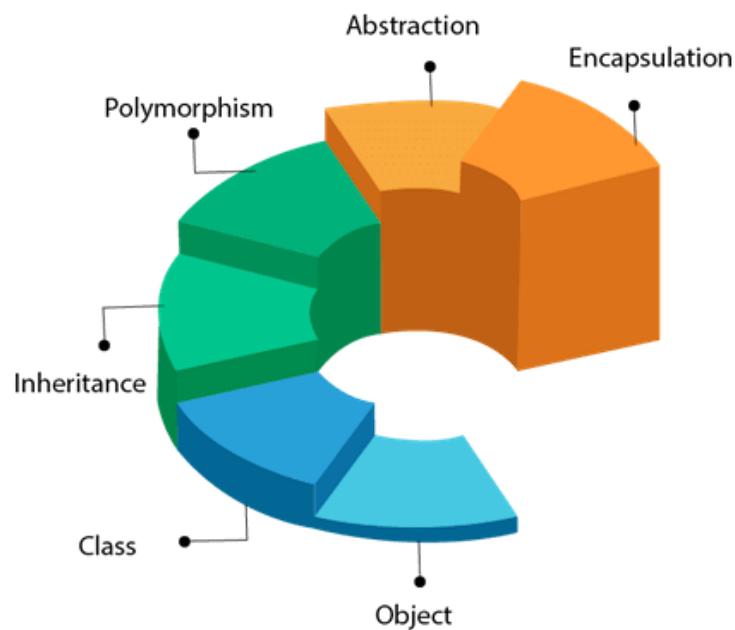


Object Oriented Programming



Object Oriented Programming

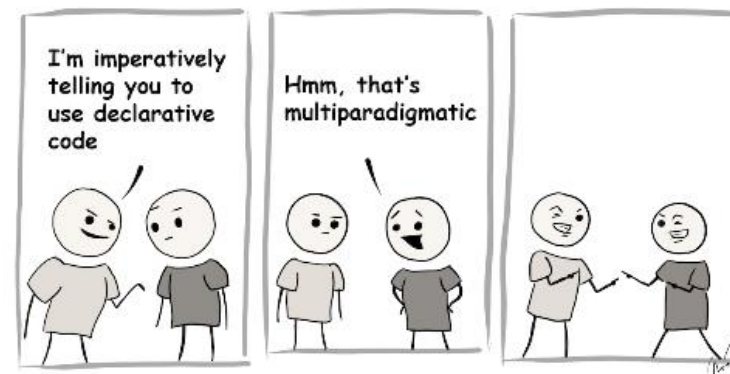
- paradigmi di programmazione
 - OOP
- oggetto
- classe
 - attributi
 - costruttore
 - metodi

SUMMARY



paradigma di programmazione

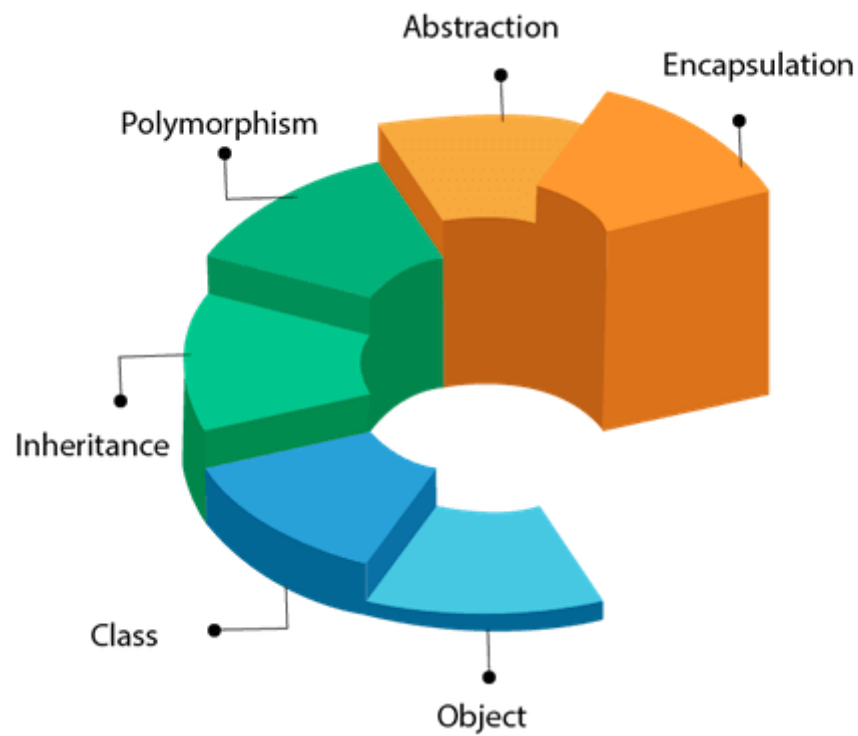
- il ***paradigma di programmazione*** definisce il modo in cui il programmatore concepisce il programma
- i vari paradigmi si ***differenziano***
- per le ***astrazioni*** usate per rappresentare gli elementi di un programma (funzioni, oggetti, variabili ...)
- per i ***procedimenti*** usati per l'elaborazione dei dati (assegnamento, iterazione, gestione del flusso dei dati ...)



