La Programmazione Orientata agli Oggetti

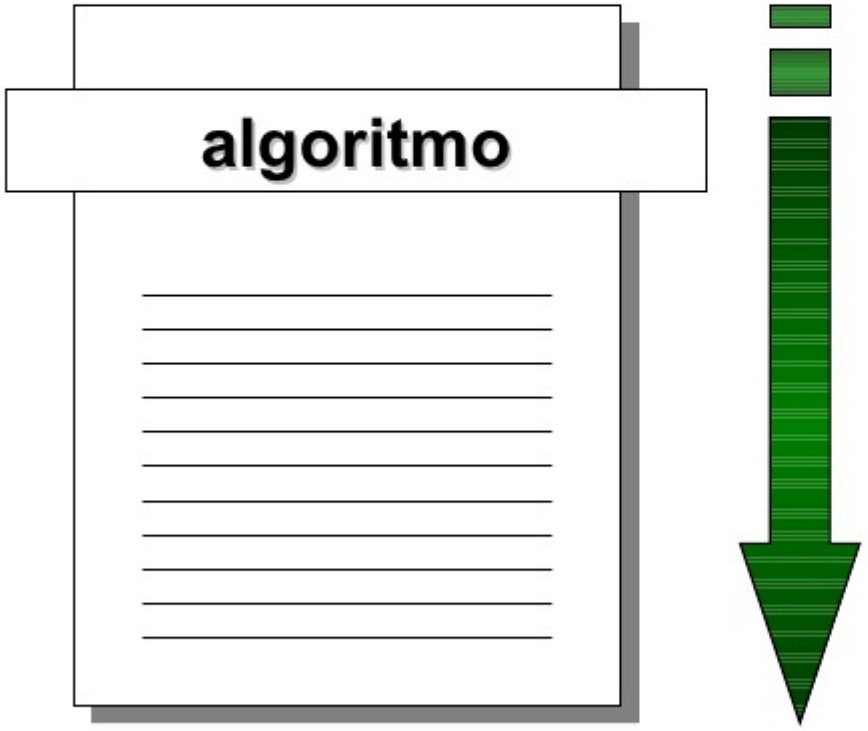
(O.O.P.)

Object-Oriented Programming

Sappiamo che la **Programmazione**

imperativa

si basa sul concetto di :



Insieme di ***istruzioni***

che a partire dai
dati di input

permettono di ottenere
dei ***risultati*** in output

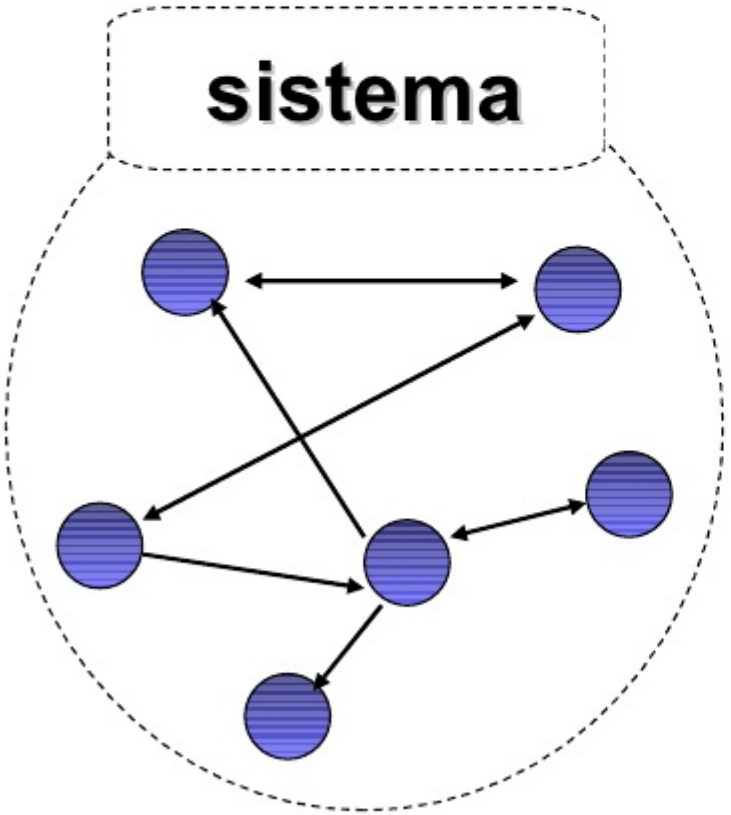
Come si programma :

Applicando i principi
della ***programmazione strutturata:***

- *metodologia di analisi* ***Top-Down***
- ***Strutture*** *di controllo*
  - *-sequenza*
  - *-selezione*
  - *-iterazione*

La **Programmazione**
orientata agli oggetti

si fonda invece sul concetto di :



sistema *è un'insieme di*

entità

interagenti

(*oggetti*)

in cui ***ogni*** *componente è caratterizzato da*

proprietà (*attributi*)

e

azioni (*metodi*)

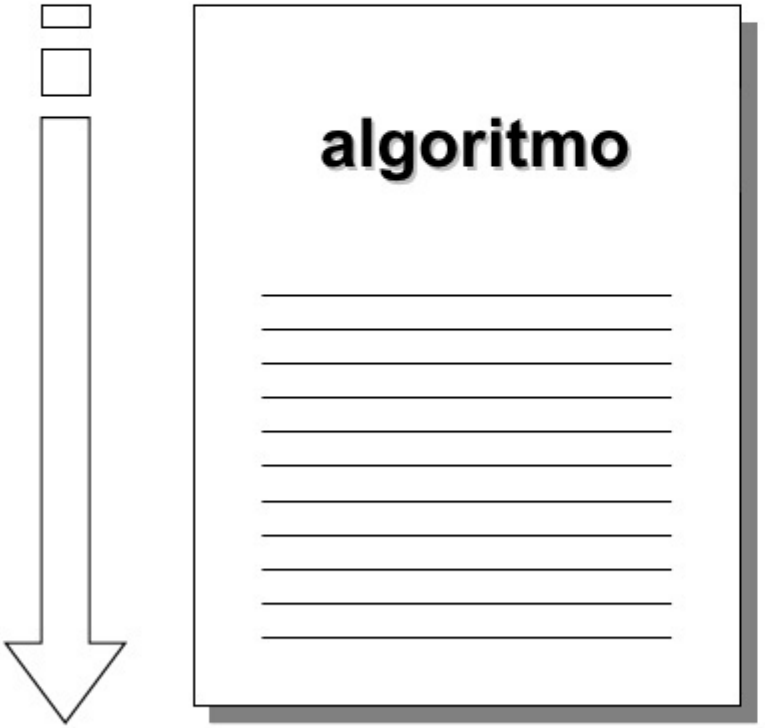
Come si programma in O.O.P. ?

