

La Programmazione Orientata agli Oggetti

(O.O.P.)

Object-Oriented
Programming

Estratti da presentazioni di Maria Caporale, di Roberta Gerboni e dal sito schoolfuture.wordpress.com

OOP – NUOVO PARADIGMA

Dall'inizio degli anni '90 dello scorso secolo, la programmazione orientata ad oggetti OOP si impone come nuova metodologia di programmazione.

Si abbandonano i vecchi paradigmi (imperativo, funzionale, logico) per dare lo start ad una nuova metodologia: «Paradigma ad Oggetti» il cui scopo principale è quello di Progettare un Sistema e non un singolo programma. I vecchi paradigmi, con le loro strutture dati e di controllo, non si sono dimostrati strumenti efficaci; il risultato dell'applicazione di questi strumenti è un sistema software di bassa qualità.

OOP – NUOVO PARADIGMA

Come deve essere «un sistema software di alta qualità»



OOP – NUOVO PARADIGMA

Parametri valutativi

SISTEMA SOFTWARE
DI ALTA QUALITÀ

Documentato

- Specifiche Funzionali;
- Specifiche Tecniche;

Parametrizzato

- Nessuna istruzione statica;
- Utilizzo di parametri e variabili;

Modulare

- Componenti (moduli) indipendenti ma integrati tra loro;
- Ogni modulo del sistema si occupa di gestire funzionalità specifiche;

Riusabile/ Affidabile

- Riutilizzabile in altri contesti;
- Riutilizzato, quindi già testato e collaudato, implica un'affidabilità maggiore;

Incrementabile/ Estendibile

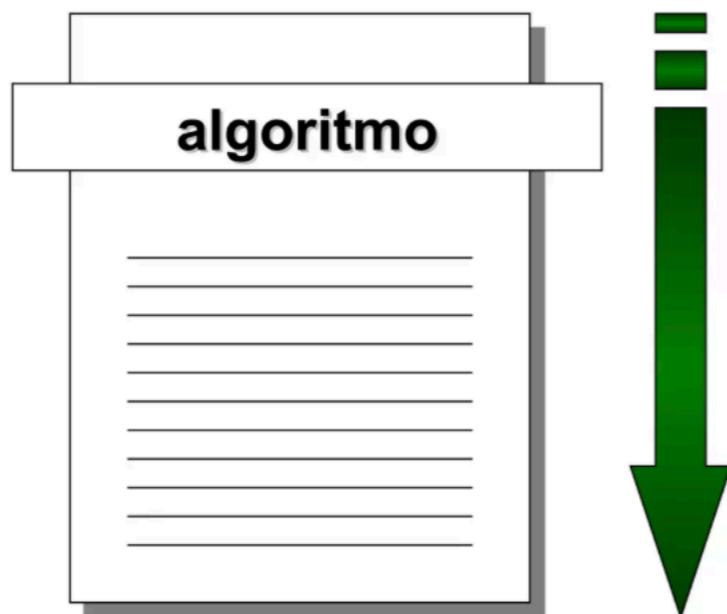
- Incrementabile ed estendibile con un impatto minimo;

Indipendente dalla piattaforma

- Non dipende dalla piattaforma di utilizzo né dal sistema operativo;

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 :

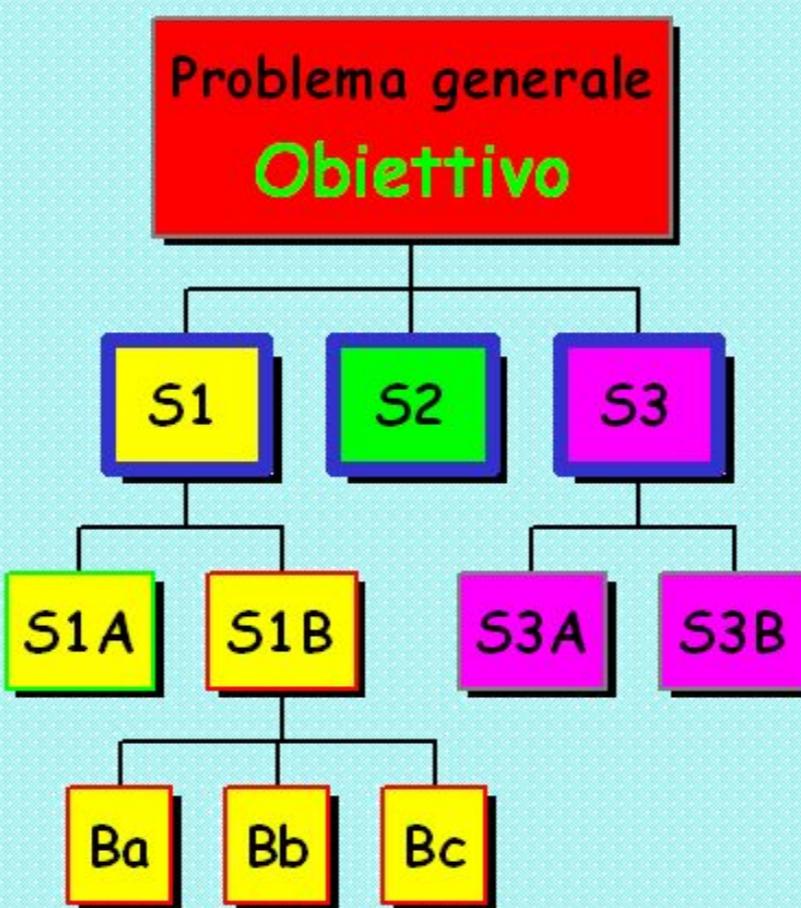
*Applicando i principi
della **programmazione strutturata**:*

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

Metodologia di analisi top-down

Top-down

è la tecnica che consiste nel definire la strategia di risoluzione di un problema attraverso livelli di dettaglio sempre più precisi, scomponendo il problema generale in sottoproblemi particolari.



