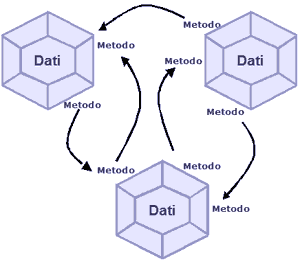
**OOP**

La programmazione orientata agli oggetti (OOP, Object Oriented Programming) è un paradigma di programmazione, che prevede di raggruppare in un'unica entità (la classe) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un "oggetto" software dotato di proprietà (dati) e metodi (procedure) che operano sui dati dell'oggetto stesso.

Il paradigma OOP è basato sul fatto che esiste una serie di oggetti che interagiscono vicendevolmente, scambiandosi messaggi ma mantenendo ognuno il proprio stato ed i propri dati. Graficamente:



In OOP, le caratteristiche di un oggetto vengono denominate **proprietà** e le azioni sono dette **metodi.**

N.B La programmazione orientata agli oggetti è particolarmente adatta a realizzare interfacce grafiche.

**CLASSI**

Le classi sono uno strumento per costruire strutture dati che contengano non solo dati ma anche il codice per gestirli.

Diremo allora che più oggetti software che **hanno le stesse proprietà e gli stessi metodi** possono essere raggruppati in una classe ben definita di oggetti.

Come tutti le costruzioni che permettono di definire le strutture dati, una classe definisce un nuovo tipo di dato.

I membri di una classe sono dati (esattamente come i membri di un record), chiamati attributi, e metodi, ovvero procedure, che operano su un oggetto.

Dunque, una classe rappresenta, sostanzialmente, una **categoria particolare di oggetti** e, dal punto di vista della programmazione, è anche possibile affermare che una classe funge da tipo per un determinato oggetto ad essa appartenente

**OGGETTO**

Un oggetto è una **istanza** di una classe.

L’univocità di ogni istanza viene definita con il termine di **identità** (identity): ogni oggetto ha una propria identità ben distinta da quella di tutte le altre possibili istanze della stessa classe a cui appartiene l’oggetto stesso.

Ogni oggetto possiede tutti gli attributi definiti nella classe, ed essi hanno un valore, che può mutare durante l'esecuzione del programma come quello di qualsiasi variabile.

Si definisce **stato di un oggetto**, l’insieme dei valori delle sue proprietà in un determinato istante di tempo.

Un oggetto occupa memoria, la sua classe definisce come sono organizzati i dati in questa memoria.

L’insieme dei metodi che un oggetto è in grado di eseguire viene definito, invece, **comportamento** (behavior).

Sintatticamente, i metodi di una classe vengono invocati "su" un particolare oggetto, e ricevono come parametro implicito l'oggetto su cui sono stati invocati.(In java per esempio si utilizza **this,** mentre in phyton **self**).

**PROPRIETA’**

Le proprietà rappresentano i dati dell’oggetto, ovvero le informazioni su cui i metodi possono eseguire le loro elaborazioni.

In generale, esistono **tre tipologie di proprietà**:

1. **Gli attributi** rappresentano quelle proprietà che descrivono le caratteristiche peculiari di un oggetto.
2. **I componenti**, invece, sono identificabili in quelle proprietà che sono atte a svolgere delle azioni.
3. Infine, **i peer objects** definiscono delle proprietà che a loro volta sono identificate e definite in altri oggetti.

**MESSAGGI**

i metodi rappresentano le azioni che un oggetto è in grado di eseguire. Ma, in base a quale criterio vengono scatenate ed eseguite tali azioni? Semplicemente rispondendo alle **“sollecitazioni” provenienti da altri oggetti**.

Tali sollecitazioni vengono definiti **messaggi.**

Si è soliti suddividere i messaggi nelle seguenti categorie:

* **Costruttori** I Costruttori costituiscono il momento in cui viene creato un oggetto. Essi devono essere richiamati ogni volta che si vuole creare una nuova istanza di un oggetto appartenente ad una classe e, solitamente, svolgono al loro interno funzioni di inizializzazione.
* **Distruttori** I Distruttori, come si intuisce facilmente dal nome, svolgono la funzione inversa dei costruttori: distruggono un oggetto (ovvero ne eliminano la allocazione dalla memoria).

N.B Molti linguaggi orientati agli oggetti (come ad esempio Java e C#) non forniscono un uso diretto dei distruttori ma prevedono una rimozione automatica dell’oggetto.