paradigmi

- paradigma ***imperativo***
 - programmazione procedurale ('60)
 - programmazione strutturata ('60-'70)
- programmazione orientata agli ***eventi***
 - *interfacce grafiche*
- programmazione ***logica***
 - *intelligenza artificiale*
- programmazione ***funzionale***
 - *applicazioni matematiche e scientifiche*
- programmazione ***orientata agli oggetti***

programmazione orientata agli oggetti



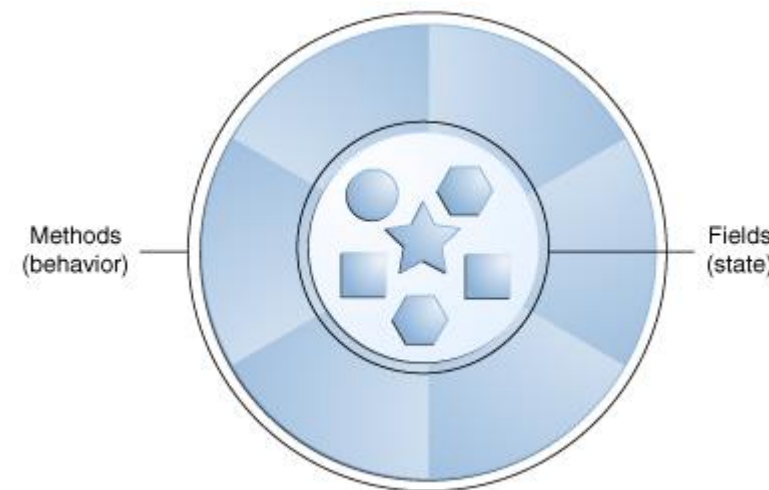
Object Oriented Programming

- la *programmazione orientata agli oggetti* (**OOP**, **O**bject **O**riented **P**rogramming) permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi



oggetto

- analisi della realtà e definizione del ***dominio applicativo***
 - evidenziare informazioni essenziali eliminando quelle non significative per il problema
- un ***oggetto*** rappresenta un oggetto fisico o un concetto del dominio
 - memorizza il suo ***stato*** interno in campi privati (attributi dell'oggetto)
 - concetto di ***incapsulamento*** (black box)
 - offre un insieme di ***servizi***, come ***metodi*** pubblici (comportamenti dell'oggetto)
- realizza un ***tipo di dato astratto***
 - (ADT - *Abstract Data Type*)



esempio: automobile di Mario



- nome dell'oggetto: «AutoDiMario»
- classe: ***Automobile***
- ***attributi***
 - velocità_massima → 220
 - velocità_attuale → 120
 - tipo_freni → tamburo
- ***metodi***
 - accelera(n) → aumenta velocità nKm/h
 - frena()

classi e oggetti

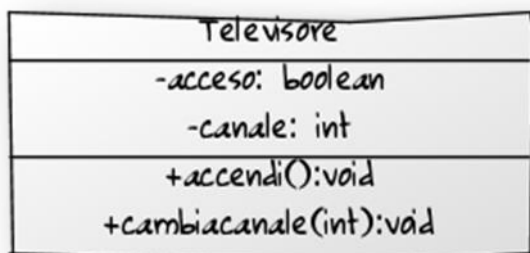
- ogni oggetto ha una **classe** di origine (*è istanziato da una classe*)
- la classe definisce la stessa **forma iniziale** (campi e metodi) a tutti i suoi oggetti
- ma ogni oggetto
 - ha la sua **identità**
 - ha uno stato e una locazione in memoria distinti da quelli di altri oggetti
 - sia istanze di classi diverse che della stessa classe