Metodologia di analisi top-down

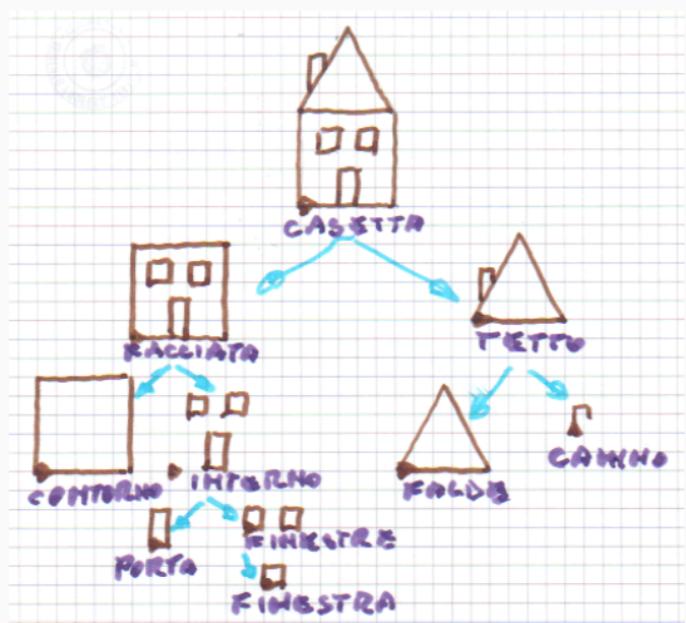
Il problema

Voglio far disegnare a Tartaruga una casetta con porte finestre e camino. Qualcosa che assomigli a questo:

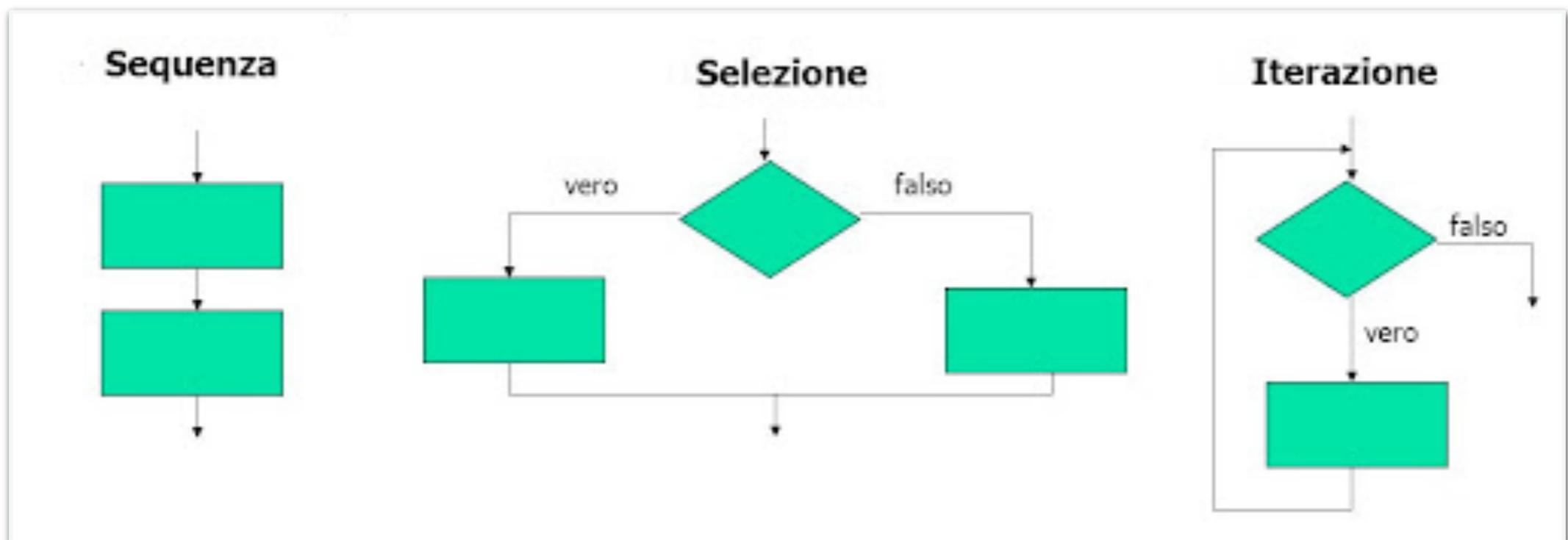


L'analisi

Prima di scrivere un programma che disegni la casetta, dobbiamo analizzarlo. L'analisi consiste nel suddividere il problema in parti più semplici e ripetere questa operazione in modo ricorsivo finché si arriva a parti elementari. Nel nostro caso una possibile analisi è:

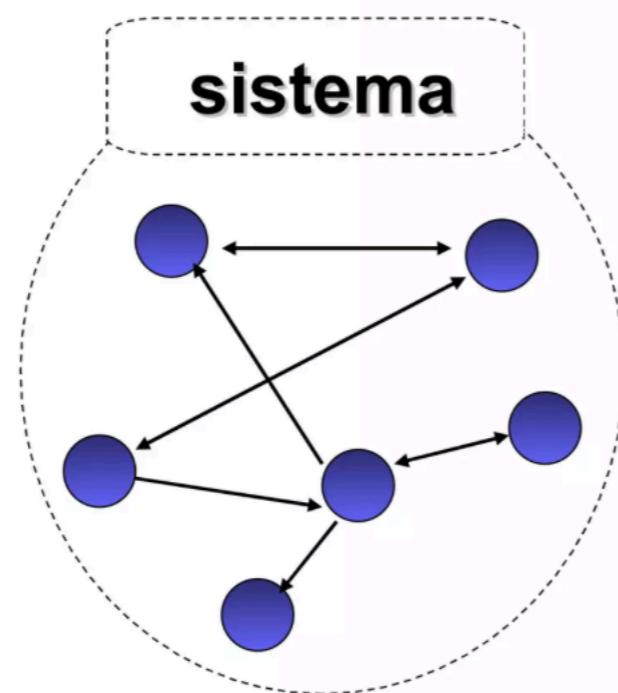


Strutture di controllo della programmazione procedurale / imperativa



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)