… applicando i principi
della ***programmazione strutturata***

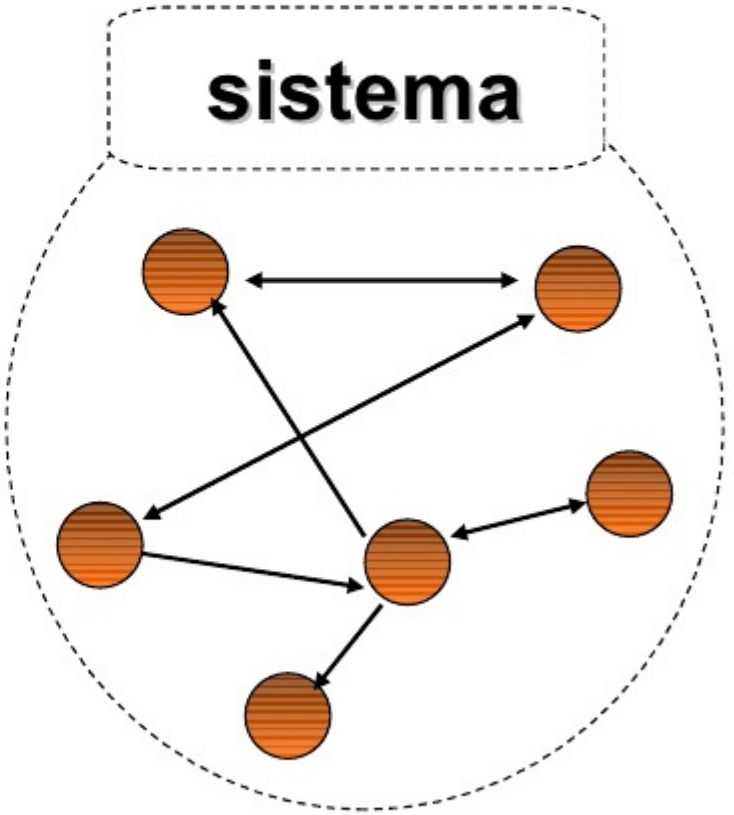
cioè …

- *metodologia di analisi Top-Down*
- *strutture di controllo*
  - *-sequenza*
  - *-selezione*
  - *-iterazione*

Quindi quello che cambia è solamente la visione del problema...



sequenza



interazione

Non a caso si parla di programmazione orientata agli oggetti

Per cui è necessario individuare :

- le ***entità*** *(gli oggetti)* presenti all'interno del sistema
- le ***modalità*** di interazione reciproca

Per la risoluzione di un **Problema**

Per la gestione di un **Sistema**

ricapitolando...

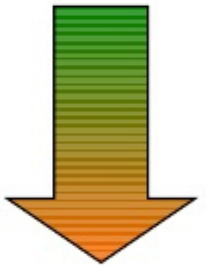
Programmazione
imperativa

Problema
complesso

scomposizione in
procedure

Programmazione
orientata agli oggetti

Sistema
complesso



scomposizione in
entità interagenti
(oggetti)

Oggetti

e

Classi di oggetti

(secondo il formalismo UML)

oggetto

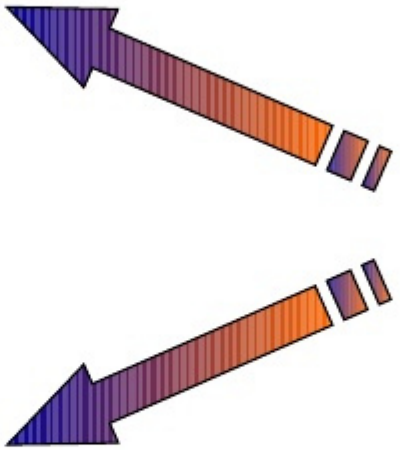
Ford Focus 1.4

Velocità = **108**
colore = **nero**
livello carburante = **15,6**
posizione marcia = **5**

oggetto

Fiat Punto 60

Velocità = **65**
colore = **verde**
livello carburante = **32,4**
posizione marcia = **3**



Classe

automobile
velocità colore livello carburante posizione marcia
avvia accelera gira fermati cambia marcia rifornisci

