Introduzione a Java

Comandi e Nozioni di Base

```
public class Main{                              // Inizializza la classe Main
    public static void main(String[] args){ // Inizializza il metodo main, il punto di partenza
     // Something something                 // del codice
    }
}
```

```
public static void NomeDelMetodo(){ // E' il modo standard per inizializzare un metodo, in particolare
...                                 // solo NomeDelMetodo può essere qualsiasi. Si dice firma del metodo
...
}
```

```
System.out.println("Hello World!"); // Stampa e "Hello World!" e va a capo
System.out.print("Hello World!");  // Stampa senza andare a capo
```

I blocchi di codice sono separati da graffe {} (basta contare quante se ne aprono)
L'indentazione (la spaziatura delle righe) e' **importante** per questioni di chiarezza e leggibilità.

Le **variabili** vanno *inizializzate* prima di poter essere usate. Per esempio:

```
...
int IlMioPrimoPreferito = 2;
...
```

L'operatore = è l'operatore di assegnazione, e *assegna* alla variabile IlMioPrimoPreferito il valore 2.
l'assegnazione **NON** è opzionale, prima di usare una variabile va sempre inizializzata.

E' buona norma dare dei nomi **sensati** alle variabili, Rende più semplice sia la fase di scritture che la fase di revisione.
Pena: *suicidio professionale*.

Inoltre devono essere **unici** per tutto il codice, indipendentemente dal loro tipo.

I **;** vanno inseriti alla fine di **ogni** istruzione, in caso contrario, il programma va in crash.

Il linguaggio Java è *case sensitive*, per esempio *void* **NON** è *VOID*.

Operazioni matematiche

```
public class Main{
    public static void main(String[] args){
        int a=5;
        int b=2;
        System.out.println(a+b); //Somma
        System.out.println(a-b); //Differenza
        System.out.println(a*b); //Prodotto
        System.out.println(a/b); //Divisione
        System.out.println(a%b); //Modulo (resto della divisione tra a e b)
        System.out.println(++a); //Incremento per 1
        System.out.println(--a); //Decremento per 1
    }
}

OUTPUT:
7    //5 + 2 = 7
3    //5 - 2 = 3
10   //5 * 2 =10
2    //5 / 2 = 2 (il compilatore arrotonda all'intero più piccolo, a meno che a e b
siano di tipo float)
1    //5 % 2 = 1 (Perché 5= 2*2 + 1)
6    // 5 + 1 = 6
4    // 5 - 1 = 4
```

Richiamare un metodo

```
public class Main{
    public static void main(String[] args){
        somma(5,8);
    }
```

```
    public static void somma(int a, int b){
        System.out.println(a+b);
    }
}

OUTPUT:
13
```

il metodo somma accetta due *argomenti* di tipo *intero*. Se al momento della chiamata, i due valori sono:
A) non interi, o
B) non sono due,
avremo un *errore* e il programma non partirà.

somma(5,8) e' detta *chiamata* del metodo *somma* con argomenti 5 e 8.

Si può chiamare un metodo dentro un altro.

Inoltre, un metodo potrebbe non accettare alcun argomento come ad esempio un ipotetico metodo *Hello()*:

```
...
public static void Hello(){
    System.out.println("No args required!");
}
...

OUTPUT:
No args required!
```

Nota:
Un metodo inizializzato *non* va chiamato per forza. Tendenzialmente ogni pezzo di codice deve avere senso di esistere ma mentre si *debuggando* il programma, commentare le parti che non danno problemi è una buona idea.

la keyword *void* indica che il metodo dichiarato *non* ritornerà alcun dato alla fine dell'esecuzione.
In caso non sia presente la keyword *void* è necessario ad un certo punto del metodo (ragionevolmente la fine) usare la keyword *return* seguita dal un dato dello stesso tipo indicato nella *firma* del metodo. Per esempio:

```
public class Main{
    public static void main(String[] args){
        int z= Somma5(3);
        System.out.println(z);
```

```
    }

    static int Somma5(int x){
        return x+5;
    }
}

OUTPUT:
8
```

Error handling 101

Ci sono principalmente *2* tipi di errore:

1. **Compiletime error** (errore a tempo di compilazione), indica che il programma in sè, contiene errori sintattici che vengono trovati subito.
2. **Runtime error** (errore a tempo di esecuzione), indica che dopo la compilazione, durante l'esecuzione del programma (magari per colpa dell'input di un utente, o per l'interazione con altra roba) il programma riscontra un errore.
   In entrambi i casi il programma va in *crash*.

Alternativamente il compilatore potrebbe lamentarsi con un *Warning* dicendo che ci sono metodi, variabili, classi, etc che non sono state usate. Non impedisce il funzionamento del programma, ma una revisione è fortemente suggerita.

Variabili

I principali tipi di variabili sono:

1. String, contiene del testo, come "Hello!";
2. int, contiene dei numeri interi, anche negativi;
3. float, contiene numeri con la virgola come 19.99, anche negativi;
4. char, contiene caratteri singolo, per esempio 'a';
5. boolean, contiene un valore binario che può essere **solo** vero o falso;