

## PROGRAMAZIONE ORIENTATA AGLI OGGETTI - MODULO 5

### Sommario

Cosa apprenderemo in questo modulo? .....	2
Modulo 5 - Polimorfismo .....	2
Ricapitolando ! .....	5

### Cosa apprenderemo in questo modulo?

- Cos'è il polimorfismo?
- A che serve?
- Quali sono i suoi tipi?

### Modulo 5 - Polimorfismo

Il termine polimorfismo è di origine greca e significa "molte forme" (poli = molte, morphos = forme). Nella programmazione ad oggetti questo concetto permette alle classi più specializzate di essere trattate come il tipo più generico da cui derivano, consentendo una gestione dei tipi più omogenea.

Ci sono tre tipi di polimorfismo suddivisi in due macro categorie: Polimorfismo "Universale" (che a sua volta si suddivide in due sottocategorie) e Polimorfismo "Ad Hoc". E' importante notare che non tutti i linguaggi ad oggetti implementano tutti i tipi di polimorfismo:

- **Universale:**

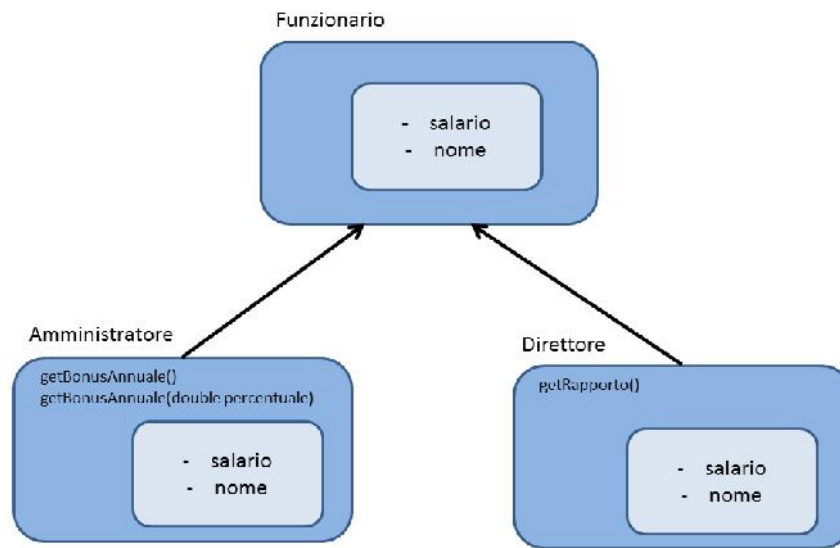
- o **Inclusione:** un puntatore (riferimento) alla classe madre può puntare a una istanza di una classe figlia (che corrisponde al tipo di polimorfismo più diffuso). Vedere "Ridefinizione dei metodi" nei paragrafi seguenti.
- o **Parametrico:** Non limitato all'uso di *template* (in C ++, per esempio) ed ai tipi *generics* (in Java).

- **Ad-Hoc:**

Overloading: due funzioni / metodi con lo stesso nome, ma con firme diverse.

Il concetto alla base del polimorfismo è quello di trattare cose diverse (polimorfiche) in maniera omogenea. In questo modo, è possibile che uno stesso metodo possa trattare, allo stesso maniera, un insieme di classi diverse tra loro.

## PROGRAMAZIONE ORIENTATA AGLI OGGETTI - MODULO 5



```
1. public classe Funzionario {
2.     public double salario;
3.     public String nome;
4. }
5.
6. public classe Amministratore estende Funzionario {
7.     int numeroFunzionari;
8.
9.     public double getBonusAnnuale() {
10.         restituisci salario * 0.3;
11.     }
12.
13.     public double getBonusAnnuale(double percentuale) {
14.         restituisci salario * 0.3 * percentuale;
15.     }
16. }
```

Il polimorfismo è importante perché permette che la semantica di un'interfaccia sia separata dalla sua implementazione. Ma si noti che non è esattamente il polimorfismo che permette la separazione della semantica di un'interfaccia dalla sua effettiva implementazione. Il polimorfismo permette in realtà che diverse implementazioni vengano trattate in maniera omogenea.