Oggetto = ***istanza*** di una **Classe**

Diagramma delle Classi

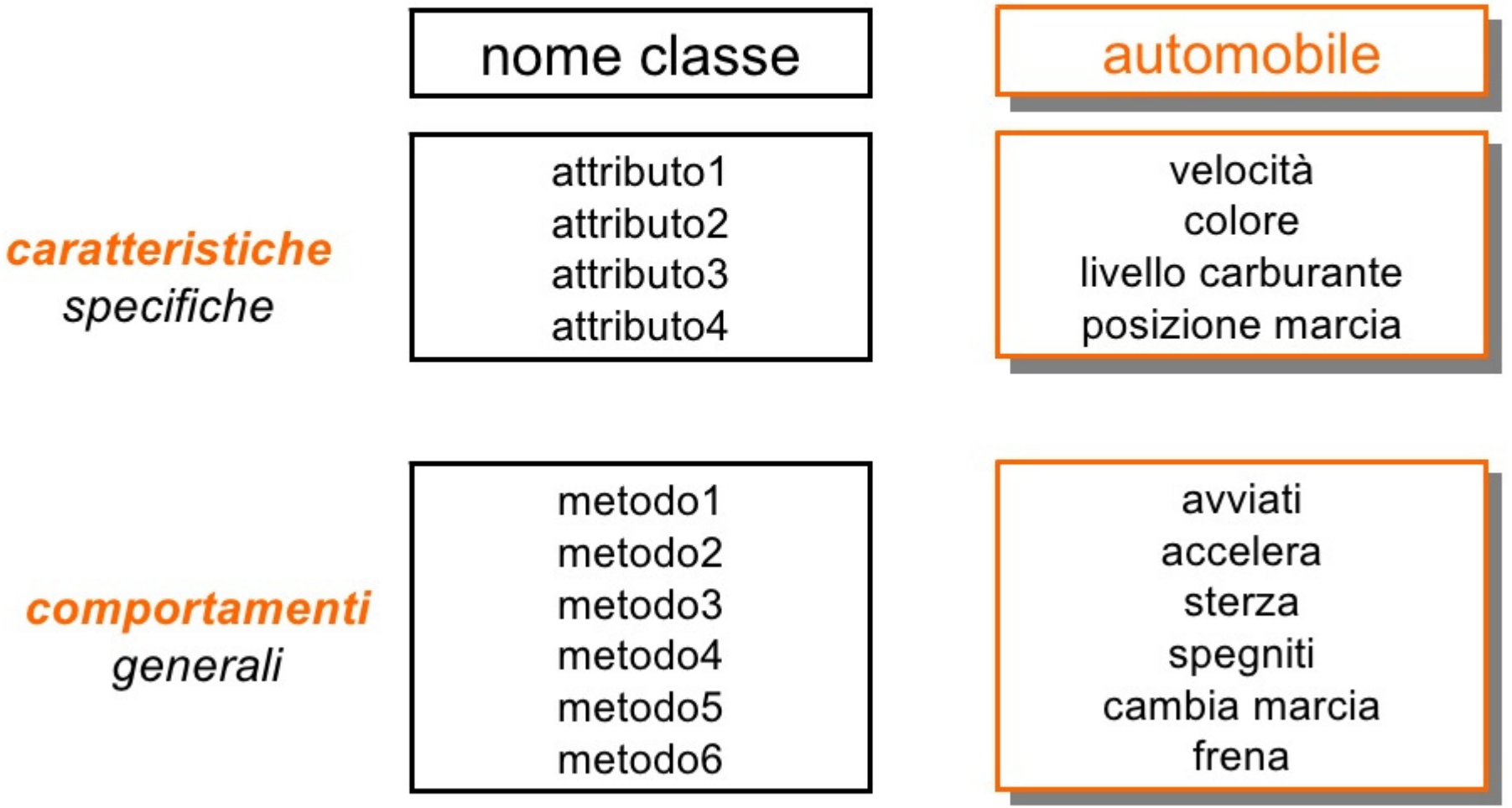
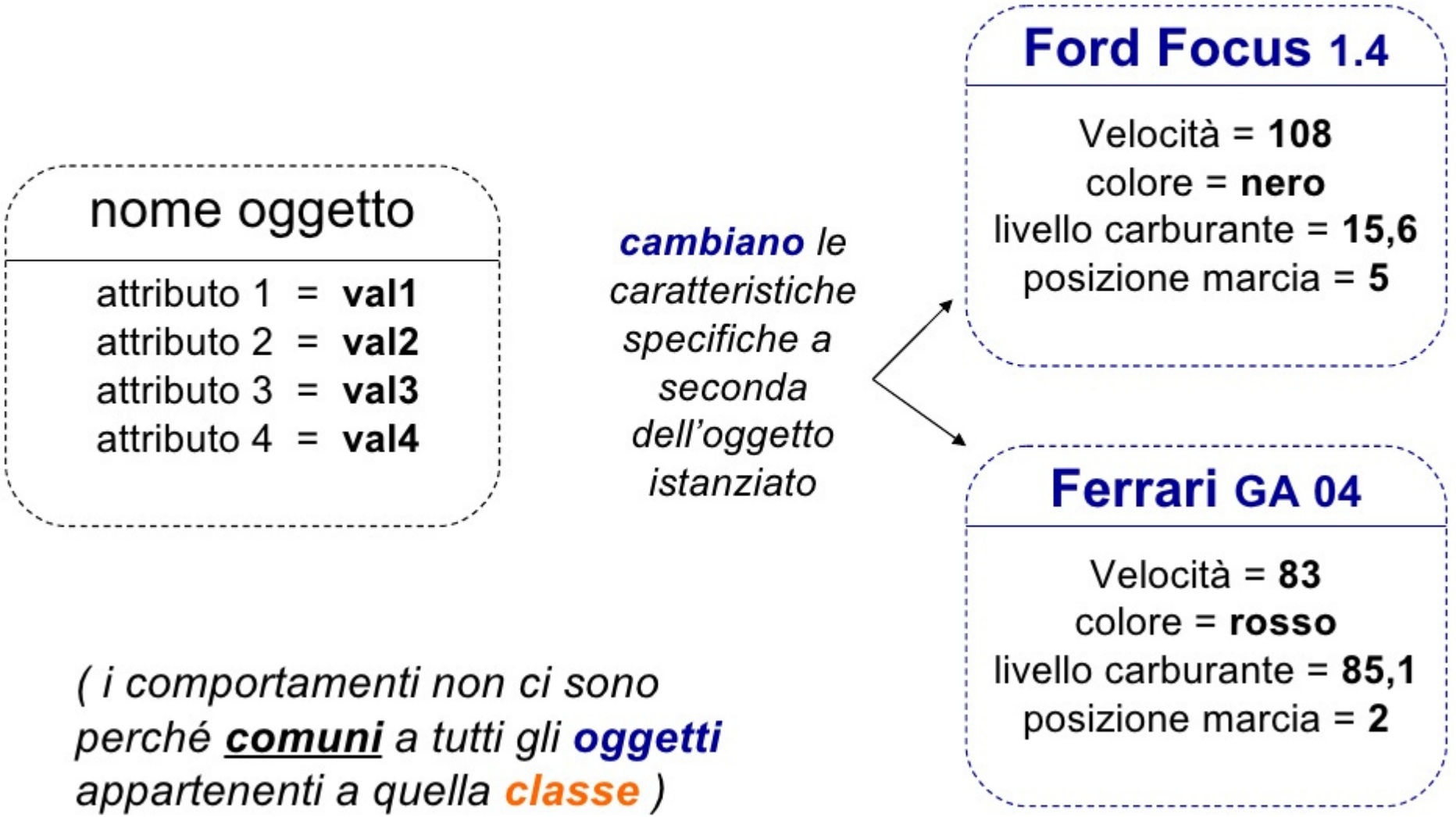


Diagramma degli **Oggetti**



*(i comportamenti non ci sono perché **comuni** a tutti gli **oggetti** appartenenti a quella **classe**)*

Classi

- **Ereditarietà** *e* **Gerarchia** *di ereditarietà*
- **Tipi** *di ereditarietà*
- **Polimorfismo**

Ereditarietà

Classe

automobile
velocità colore livello carburante posizione marcia
avvia accelera sterza frena cambia marcia accendi luci