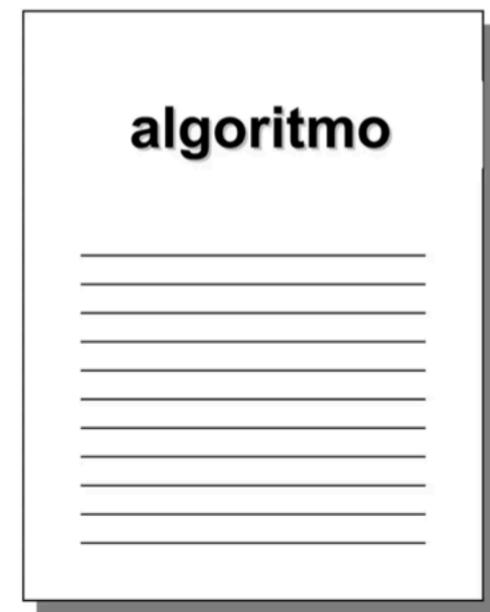
Come si programma in O.O.P. ?

*... applicando i principi
della **programmazione strutturata***

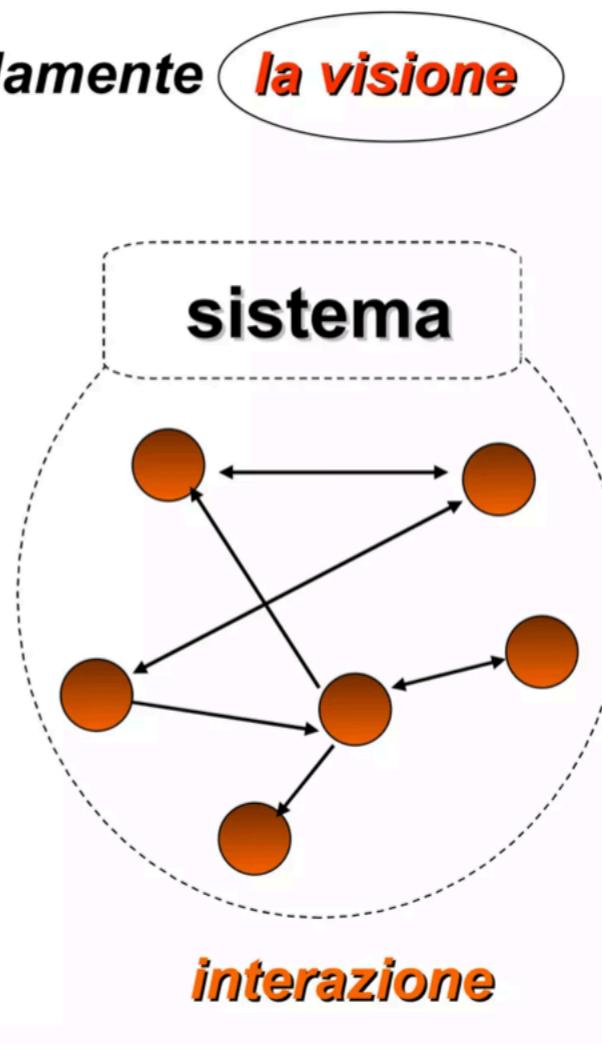
cioè ...

- *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***

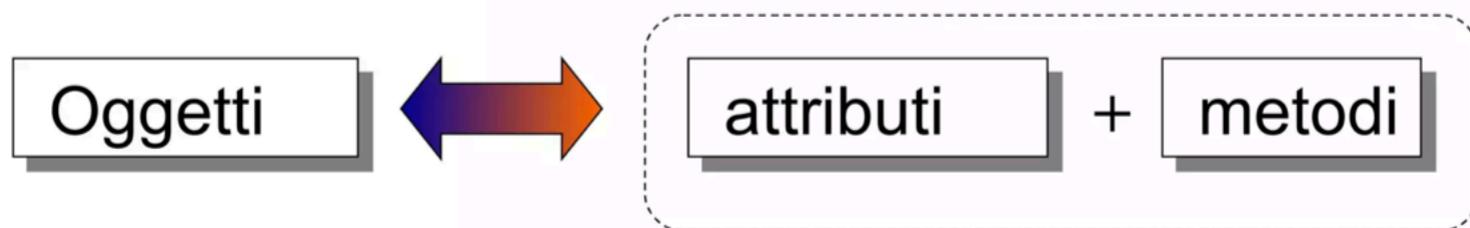
Per cui è necessario individuare :

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

Per la risoluzione di un **Problema**



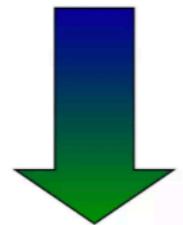
Per la gestione di un **Sistema**



ricapitolando...

Programmazione
imperativa

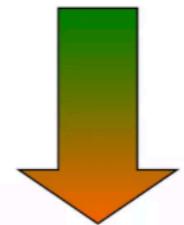
Problema
complesso



scomposizione in
procedure

Programmazione
orientata agli oggetti

Sistema
complesso



scomposizione in
entità interagenti
(oggetti)

