

## 3 - Introduzione alla programmazione ad oggetti

La programmazione ad oggetti cerca di creare una rappresentazione della realtà. Nel mondo reale qualsiasi cosa è un oggetto, e ogni oggetto ha uno **stato** e un **funzionamento**

### Oggetti

Alcuni esempi che troviamo nel mondo reale sono:



#### *Stato*

- Marcia corrente
- Velocità di marcia

#### *Funzionamento*

- Cambia marcia
- Frena



#### *Stato*

- Accesa
- Spenta

#### *Funzionamento*

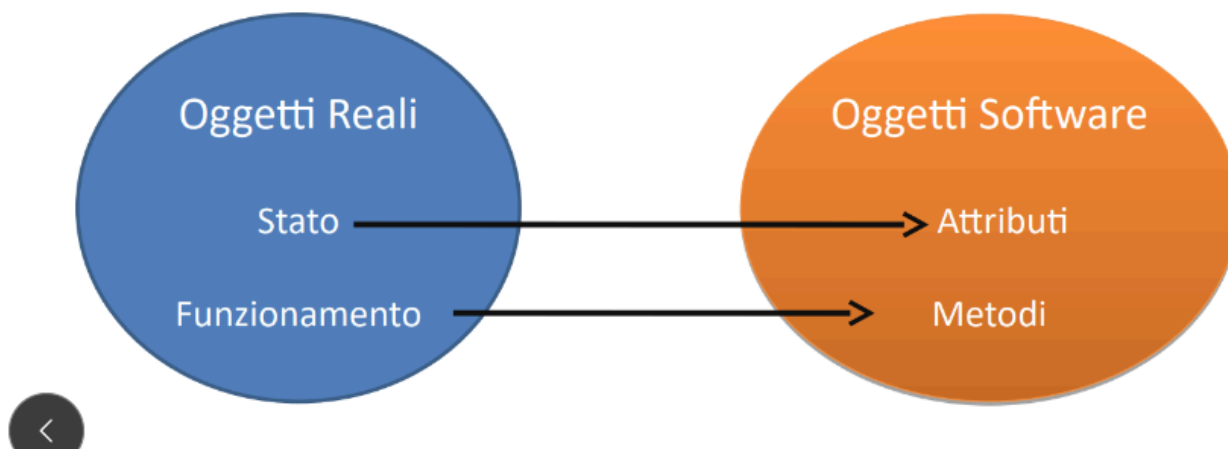
- Accendi
- Spegni

Come si può vedere non ci sono superficialità, ma **dettagli utili soltanto allo scopo che si vuole raggiungere**

### Oggetti software

Sostanzialmente gli **oggetti software** sono una mappatura di quello che si trova nel mondo reale.

Lo stato viene memorizzato negli **attributi** o **variabili**, mentre il loro funzionamento si ritrova nei **metodi** o **funzioni**



Lo stato è dato dai **valori che assumono i suoi attributi** (Es. la bicicletta ha marcia corrente=5, velocità=30 km/h)

Il funzionamento è dato dai suoi metodi (es. bicicletta -> cambia marcia), gli stessi metodi **possono modificare** lo stato dell'oggetto agendo sui suoi attributi

## Vantaggi

Gli oggetti hanno diversi vantaggi:

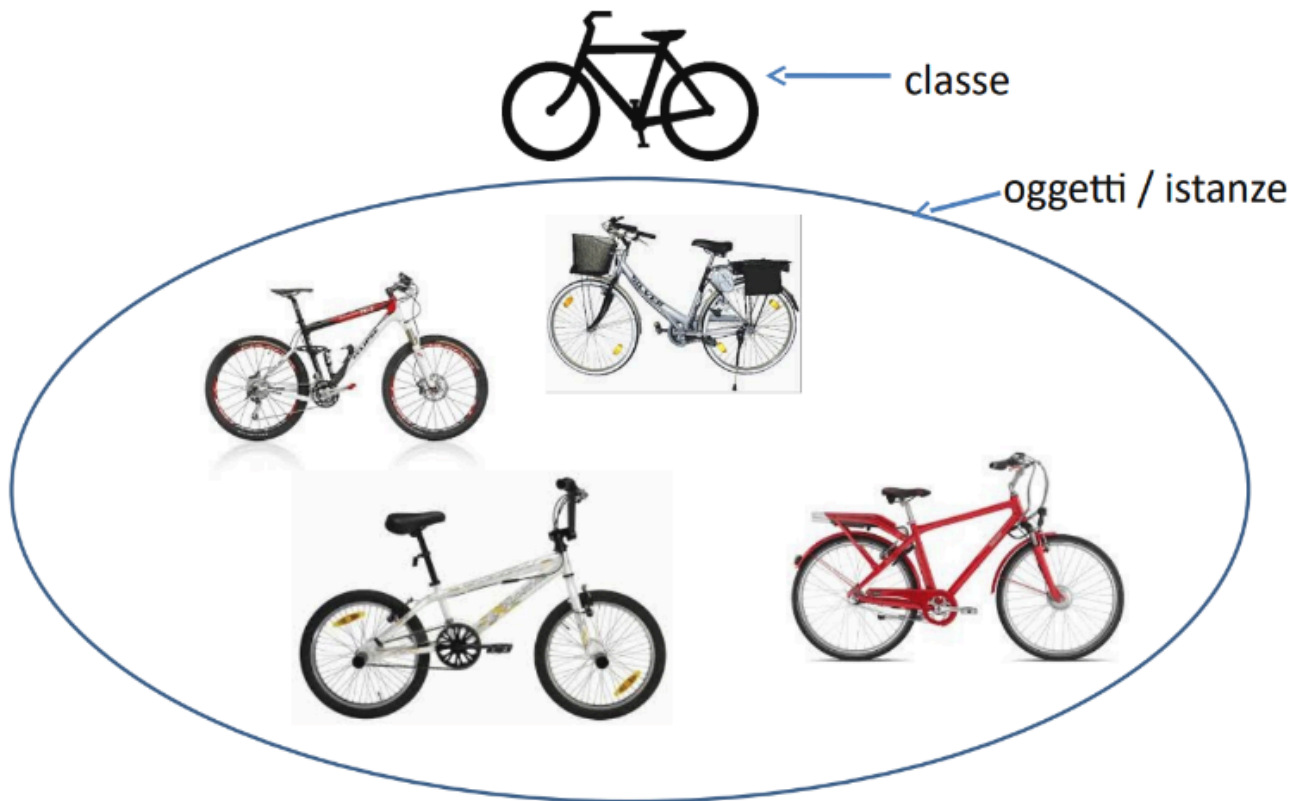
- **Modularità:** ogni oggetto è indipendente dagli altri, il suo codice può essere gestito separatamente (avendo vita propria)
- **Information hiding:** i dettagli implementativi di ogni oggetto sono nascosti dagli altri oggetti, che interagiscono solo tramite i suoi metodi
- **Riutilizzo:** oggetti scritti da altri possono facilmente essere ri-utilizzati
- **Sostituzione:** ogni oggetto può essere facilmente essere sostituito

## Classe

Nel mondo reale molti oggetti condividono delle caratteristiche simili, per questo si raggruppano sia per queste sia per funzionamento simile, le riconosciamo come **classi** (che abbiamo visto nei vari linguaggi)

(Es. tutte le biciclette che abbiamo posseduto sono delle biciclette)