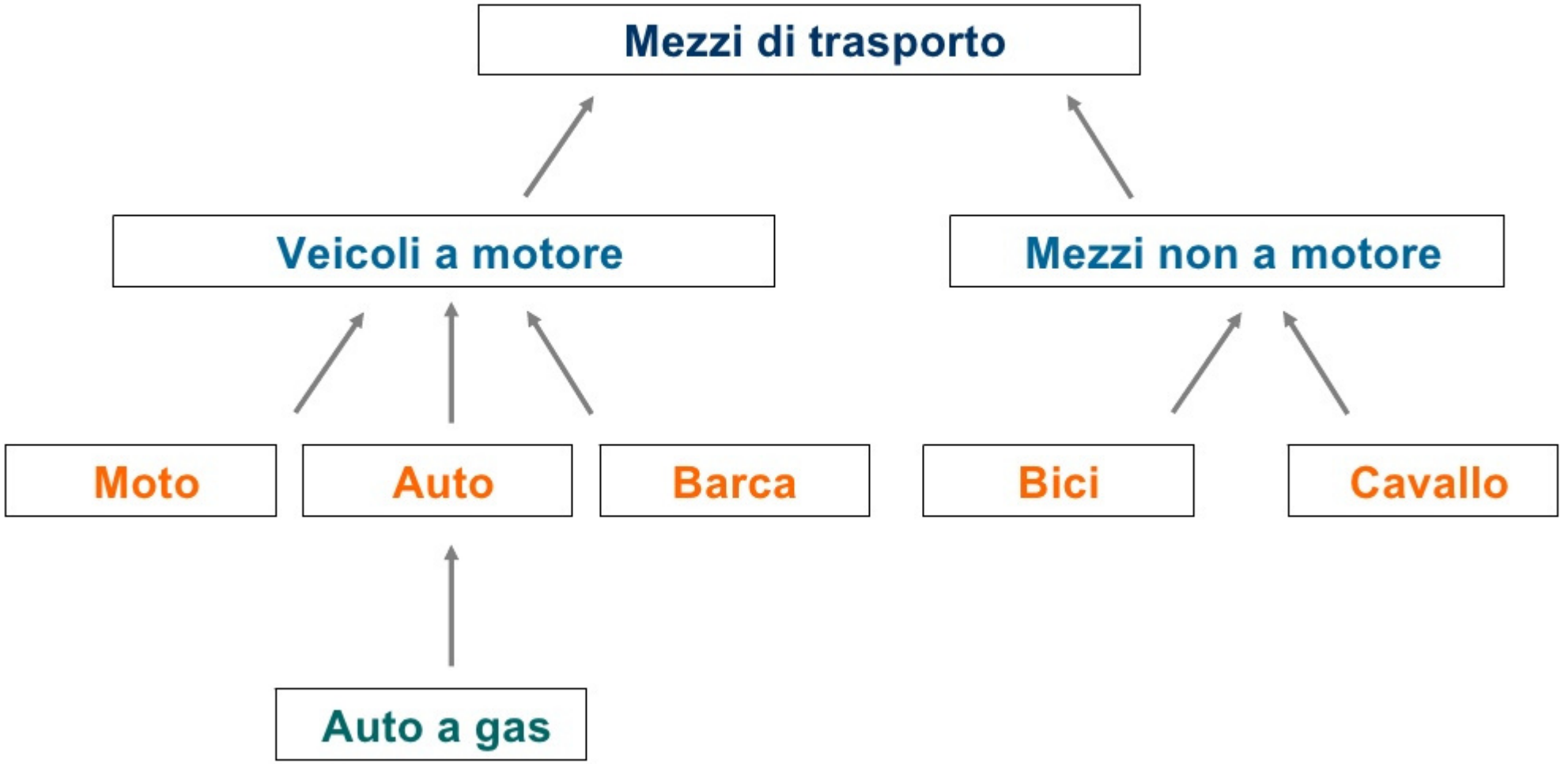
classe genitrice

costruisco
nuove classi
partendo da
quelle già
esistenti

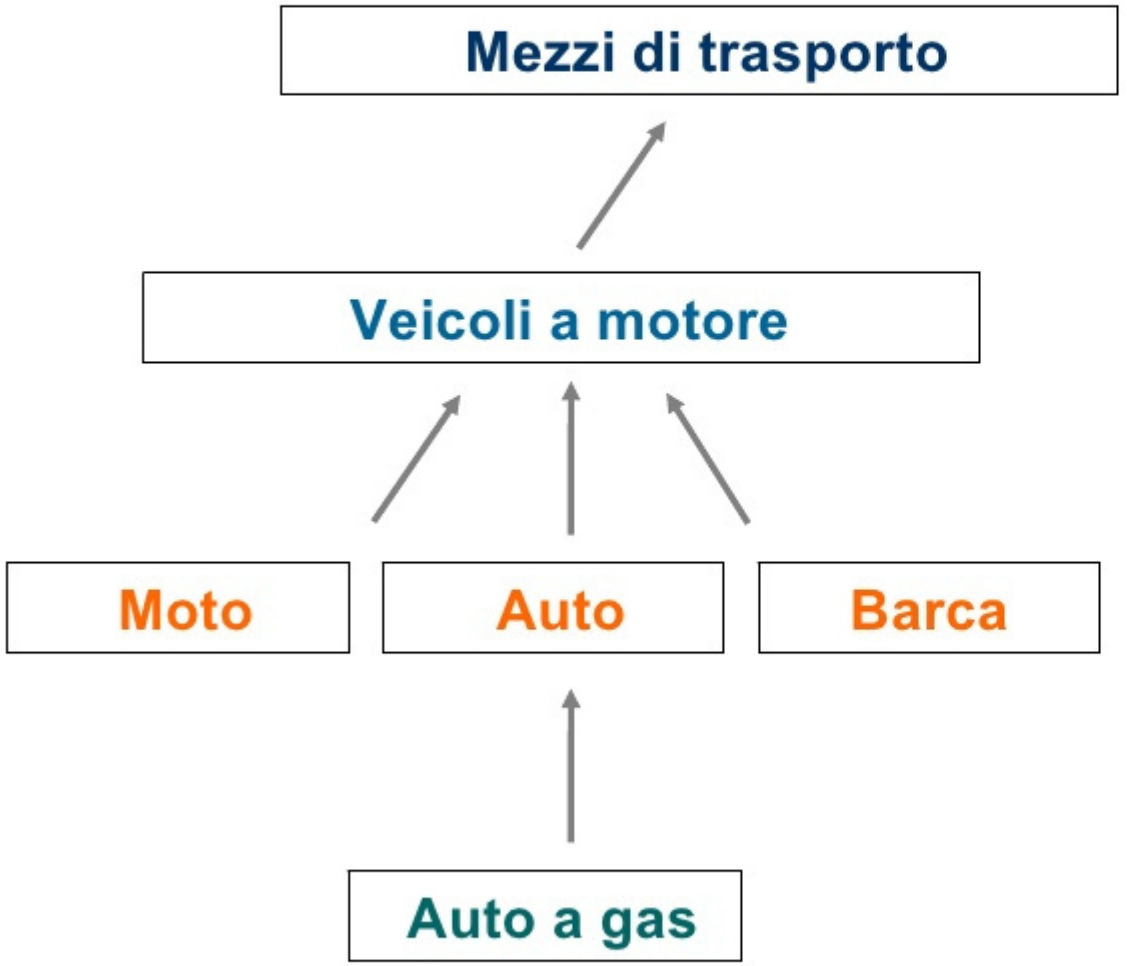
Nuova Classe

automobile a gas
velocità colore livello carburante posizione marcia ***tipo di gas***
avvia accelera sterza frena cambia marcia accendi luci ***scambia gas-benz.***