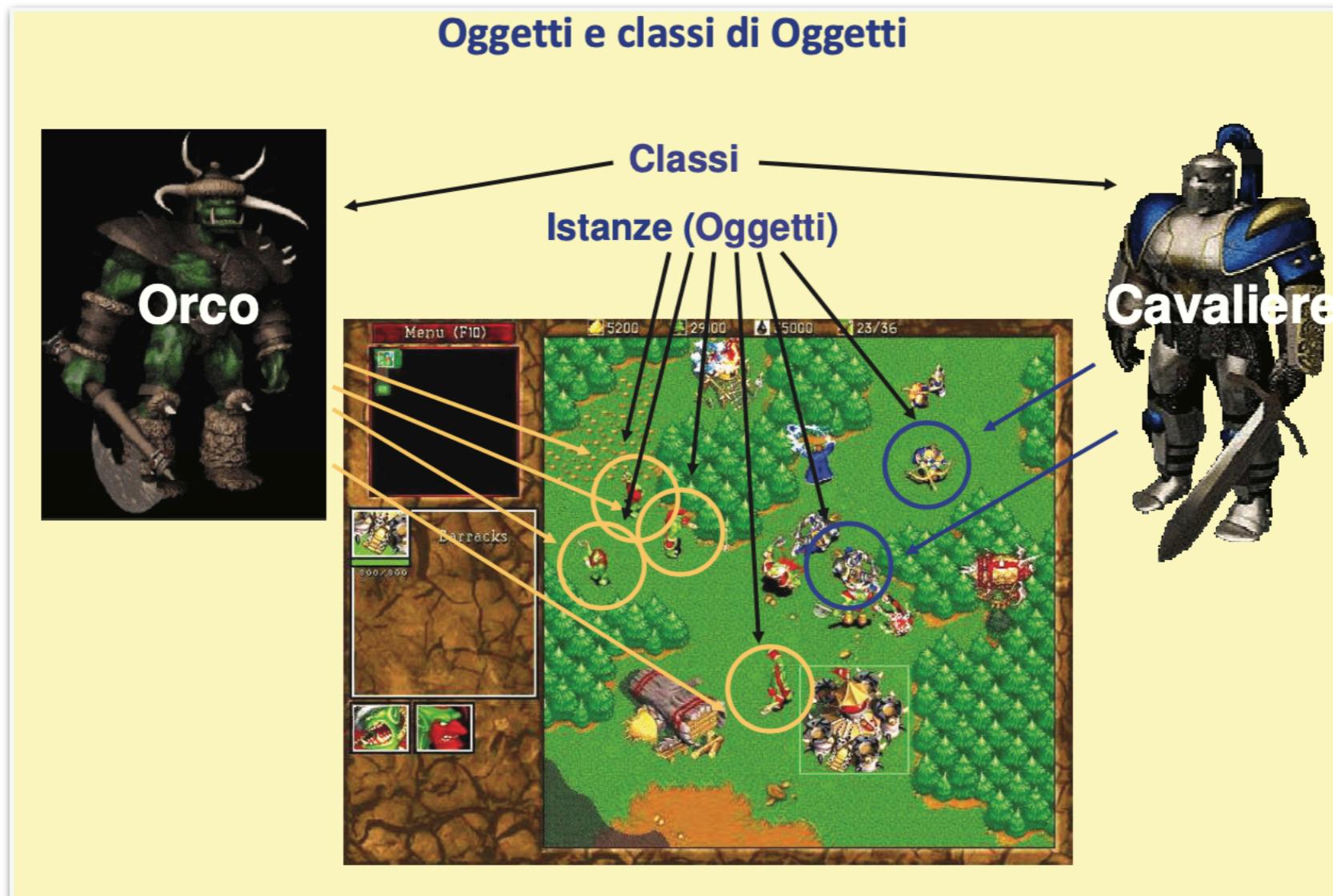
Oggetti

e

Classi di oggetti

(secondo il formalismo UML)

Esempio 1 - un videogioco



L'oggetto - gli attributi - i metodi

Ogni **oggetto** è definito da:

- **Attributi** che rappresentano le sue **caratteristiche o proprietà fisiche utili a definire il suo stato e sono campi (variabili o costanti)**
- **Metodi** che rappresentano i **comportamenti ammissibili** o le **azioni, le proprietà dinamiche**, cioè le funzionalità dell'oggetto stesso e chi usa l'oggetto può attivarli.
I metodi vengono realizzati con le **funzioni** contenenti le istruzioni che implementano le azioni dell'oggetto e possono avere parametri e fornire valori di ritorno.

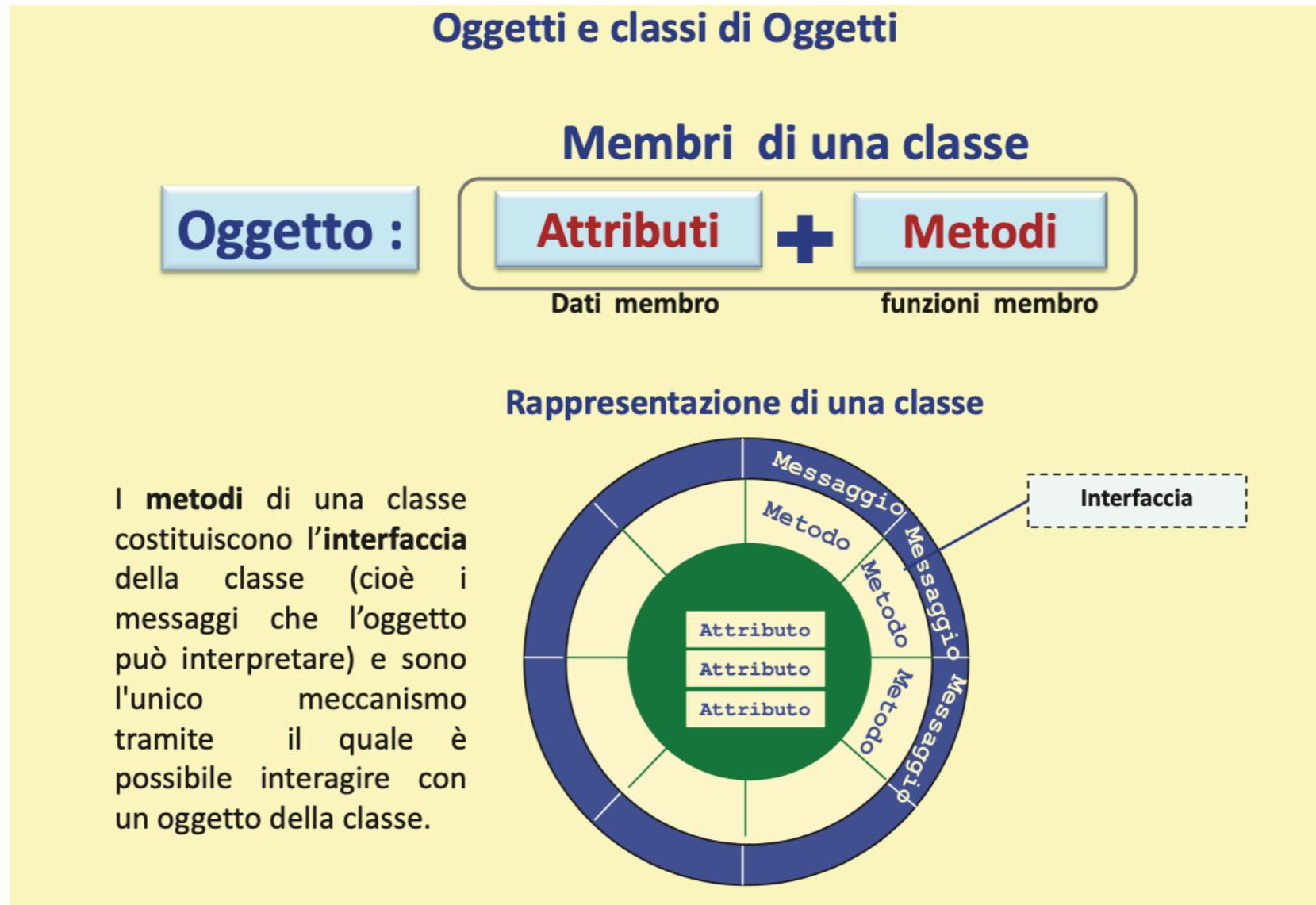
Lo **stato di un oggetto**, è l'insieme dei valori delle sue proprietà in un determinato istante di tempo (**valori assunte dalle variabili**). Se cambia anche un solo valore di una proprietà di un oggetto, il suo stato varierà di conseguenza.