Nell'esempio seguente della super classe "Animale", si può applicare il concetto di polimorfismo in questo modo:

```
1. classe astratta Animale {
2.     astratto emettiSuono();
3. }
4.
5. classe Gato estende Animale {
6.     emettiSuono() {
7.         scrivi("miauuuuuuu!!!!");
8.     }
9. }
10.
11. classe Cane estende Animale {
12.     emettiSuono() {
13.         scrivi("auauauauauauau!!");
14.     }
15. }
```

## PROGRAMMAZIONE ORIENTATA AGLI OGGETTI - MODULO 5

Pertanto, il polimorfismo permette a classi ereditate di avere un comportamento simile alla classe madre, ma permette anche alcune specializzazioni dei comportamenti stessi. Quindi, il polimorfismo mantiene una coerenza tra la gerarchia delle classi, ed allo stesso tempo consente anche livelli di specializzazione e generalizzazione (dopo tutto, anche i cani e gatti sono entrambi animali mammiferi, ognuno con le proprie caratteristiche comportamentali).

Inoltre, il polimorfismo permette una maggiore variabilità della definizione dei metodi di una classe, ovvero permette di definire più versioni di un unico metodo a condizione che ciascuna delle versioni riceverà un elenco di parametri diversi (overloading). La decisione di quale metodo deve essere utilizzato viene presa in fase di esecuzione, attraverso un meccanismo chiamato "late binding" (associazione tardiva).

In caso di polimorfismo, è necessario che i metodi abbiano esattamente la stessa firma, utilizzando il meccanismo di ridefinizione (override) dei metodi. Questo meccanismo di ridefinizione non deve essere confuso con il meccanismo di overloading dei metodi. E' importante notare che quando si utilizza il polimorfismo, il comportamento che sarà adottato dal metodo sarà definito solo in fase di esecuzione.

```
1.  Programma Principale {
2.      Animale a;
3.      Gatto g = Gatto();
4.      Cane c = Cane();
5.      a = g;
6.      a.suono(); // miagola
7.      a = c;
8.      a.suono(); // abbaia
9.  }
```

Dettagliando meglio i concetti espressi fin ora abbiamo che:

- **Ridefinizione di metodi:** si verifica quando una classe figlia ridefinisce il comportamento di un metodo di una classe madre. Questo tipo di polimorfismo è anche chiamato di sovrascrittura dei metodi (overriding). Nell'overriding un metodo ridefinito da una classe figlia mantiene la stessa firma del metodo della classe madre, ma cambia la sua implementazione interna.
- **Sovraccarico di metodi:** anche conosciuto come overloading, questo concetto riguarda le molteplici definizioni di uno stesso metodo all'interno di una stessa classe. La differenziazione tra i vari metodi è data attraverso la definizione di firme diverse per ognuna delle "versioni" del metodo, cioè, anche se i metodi hanno lo stesso nome, ogni singola versione avrà un elenco distinto di parametri. Il compilatore non considera esistenza di conflitti in questo caso perché è in grado di identificare il metodo da invocare attraverso la sua lista di parametri
- **Associazione tardiva:** si verifica quando il metodo da richiamare è scelto in fase di esecuzione del programma.

### **Ricapitolando !**

In questa sezione abbiamo imparato il polimorfismo, concetto molto importante nella programmazione orientata agli oggetti. Abbiamo visto che il polimorfismo consente di programmare in maniera più generale invece di programmare nello specifico, ma ha senso parlare di polimorfismo solo quando utilizziamo i concetti di ereditarietà. Se necessario, riesamina il contenuto o fai una ricerca di nuove informazioni da altre fonti!