Gerarchia di ereditarietà



Gerarchia *di ereditarietà*



Sopraclasse *(o Superclasse)*

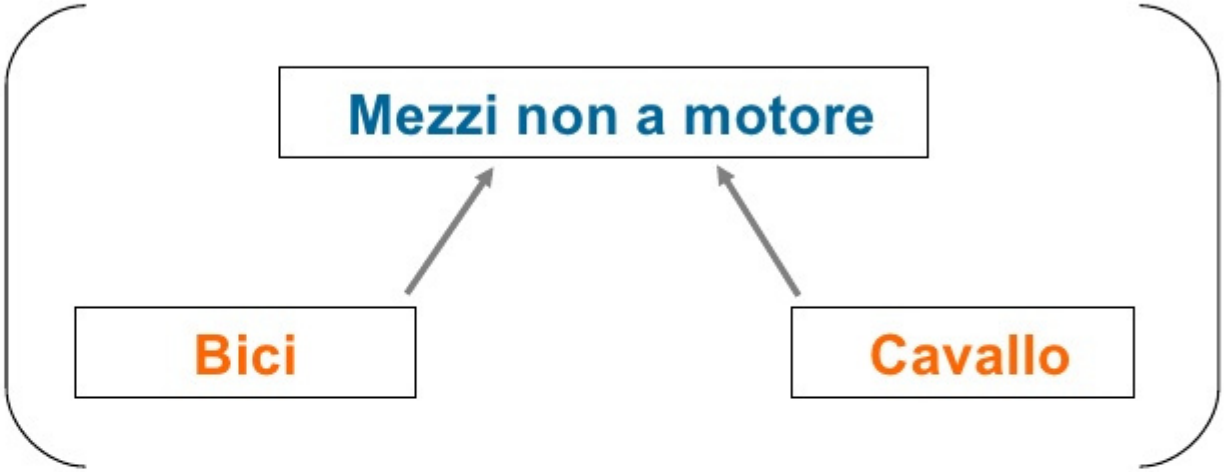
Sottoclasse *(o Superclasse)*

Sottoclasse *(o Superclasse)*

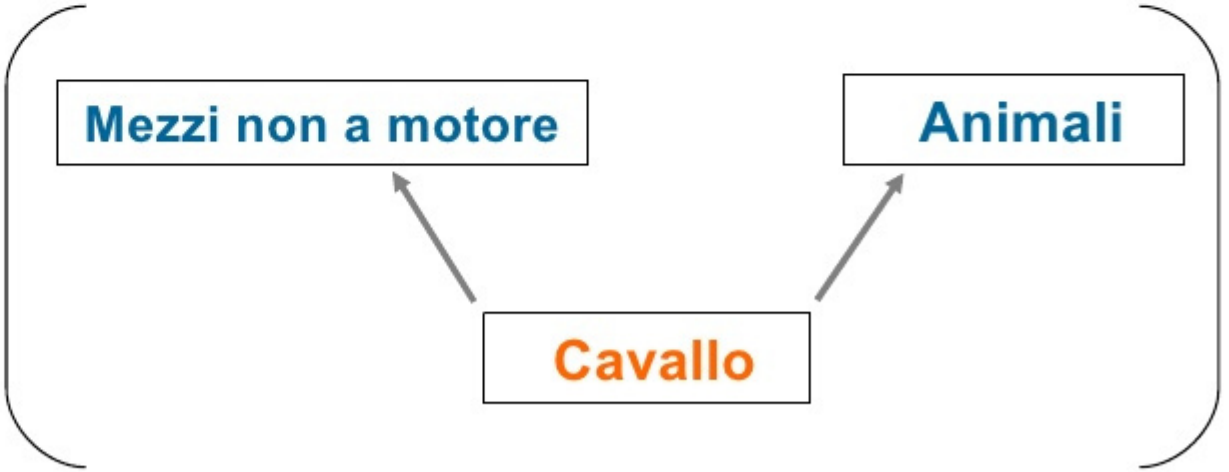
Sottoclasse

Tipi *di ereditarietà*

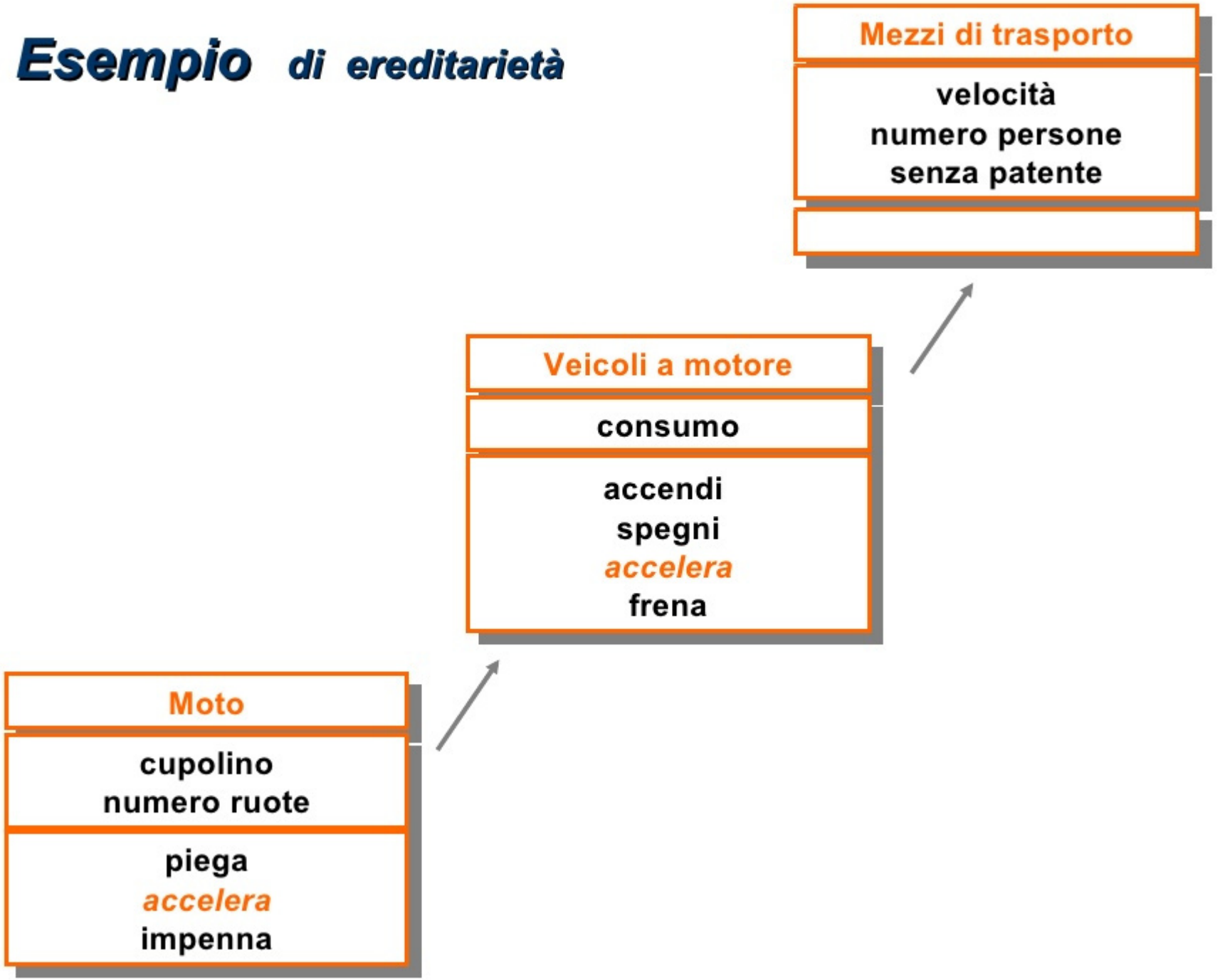
Eredità singola



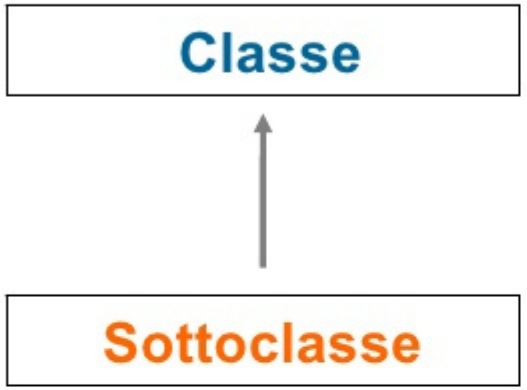
Eredità multipla



Esempio di ereditarietà



Ereditarietà



La **Sottoclasse** si differenzia:

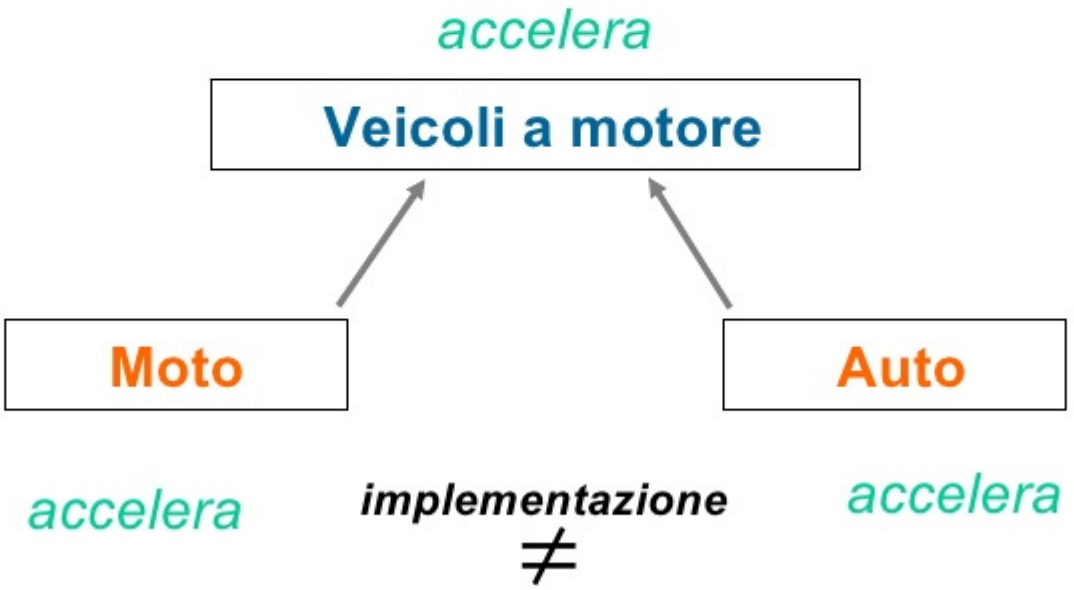
1. per ***Estensione*** quando la sottoclasse ***aggiunge*** nuovi attributi e metodi

2. per ***Ridefinizione*** quando la sottoclasse ridefinisce i metodi ***riscrivendone*** il codice ereditato

*(**overriding** **del metodo**)*

Polimorfismo . 1°

Indica la possibilità per i ***metodi*** di ***assumere forme diverse*** (cioè *implementazioni diverse)* all'interno della gerarchia delle classi



Polimorfismo . 2°

Indica la possibilità per i ***metodi*** di ***assumere forme diverse*** (cioè *implementazioni diverse)* all'interno della stessa classe

Classe Auto

- ***Frena*** ( )
- ***Frena*** (mano, 3)
- ***Frena*** (motore)

= ***nome***

≠ ***parametri***

(overloading dei metodi)