Nel corso dell'elaborazione, un oggetto è un'entità soggetta ad una **creazione**, ad un **suo utilizzo** e, infine, alla sua **distruzione**.

8

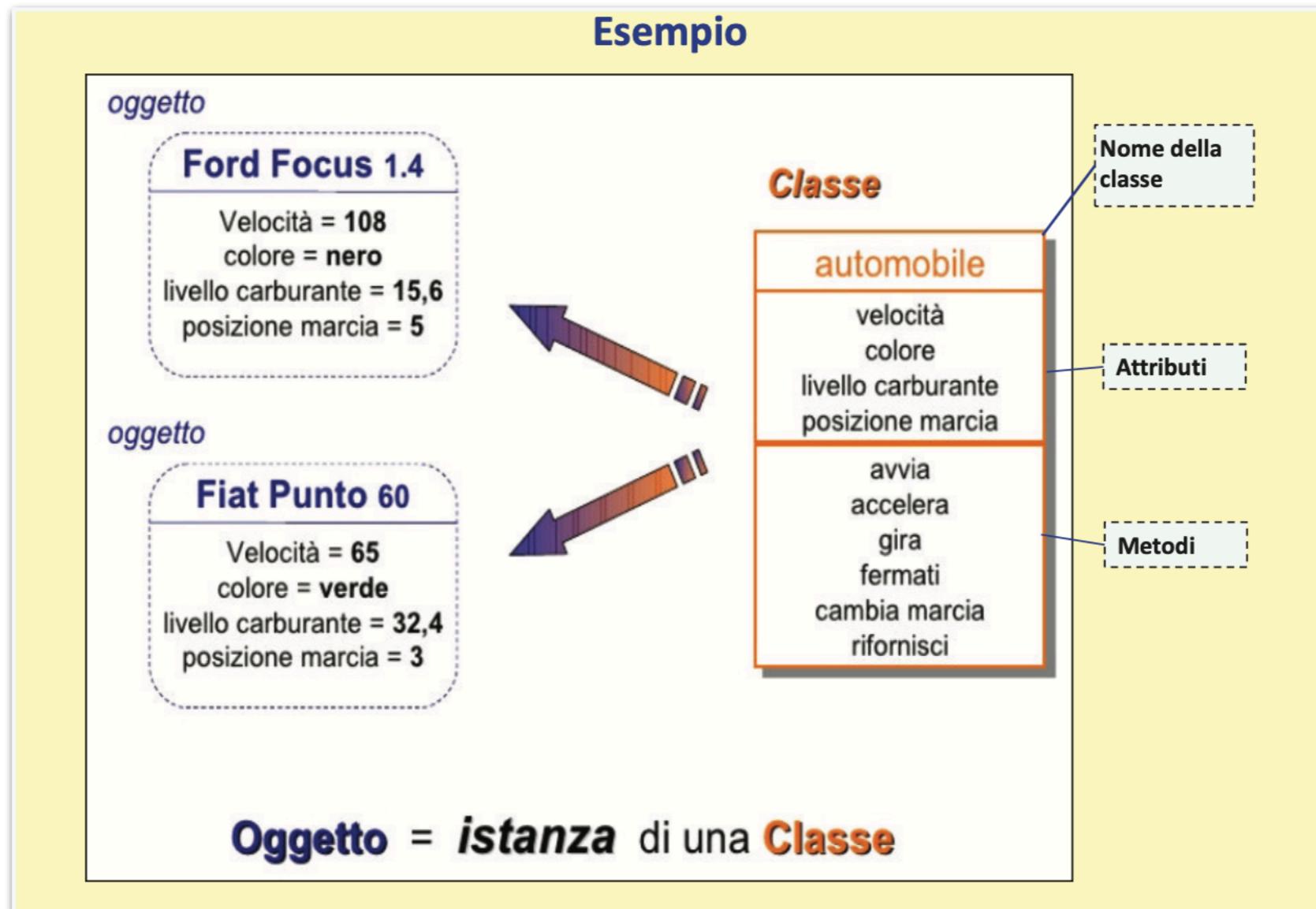
- L'attributo: è ciò che caratterizza un oggetto. Si può chiamare attributo/variabile istanza/di esemplare
- I metodi che rappresentano comportamenti di un oggetto, si chiamano metodi di istanza