# Cosa è una classe?



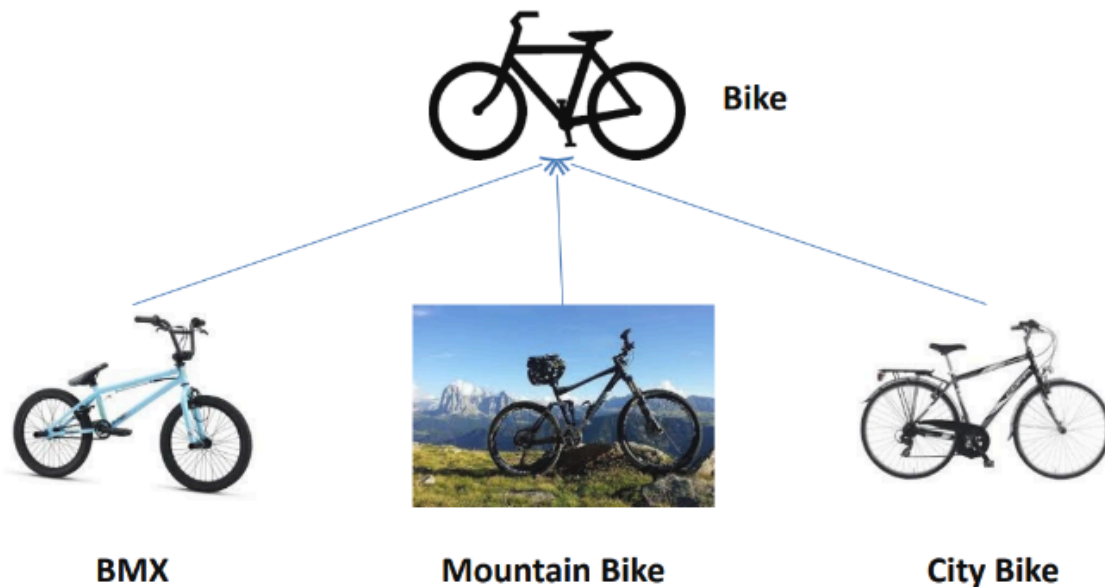
Riconosciamo gli **oggetti** o **istanze** come specifiche di classi

## Ereditarietà

Alcune classi di oggetti hanno delle caratteristiche in comune, ma si possono avere caratteristiche in più o comportamenti specifici diversi dalla classe originale, **l'ereditarietà** permette quindi ad una classe di ereditare attributi e metodi di un'altra classe

Riprendendo l'esempio delle biciclette:

# Ereditarietà



**BMX, Mountain Bike e City Bike ereditano tutti gli attributi e i metodi di Bike. In più, potranno avere altri metodi e attributi specifici**

L'ereditarietà è un meccanismo potente per rappresentare una gerarchia tra classi

## Programmazione Object Oriented

Nella programmazione imperativa è possibile accedere alle variabili globali da qualsiasi parte del programma;

In questa maniera andiamo a perdere **l'information-hiding**, nei programmi molto grandi diventa praticamente ingestibile, i moduli non possono essere sviluppati indipendentemente (alcune funzioni potrebbero leggere o scrivere una variabile globale in contemporanea ad altre)

La soluzione sarebbe quella di incapsulare ogni variabile globale in un modulo insieme ad un gruppo di operatori che possono accedere a tali variabili in modo che gli altri moduli possono accedere indirettamente alle variabili solo per mezzo di questi operatori

## I metodi

Gli oggetti sono costituiti da **dati privati** e **operazioni permesse su tali dati**, tra di loro comunicano tramite passaggio di **messaggi** (chiamata a procedura, detta metodo, che appartiene all'oggetto)

Nei linguaggi ad oggetti come C++ nelle definizioni di classi si avranno metodi accessibili da qualunque punto nel programma e altri che potranno essere accessibili solo alla classe

stessa (in C questo non è possibile con la struct)

```
typedef int numSides;
typedef int sideLength;
#include <math.h>

class Square {
public:                                // metodi accessibili da qualsiasi punto
    Square(sideLength side): s(side), n(4) {};           // costruttore
    sideLength getSide() { return s;}
    sideLength perimeter() { return n * s;}
    double area() { return (double) s * s;}
private:                              // metodi accessibili da oggetti Square
    sideLength s;
    const numSides n;
};

class Triangle {
public:
    Triangle(sideLength side): s(side), n(3) {};        // costruttore
    sideLength getSide() { return s;}
    sideLength perimeter() { return n * s;}
    double area() { return sqrt(3.0) *getSide() * getSide() / 4.0;}
private:
    sideLength s;
    const numSides n;
};
```

Il passaggio di messaggi permette la comunicazione:

- fra **oggetti**
- fra **oggetti** e **programma client**

Quando viene inviato un messaggio ad un oggetto viene selezionato un metodo fra quelli disponibili per rispondere (ricordando che anche i metodi possono essere non visibili o **privati**)