Oggetti

- **Attributi** *e* **Metodi**
- **Incapsulamento**
- **Struttura** *degli oggetti*
- **Interazione** *tra oggetti*
- **L'interfaccia** *verso l'esterno*

attributo1
attributo2
attributo3
attributo4

*Costituiscono la **memoria** dell'oggetto e consentono di tenere traccia dello **stato** dell'oggetto*

metodo1
metodo2
metodo3
metodo4
metodo5
metodo6

*Sono le operazioni che un oggetto è in grado di compiere (**comportamenti**)*

***Attraverso i metodi** un oggetto può accedere alla sua memoria e **modificare** il suo stato*

attributi

Descrivono le proprietà **statiche** *dell'oggetto*

Nella ***programmazione*** *gli attributi vengono realizzati attraverso l'uso delle* **variabili** *utilizzate dall'oggetto per memorizzare i dati*

metodi

*Descrivono le proprietà **dinamiche** dell'oggetto*

*Nella **programmazione** i metodi vengono realizzati attraverso la scrittura di codice (**procedure** e **funzioni**) che implementano le operazioni dell'oggetto*

Incapsulamento

La proprietà dell'oggetto di ***incorporare*** al suo interno ***attributi*** e ***metodi*** viene detta *incapsulamento*

L'oggetto è quindi un ***contenitore*** sia di strutture dati e sia di procedure che li utilizzano