La classe: descrizione di un Abstract Data Type (ADT)



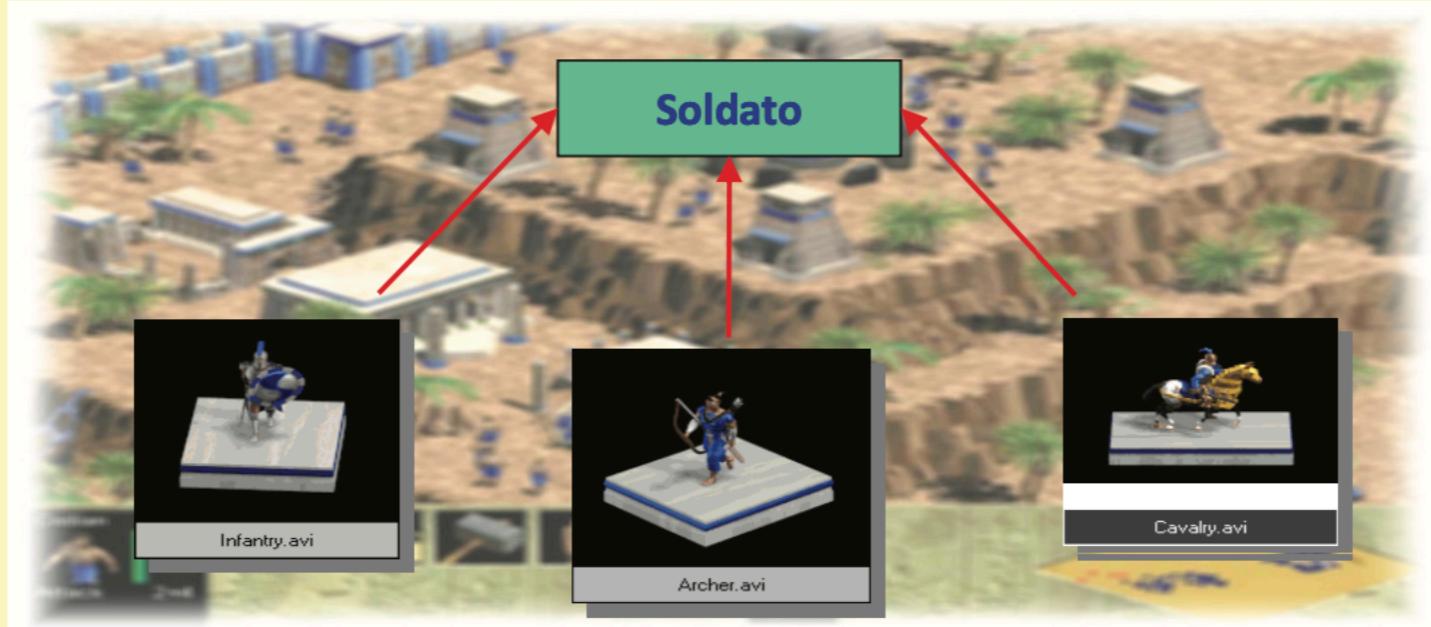
Oggetto: si può chiamare istanza o esemplare di una classe

Esempio 2 - automobile



Esempio 3 - soldato

Esempio



- Tutti i soldati devono capire il messaggio *attacca*. Il messaggio ha conseguenze diverse a seconda del tipo di **soldato**:
 - un **arciere** scaglia una freccia
 - un **fante** colpisce di spada
 - un **cavaliere** lancia una lancia
- Il **giocatore/computer** deve gestire una lista di soldati e poter chiedere ad ogni soldato di *attaccare* indipendentemente dal tipo.

Diagramma delle Classi

**caratteristiche
specifiche**

nome classe

attributo1
attributo2
attributo3
attributo4

**comportamenti
generali**

metodo1
metodo2
metodo3
metodo4
metodo5
metodo6

automobile

velocità
colore
livello carburante
posizione marcia

avviati
accelera
sterza
spegniti
cambia marcia
frena

Diagramma degli Oggetti

nome oggetto

attributo 1 = **val1**
attributo 2 = **val2**
attributo 3 = **val3**
attributo 4 = **val4**