classi e tipi di dato

- una classe è a tutti gli effetti un ***tipo di dato*** (come gli interi e le stringhe e ogni altro tipo già definito)
- un tipo di dato è definito dall'insieme di ***valori*** e dall'insieme delle ***operazioni*** che si possono effettuare su questi valori
- nella programmazione orientata agli oggetti, è quindi possibile sia utilizzare tipi di dato esistenti, sia definirne di nuovi tramite le classi
- i nuovi tipi di dato si definiscono ***ADT*** (*Abstract Data Type*)

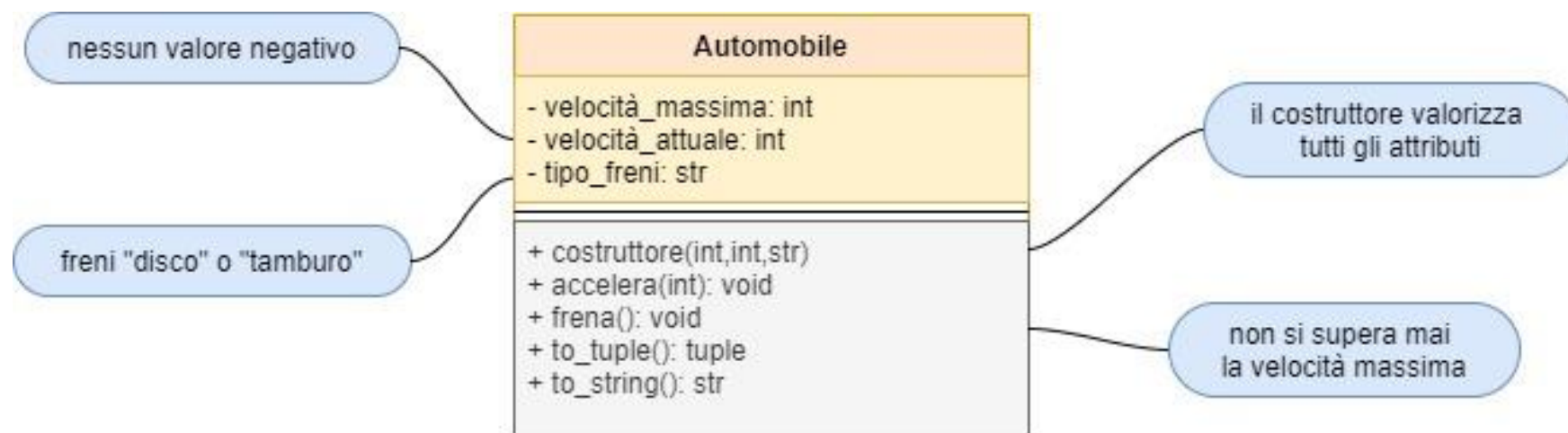
diagramma delle classi



- la prima sezione contiene il ***nome*** della classe
- la seconda sezione definisce i suoi ***attributi***
- la terza i ***metodi***, le operazioni che si possono compiere sull'oggetto