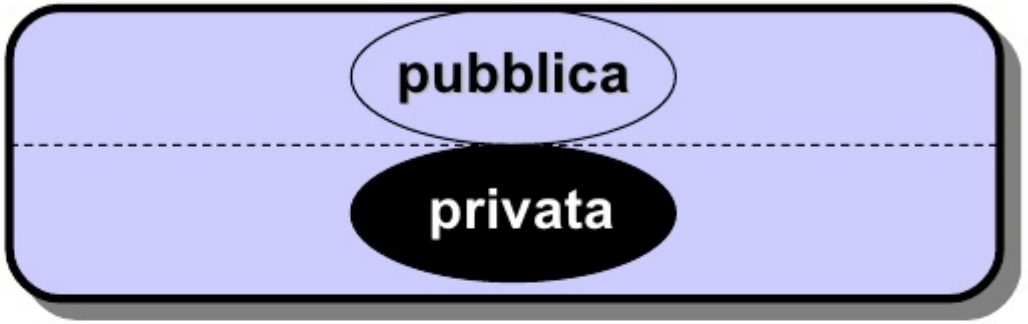
Viene visto come una ***scatola nera*** (*o blackbox*) permettendo così il

mascheramento dell'informazione (***information hiding***)

Sezione ***Pubblica***

Sezione ***Privata***

Struttura degli oggetti



Sezione **Pubblica**

attributi *e* ***metodi***
che si vogliono rendere ***visibili*** all'esterno
(e quindi utilizzabili dagli altri oggetti)

Sezione **Privata**

attributi *e* ***metodi***
che ***non sono accessibili*** ad altri oggetti
(e quindi si rendono invisibili all'esterno)

L'interfaccia verso l'esterno

Un oggetto può essere utilizzato ***inviando ad esso dei messaggi***

L'insieme dei messaggi rappresenta ***l'interfaccia*** di quell'oggetto

L'interfaccia ***non consente di vedere*** come sono implementati i metodi, ma ne permette il loro utilizzo e l'accesso agli attributi pubblici

Ambiente esterno

Sezione
Pubblica

avvia( )
accelera( )
sterza( )
frena( )

liv.carburante
velocità

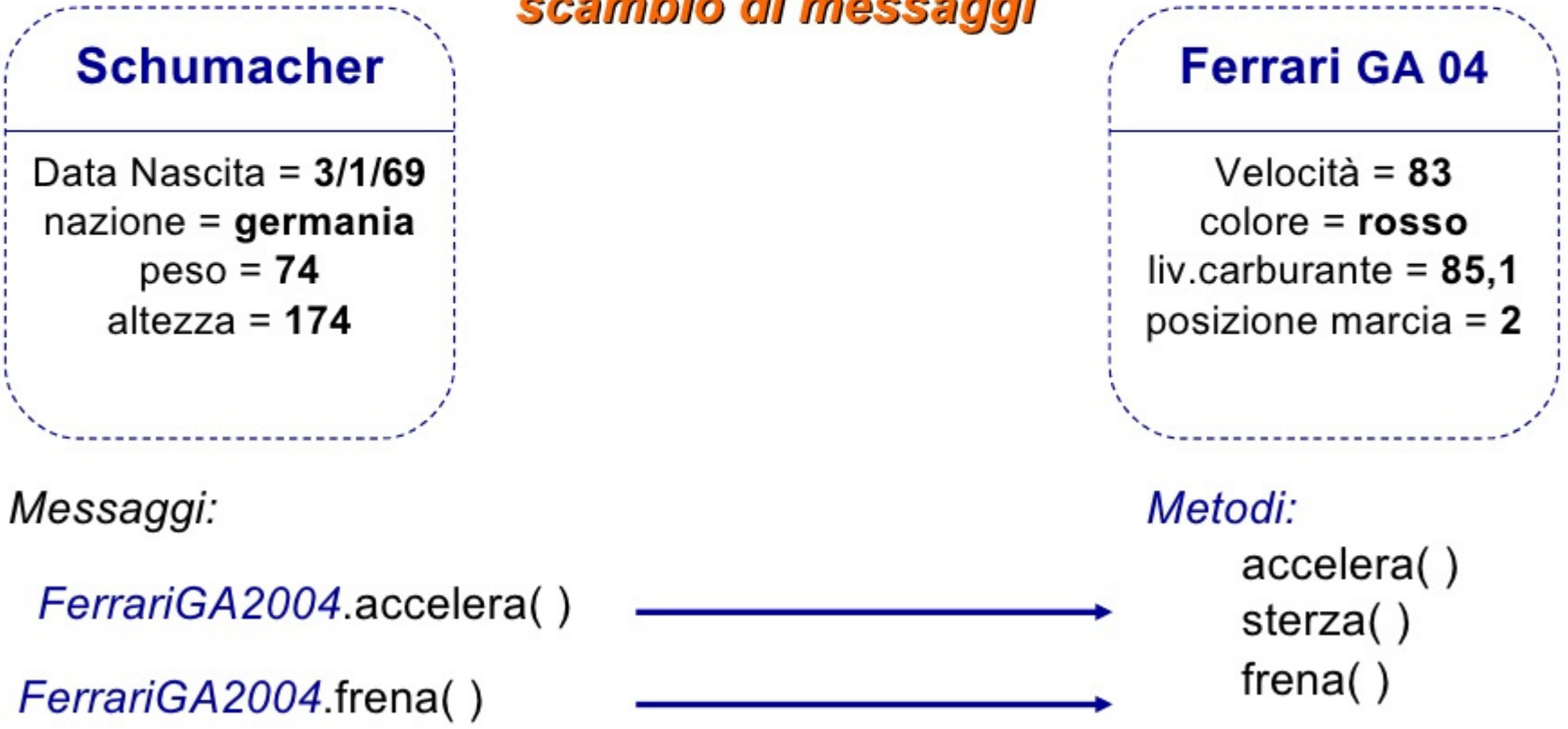
Sezione
Privata

?