* **Accessori (Accessors)** I messaggi di tipo “Accessors” vengono utilizzati per esaminare il contenuto di una proprietà di una classe.
* **Modificatori (Mutators)** I Modificatori rappresentano tutti i messaggi che provocano una modifica nello stato di un oggetto.

Le più comuni relazioni tra classi, in un programma ad Oggetti sono identificabili in tre tipologie:

* **Associazioni (Use Relationship)**: L’Associazione è il tipo di Relazione più intuitiva ed anche più diffuso. In generale, diciamo che una classe A utilizza una classe B se un oggetto della classe A è in grado di inviare dei messaggi ad un oggetto di classe B. Se la navigazione è, invece, possibile in entrambe le direzioni si parlerà di **associazione bidirezionale.**
* **Aggregazioni (Containment Relationship):** La relazione di tipo Aggregazione si basa, invece, sul seguente concetto: Un oggetto di classe A contiene un oggetto di classe B se B è una proprietà (attributo) di A.
* **Specializzazioni (Inheritance Relationship):** La relazione di tipo Specializzazione si basa sul concetto di **ereditarietà**: Un oggetto di classe A deriva da un oggetto di classe B se A è in grado di compiere tutte le azioni che l’oggetto B è in grado di compiere.

**INCAPSULAMENTO**

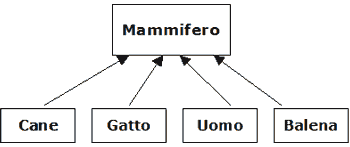
L’incapsulamento è la proprietà per cui i dati che definiscono lo stato interno di un oggetto e i metodi che ne definiscono la logica sono accessibili ai metodi dell'oggetto stesso, mentre non sono visibili ai client. Per alterare lo stato interno dell'oggetto, è necessario invocarne i metodi pubblici, ed è questo lo scopo principale dell'incapsulamento. Infatti, se gestito opportunamente, esso permette di vedere l'oggetto come una black-box, cioè una "scatola nera" con la quale l'interazione avviene solo e solamente tramite i metodi definiti dall'interfaccia. Il punto è dare delle funzionalità agli utenti nascondendo i dettagli legati alla loro implementazione.

**ERIDATERIETA’**

L’ereditarietà costituisce il secondo principio fondamentale della programmazione ad oggetti. In generale, essa rappresenta un meccanismo che consente di creare nuovi oggetti che siano basati su altri già definiti.

Si definisce **oggetto figlio** (child object) quello che eredita tutte o parte delle proprietà e dei metodi definiti nell’**oggetto padre** (parent object).

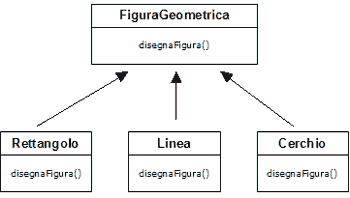
Nel mondo Object Oriented, potremmo riportare tale esempio definendo un oggetto Mammifero che inglobi tutte le caratteristiche comuni ad ogni mammifero. Da esso, poi, deriverebbero gli altri child object: Cane, Gatto, Uomo, Balena, etc.



**POLIFORMISMO**

Letteralmente, la parola polimorfismo indica la possibilità per uno stesso oggetto di **assumere più forme**.

Riferendoci ad un sistema software ad oggetti, il polimorfismo indicherà l’attitudine di un oggetto a mostrare più implementazioni per una singola funzionalità.



Con l’utilizzo del polimorfismo, il sistema è in grado di capire autonomamente quale figura geometrica debba essere disegnata ed invocarne direttamente il metodo disegnaFigura appartenente alla classe figlia coinvolta.

**ASTRAZIONE DEI DATI**

Più in particolare, si può affermare che l’**Astrazione dei Dati** viene utilizzata per gestire al meglio la complessità di un programma, ovvero viene applicata per decomporre sistemi software complessi in componenti più piccoli e semplici che possono essere gestiti con maggiore facilità ed efficienza.

**Classe Astratta**, ovvero una classe che rappresenta, fondamentalmente, un modello per ottenere delle classi derivate più specifiche e più dettagliate. In una classe astratta, solitamente sono contenuti pochi metodi, I metodi appartenenti a questa ultima tipologia (e che sono definiti nella classe astratta) prendono il nome di **Metodi Astratti**.