Primo programma in Java

```
public class Main {
    public static void main(S tring[] args) {
        Sys tem.ou t.p rin tln ("Hello World");
    }
}
```

- Ogni riga di codice eseguita in Java deve trovarsi all'interno di una classe.
- Il nome della classe deve sempre iniziare con una lettera maiuscola e deve essere uguale al nome del file Java. Questo perché Java utilizza il nome della classe per trovare ed eseguire il codice. Se i nomi non corrispondono, Java genererà un errore e il programma non verrà eseguito.
- Il metodo `main()` è necessario in ogni programma Java. È il punto in cui il programma inizia a essere eseguito.

Operatori aritmetici

Simbolo	Descrizione
+	somma
-	sottrazione
/	divisione
*	moltiplicazione
% (operatore modulo)	resto della divisione
++	incremento
--	decremento

Operatori di confronto

Simbolo	Descrizione
<	minore
>	maggiore
<=	minore o uguale
>=	maggiore o uguale
==	uguale
!=	diverso

Operatori logici

Simbolo	Descrizione
&&	AND (se tutte le condizioni devono essere `true`)
||	OR (se almeno una condizione deve essere `true`)
!	NOT (per invertire una condizione)

WHILE

```
while (condizione) {
    // blocco di codice da eseguire finchè la condizione è true
}
```

NOTA:
- è necessario una variabile contatore che conta il numero dell'iterazione. Di solito è i perchè sta per index o iterator.
- è necessario aggiornare il valore della variabile (es. i++), altrimenti il ciclo non si interrompe mai.

DO-WHILE

```
do {
    // blocco di codice da eseguire
}
while (condi zione);
```

Ciclo FOR

```
for (inizializzazione; condizione; incremento/decremento) {
    // blocco di codice da eseguire
}
```

Si usa quando si sa esattamente quante volte si vuole ripetere un blocco di codice (es. 10 volte).
Nota:
- inizializzazione viene eseguita (1 volta) prima dell'esecuzione del blocco di codice
- la condizione viene verificata ogni volta prima di eseguire il blocco di codice
- incremento/decremento viene eseguito ogni volta dopo che il blocco di codice è stato eseguito

Cicli annidati

```
// Ciclo esterno
for (int i = 1; i <= 2; i++) {
    Sys tem.ou t.p rin tln ("Ciclo esterno: " + i);
// Eseguito due volte

   // Ciclo interno
   for (int j = 1; j <= 3; j++) {
        Sys tem.ou t.p rin tln (" Ciclo interno: "
+ j); // Eseguito 6 volte (2 * 3)
   }
}
```

Il ciclo interno viene eseguito una volta per ogni iterazione del ciclo esterno.

Operatore ternario

```
variabile = (condizione) ? espressioneTrue :
espressioneFalse;
// Esempio con if - else
int time = 20;
if (time < 18) {
    Sys tem.ou t.p rin tln ("Good day.");
} else {
    Sys tem.ou t.p rin tln ("Good evenin g.");
}
// Esempio analogo con l'oper atore ternario
int time = 20;
String result = (time < 18) ? "Good day." : "Good
evenin g.";
System.ou t.p rin tln (re sult);
```

L'operatore ternario è una short-hand ("abbreviazione") del costrutto if-else.

BREAK e CONTINUE

```
// L'istruzione break può essere utilizzata per
uscire da uno switch o da un ciclo
for (int i = 0; i < 10; i++) {
   if (i == 4) {
        break;
   }
    Sys tem.ou t.p rin tln(i);
```

BREAK e CONTINUE (cont)

```
> }
// L'istruzione continue interrompe un'iterazione (nel ciclo), se si
verifica una condizione specificata, e continua con l'iterazione
successiva nel ciclo.
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

Output (mostrare qualcosa a schermo)

```
System.out.println("Messaggio da visualizzare");
System.ou t.p rin tln (no me_ var iab ile);
System.ou t.p rin tln ("Me ssaggio da visual izz -
are " + nome_v ari abile);
```

ove:
- `System` è una classe built-in in Java
- `out` è un membro di System e sta per "output"
- `println()` è un metodo che permette di stampare sulla console e posizionare il cursore a capo.
println sta per print line.

ATTENZIONE: Il messaggio da visualizzare deve essere compreso sempre tra "" essendo delle stringhe.

Esiste anche la funzione `System.ou t.p rint()` che permette semplicemente di stampare (ovvero visualizzare) qualcosa sulla console.

Commenti