Interazione fra gli oggetti

Un ***programma ad oggetti*** è caratterizzato dalla presenza di tanti oggetti che ***interagiscono*** fra loro attraverso il meccanismo dello

scambio di messaggi



Interazione fra gli oggetti

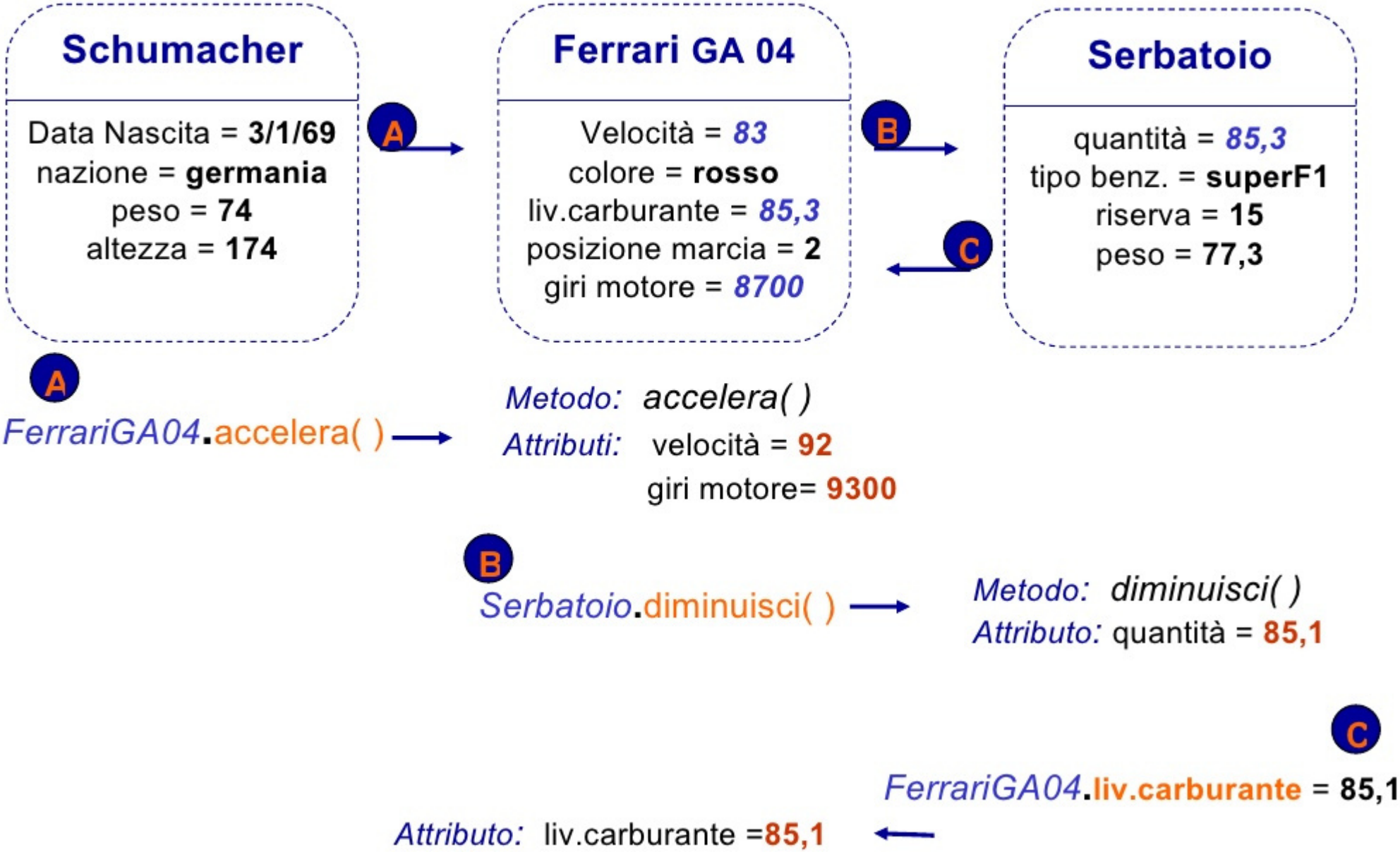
metodi

*Quando un oggetto **invoca** un metodo di un altro oggetto ?*

- *quando vuole **avere un'informazione** sul secondo oggetto*
- *quando vuole **modificare lo stato** del secondo oggetto*

*(in sostanza **conoscere** o **modificare** il valore dei suoi **attributi**)*

Es. di scambio di messaggi



Nuova Metodologia

Per costruire un programma orientato agli oggetti occorre:

- Identificare gli ***oggetti*** che caratterizzano il modello del problema
- Definire le ***classi***, indicando gli ***attributi*** e i ***metodi***
- Stabilire come gli oggetti ***interagiscono*** fra loro.

Linguaggi di programmazione orientati agli oggetti

Linguaggi ***Puri*** ⟶ **ogni cosa è un oggetto**

(...anche un numero intero è definito come oggetto)

. ***Smalltalk***

. ***Eiffel***

Linguaggi ***ibridi*** ⟶ alcuni tipi di dati ***non sono*** oggetti

(maggiore libertà a scapito di una maggiore chiarezza)

. ***C++***

. ***Java***

. ***Visual Basic***

. ***Delphi***

C++

```
# include <iostream.h>
class Base {
friend void funzione {Base&};
public:
                int pubblico;
                void Pubblico() {cout <<'Pubblico: '<<pubblico<<end1;};
private:
                int privato;
                void Privato() {cout <<'Pubblico: '<<pubblico<<end1;};
protected:
                int protetto;
                void Protetto() {cout <<'Protetto: '<<protetto<<end1;};
                };
void funzione{Base &istanza}  {
                istanza.pubblico =1;
                istanza.privato =2;
                istanza.protetto=3;
                istanza.Pubblico();
                istanza.Privato();
                istanza.Protetto();
                }
void main()   {
Base  istanzabase;
istanzabase.pubblico =10;
funzione(istanzabase);
}
```

Java

```
import java.applet.*;
import java.awt.*;
public class semplice extends Applet
{
        private  TextArea ta;
        private  Label l;
        public void init()
         {
                ta = new TextArea(10,8);
                l = new Label("numeri casuali", Label.CENTER);
                l.setBackground(Color.Yellow);

                setLayout(new BorderLayout());
                add(l, "center");
                add(ta "east");
                generaNumeri();
         }
        Public void generaNumeri()
        {
        int casuale;
        for (int i=1; i<=10; i++)
        {
                casuale = (int (Math.random()*1000);
                ta.append("->  "+casuale+"\n");
        }
 }
}
```

```
Private Sub Command1_Click()
        N = Val(InputBox("Numero di cui vuoi la radice quadrata :","Radice
Quadrata"))
        Msg = "La radice quadrata di "& N
        SqrN = RadiceQ(N)
        Select Case SqrN
                Case 0
                        Msg = Msg & "è zero."
                Case –1
                        Msg = Msg & "è un numero immaginario."
                Case Else
                        Msg = Msg & "è" & SqrN
        End Select
        MsgBox Msg
End Sub
```

Delphi

```
unit somma;
Interface

uses
  Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,  StdCtrls;

type
  TForm1 = class(TForm)
        Button1 : TButton;
        Edit1     : TEdit;
        Edit2     : TEdit;
        Edit3     : TEdit;
        Button2 : TButton;
        procedure Button2Click(Sender: TObject);
        procedure Button1Click(Sender: TObject);
  private

  public

  end;

var
  Form1: TForm1;
```

```
implementation
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
   Close;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   Edit3.text := IntToStr(StrToInt(Edit1.text) +
                                  StrToInt(Edit2.text));
end;

end.
```