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

cambiano le caratteristiche specifiche a seconda dell'oggetto istanziato

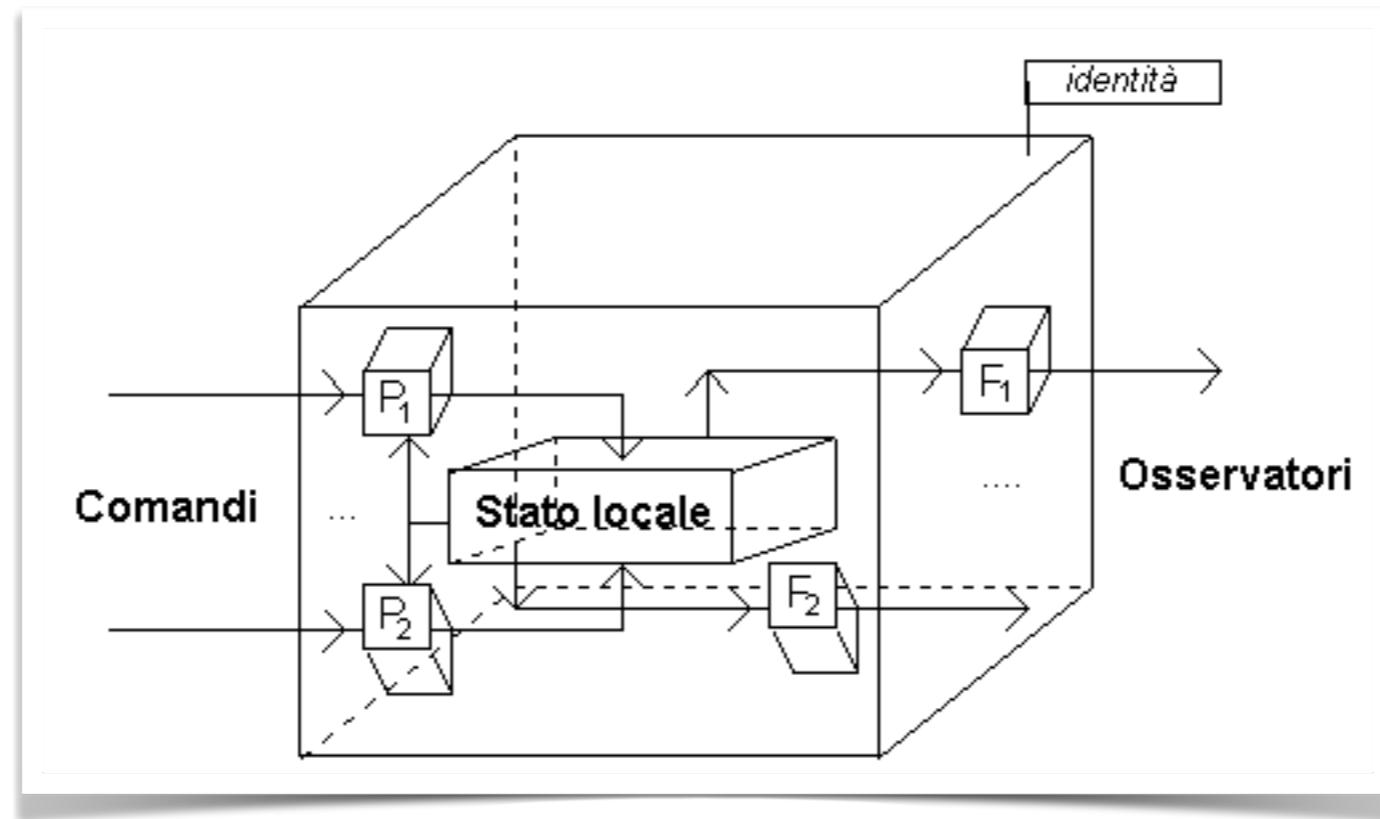
Ford Focus 1.4

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

Ferrari GA 04

Velocità = **83**
colore = **rosso**
livello carburante = **85,1**
posizione marcia = **2**

Rappresentazione di un oggetto



- Un oggetto deriva da un ADT (classe) ed è dotato di
 - **identità**, cioè un riferimento, un nome
 - **stato**, cioè il valore assunto dai propri attributi
 - **comportamento**, cioè l'insieme dei propri metodi che realizzano il comportamento dell'oggetto. P₁, P₂ sono metodi che modificano lo stato dell'oggetto; F₁, F₂ sono metodi che osservano lo stato dell'oggetto, restituiscono un valore risultato di una elaborazione (anche molto semplice).

OOP - proprietà incapsulamento - information hiding

Principali proprietà

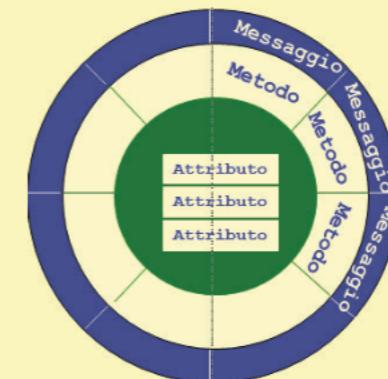
1. Incapsulamento e information hiding

Uno dei grossi vantaggi è l'**incapsulamento**, cioè la proprietà degli oggetti di *mantenere al loro interno* sia gli **attributi** (le variabili) che i **metodi** (le funzioni), che descrivono rispettivamente lo stato e le azioni eseguibili sull'oggetto.

Si ha quindi come una capsula che **isola** l'oggetto dall'ambiente esterno proteggendo l'oggetto stesso. Legato a questo concetto c'è anche quello di **information hiding** (mascheramento delle informazioni rese quindi **invisibili** dall'esterno).

Infatti l'incapsulamento:

- **nasconde l'implementazione interna:** dall'esterno della classe è noto cosa fa un metodo pubblico, cioè se ne conosce l'interfaccia (funzionalità svolta, parametri in ingresso e tipo restituito), **ma non è noto come lo fa.**
- **consente un accesso protetto e «ragionato»** dall'esterno: ad esempio è possibile proteggere da scrittura un attributo definendolo *private* e implementando un metodo pubblico che ne restituisce il valore.