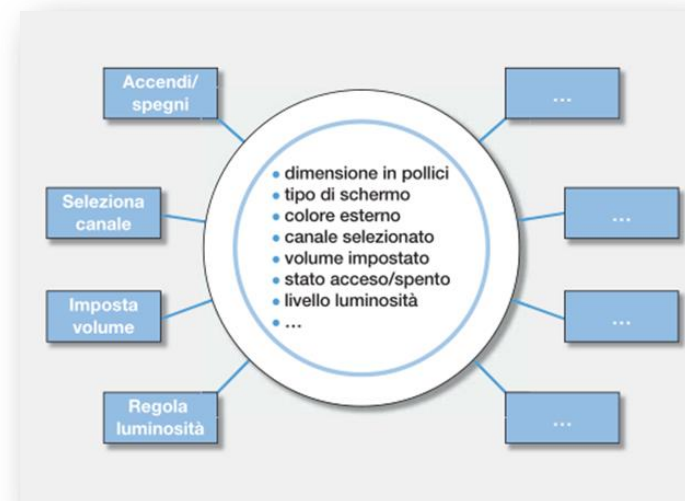
<http://www.draw.io>

classe Automobile



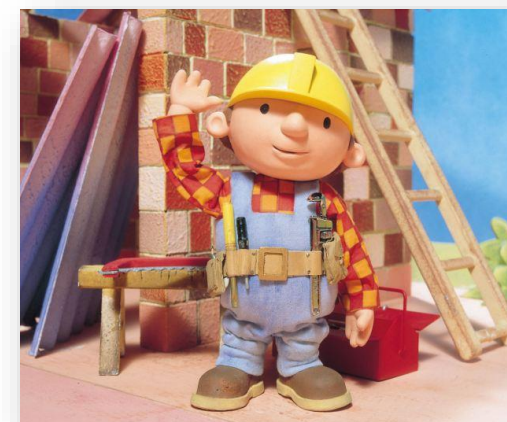
incapsulamento (information hiding)

- ***nascondere*** il funzionamento interno (la struttura interna)
- fornire un'***interfaccia*** esterna che permetta l'utilizzo senza conoscere la struttura interna



Python – self - __init__

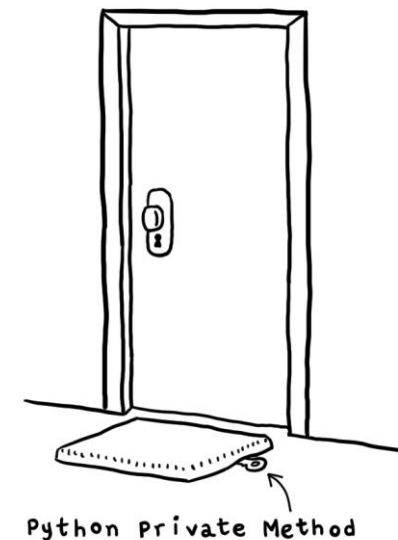
- costruzione di oggetti (*istanziamento*)
- **__init__**: metodo *costruttore*
- eseguito *automaticamente* alla creazione di un oggetto
 - *instantiation is initialization*
- **self**: primo parametro di tutti i metodi
 - non bisogna passare un valore esplicito
 - rappresenta l'oggetto di cui si chiama il metodo
 - permette ai metodi di accedere ai campi



Python definizione della classe

```
class Automobile:

    def __init__(self, vm: int, va: int, freno: str):
        '''
        vm velocità massima
        va velocità attuale
        freno tipo freno
        '''
        self._vm = max(vm, 0)
        self._va = max(va, 0)
        if freno == 'tamburo':
            self._freno = 'tamburo'
        else:
            self._freno = 'disco'
```



Daniel Stori {turnoff.us}

Python metodi

```
def accelera(self, km: int):
    ''' aumenta la velocità di km chilometri orari'''
    if km <= 0:
        return
    self._va += km
    if self._va > self._vm:
        self._va = self._vm
```

```
def frena(self):
    ''' dipende dal tipo dei freni '''
    if self._freno == 'disco':
        self._va -= 20
    else:
        self._va -= 10
    self._va = max(self._va, 0)
```

```
def to_tuple(self) -> tuple:
    return self._vm, self._va, self._freno

def to_string(self) -> str:
    s = 'velocità massima: ' + str(self._vm)
    s += ' velocità attuale: ' + str(self._va)
    s += ' freni a ' + self._freno
    return s
```


oggetti e classi



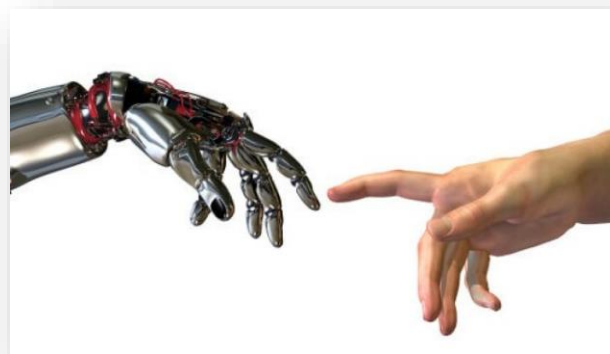
- da una ***classe*** possono essere istanziati (creati) ***più oggetti***
- tutti con gli ***stessi attributi***
 - ma ognuno ha una sua ***valorizzazione*** degli attributi
- tutti hanno lo stesso ***comportamento***
 - rispondono ai messaggi attivando i ***metodi*** della classe

