System.in);
// Genera un numero casuale tra 1 e 50
int numeroCasuale = (int) (Math.random() * 50) + 1;
// Ciclo infinito
while (true) {
    System.out.print("Indovina un numero tra 1 e 50: ");
    int numero = scanner.nextInt();
    if (numero == numeroCasuale) {
        break;
    } else if (numero < numeroCasuale) {
        System.out.println("Troppo piccolo!");
    } else {
        System.out.println("Troppo grande!");
    }
}
System.out.println("Complimenti, hai indovinato!");
scanner.close();
```

# Riepilogo

- **Istruzione di selezione** if ... else if ... else
- **Istruzione di selezione** switch
- **Operatore ternario**
- **Istruzione di ripetizione** while
- **Istruzione di ripetizione** do ... while
- **Istruzione di ripetizione** for
- **Istruzioni annidate**
- **Istruzione** continue
- **Istruzione** break