```
// commento su singola riga
/* commento
su più
righe */
```

I commenti servono per:
- documentare il codice
- avere delle note che indicano cosa si sta facendo
- fare debugging (ovvero trovare l'errore e correggerlo)

Tipi di dato

Primitivi	Reference
Sono predefiniti e built-in in Java.	Sono creati dal programmatore (tranne per `String`).
/	Possono chiamare metodi/proprietà.
Iniziano con la lettera minuscola.	Iniziano con la lettera maiuscola.
Contengono sempre un valore.	Possono anche essere `null`.
Esempi: String, Array	Esempi: int, char, boolean, float, double

Tipi di dato primitivi

Tipo di dato primitivo	Descrizione
byte	Memorizza numeri interi.
short	Memorizza numeri interi.
int	Memorizza numeri interi
long	Memorizza numeri interi
float	Memorizza numeri floating point (es. 15.4) con 6-7 cifre decimali.
double	Memorizza numeri floating point con 15-16 cifre decimali.
boolean	Memorizza un valore che può essere `true`/ `false`
char	Memorizza un singolo carattere che deve essere compreso tra "

Nota:
Quando si assegna un valore a un dato di tipo long, float e double, bisogna specificare una lettera che indica il tipo di dato:
- L per long (es. `long numero = 16578L`)
- f per float (es. `float numero = 56.78f`)
- d per double (es. `double numero = 56.783324f`)

Variabili e costanti

```
//dichiarazione di una costante
final tipo_dato NOME_C OSTANTE = valore;
// dichia razione di una variabile
tipo_dato nome_v ari abile;
// dichia razione di più variabili
// devono essere dello stesso tipo
tipo_dato nome_v ari abile1, nome_v ari abile2;
//asse gna zione di un valore a una variabile
(serve per modificare il contenuto di una
variabile)
nome_v ari abile = valore;
// inizia liz zazione di una variabile (dichi -
ara zione + valore iniziale)
tipo_dato nome_v ari abile = valore;
```

IF

```
if(condizione) {
   // blocco di codice da eseguire se la
condizione è true
}
if(con diz ione)
      singola istruzione da eseguire se la
condizione è true
```

NOTA: Le parentesi {} si possono omettere se è solo una l'istruzione da eseguire se la condizione è verificata. Tuttavia, è meglio usarle lo stesso per rendere il codice più facile da leggere e per prevenire errori.

IF- ELSE

```
if (condizione) {
   // blocco di codice da eseguire se la
condizione è true
} else {
   // blocco di codice da eseguire se la
condizione è false
}
```

IF-ELSE IF- ELSE

```
if (condizione1) {
   // blocco di codice da eseguire se la condiz -
ione1 è true
} else if (condi zione2) {
    //b locco di codice da eseguire se la condiz -
ione1 è false e la condiz ione2 è true
} else {
   // blocco di codice da eseguire se la condiz -
ione1 e la condiz ione2 sono false.
}
```

IF ANNIDATI

```
if (condizione1) {
   // il codice viene eseguito se la condiz ione1
è true
   if (condi tion2) {
       // il codice viene eseguito se la condiz -
ione1 e la condizione 2 sono true
   }
}
```

SWITCH

```
switch(espressione) {
   case valore case1:
       // blocco eseguito nel caso valore case1
        break;
   case valore case2:
       // blocco eseguito nel caso valore case2
        break;
    def ault:
       // blocco eseguito in caso tutti gli altri
casi non si sono verificati
}
```

E' possibile usare l'istruzione `switch`, al posto di molte istruzioni `if ...else`.
- l'espressione in switch viene valutata una volta
- il risultato viene confrontato con il valore di ciascun `case`
- se c'è una corrispondenza, viene eseguito il blocco di codice corrispondente.
- l'istruzione `break` interrompe lo switch dopo l'esecuzione del caso corrispondente.
- l'istruzione `default` viene eseguita se non c'è nessuna corrispondenza.