gli oggetti

- gli oggetti sono le **entità** di un programma che **interagiscono** tra loro per raggiungere un obiettivo
- vengono **creati** (istanziati) in fase di esecuzione
- ognuno di essi fa parte di una categoria (una **classe**)
- ogni classe può creare **più oggetti**, ognuno dei quali, pur essendo dello stesso tipo, è **distinto** dagli altri
- un oggetto è una **istanza** di una classe

creazione di un oggetto

- per creare un oggetto si effettua una ***istanziatura*** di una classe
- in questa fase viene riservato uno spazio di memoria per conservare i valori degli attributi dell'oggetto che si sta creando (per mantenere memorizzato lo stato dell'oggetto)



oggetti

```
from automobile import Automobile

mario = Automobile(220,120,'tamburo')
print(mario.to_tuple())
print(mario.to_string())
mario.accelera(5)
print(mario.to_string())
mario.accelera(500)
print(mario.to_string())
for i in range(25):
    mario.frena()
    print(mario.to_string())
```

definizioni

- ***campi***: memorizzano i ***dati caratteristici*** di una istanza
 - ogni pallina ha la sua posizione (x, y) e la sua direzione (dx, dy)
- ***parametri***: ***passano*** altri ***valori*** ad un metodo
 - se alcuni dati necessari non sono nei campi
- ***variabili locali***: memorizzano ***risultati parziali***
 - generati durante l'elaborazione del metodo
 - nomi ***cancellati*** dopo l'uscita dal metodo
- ***variabili globali***: definite ***fuori*** da tutte le funzioni
 - usare sono se strettamente necessario
 - meglio avere qualche parametro in più, per le funzioni

in Python tutto è un oggetto

```
n = 15                                # viene istanziato l'oggetto n della classe int
print('tipo di n ', type(n))          # type funzione
print('rappresentazione binaria di n ', bin(n)) # bin funzione
print('cifre binarie di n ', n.bit_length()) # bit_length metodo
```

```
tipo di n <class 'int'>
rappresentazione binaria di n 0b1111
cifre binarie di n 4
```

```
f = 3.25                              # f oggetto della classe float
print('tipo di f ', type(f))          # type funzione
print(f.as_integer_ratio(), 'frazione generatrice di', f) # as_integer_ratio metodo
```

```
tipo di f <class 'float'>
(13, 4) frazione generatrice di 3.25
```

Classi in Java

```
public class Automobile {  
  
    private int vm;           // velocità massima  
    private int va;           // velocità attuale  
    private String freno;     // tipo freno  
  
    public Automobile(int vm, int va, String freno) {  
        this.vm = Math.max(vm, 0);  
        this.va = Math.max(va, 0);  
        if (freno.equals("tamburo"))  
            this.freno = "tamburo";  
        else  
            this.freno = "disco";  
    }  
}
```

```
public void accelera(int km){ // aumenta la velocità di km chilometri orari
    if (km <= 0)
        return;
    this.va += km;
    if (this.va > this.vm)
        this.va = this.vm;
}

public void frena(){ // dipende dal tipo dei freni
    if (this.freno.equals("disco"))
        this.va -= 20;
    else
        this.va -= 10;
    this.va = Math.max(this.va, 0);
}

public String toString(){
    String s = "velocità massima: " + this.vm;
    s += " velocità attuale: " + this.va;
    s += " freni a " + this.freno;
    return s;
}

}
```



```
public class AutomobileMain {  
  
    public static void main(String[] args) {  
        Automobile mario = new Automobile(220,120,"tamburo");  
        System.out.println(mario.toString());  
        mario.accelera(5);  
        System.out.println(mario);  
        mario.accelera(500);  
        System.out.println(mario);  
        for (int i = 0; i < 25; i++) {  
            mario.frena();  
            System.out.println(mario);  
        }  
    }  
}
```