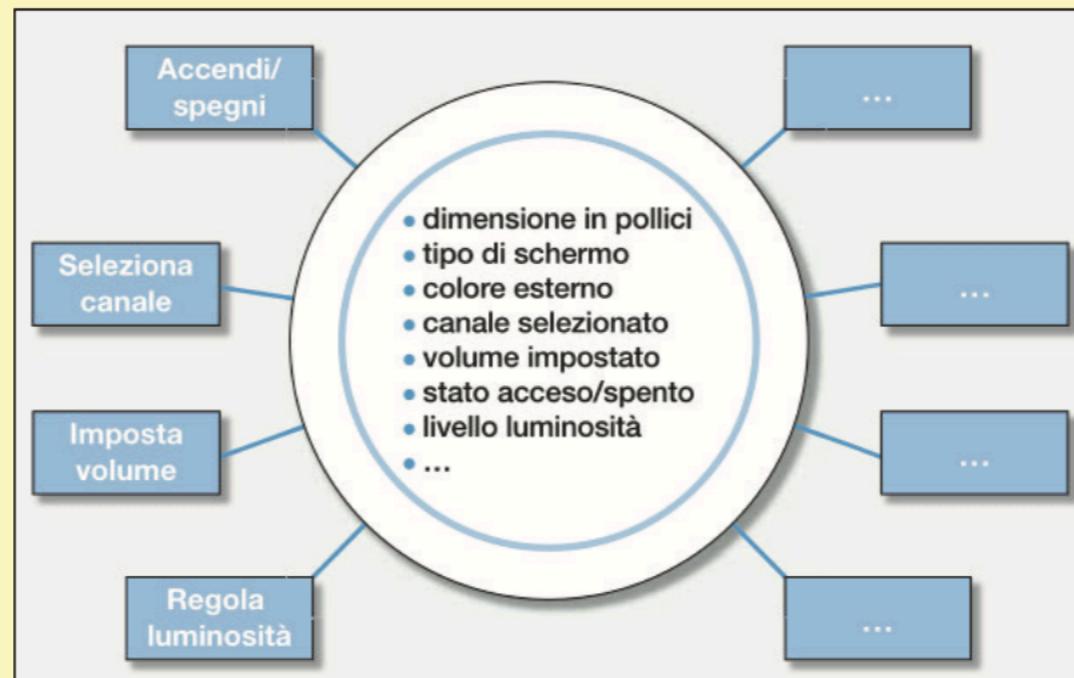
13

OOP - proprietà incapsulamento - information hiding

Esempio

La struttura privata interna di un oggetto che modella un **televisore** è incapsulata e protetta dall'interfaccia che espone all'esterno le sole *operazioni pubbliche* che permettono di interagire con l'oggetto stesso.

Classe Televisore



OOP - proprietà incapsulamento - information hiding

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**

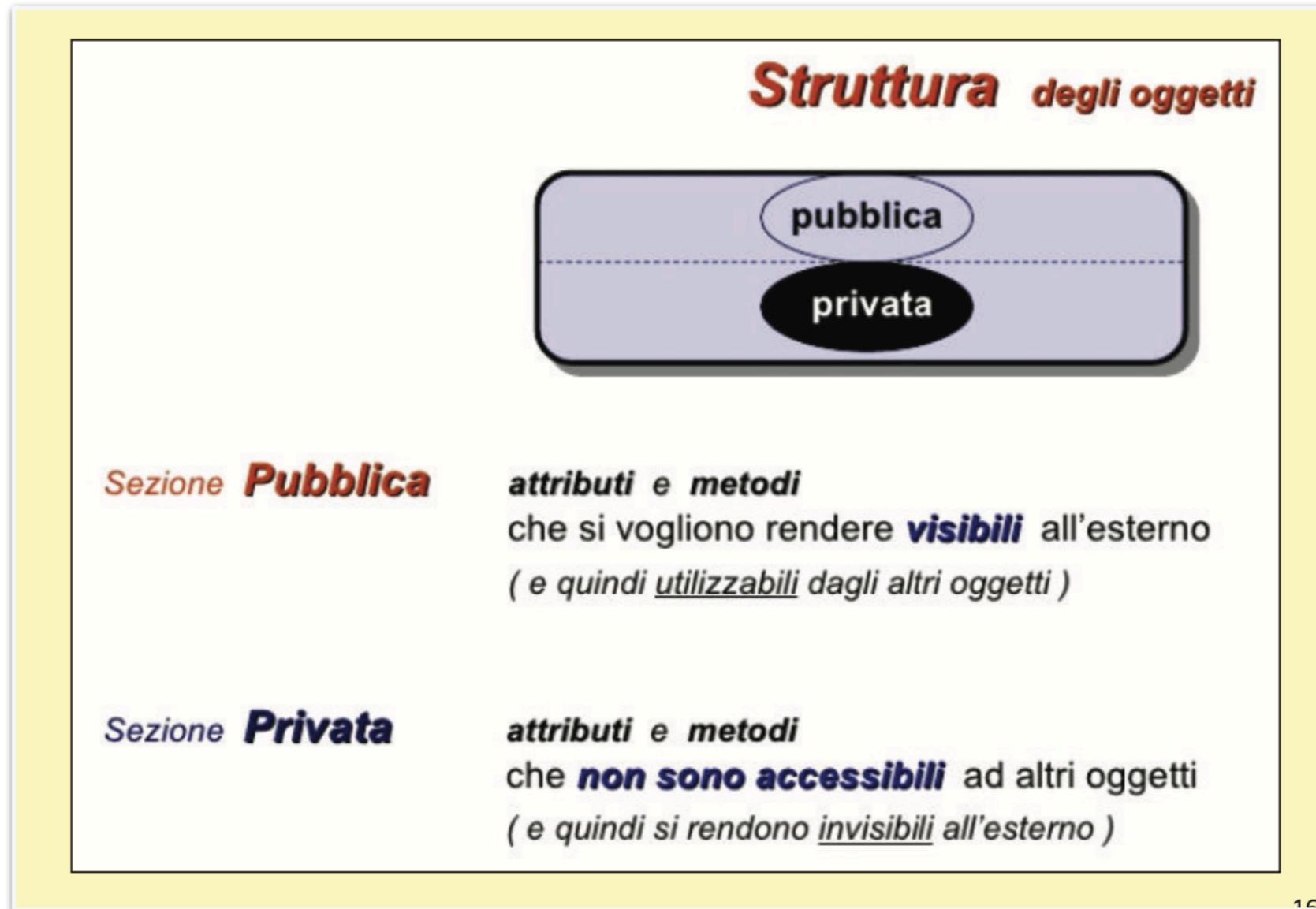
attributi

metodi

attributi

metodi

OOP - proprietà incapsulamento - information hiding



16

OOP - interfaccia pubblica di una classe / di un oggetto per l'accesso ai metodi pubblici

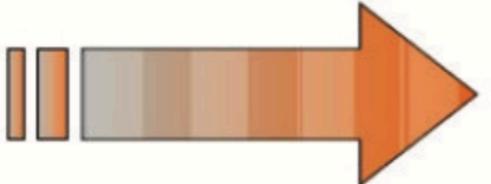
L'interfaccia verso l'esterno

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

Ambiente esterno



Sezione Pubblica

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

liv.carburante

velocità

Sezione Privata

?

17

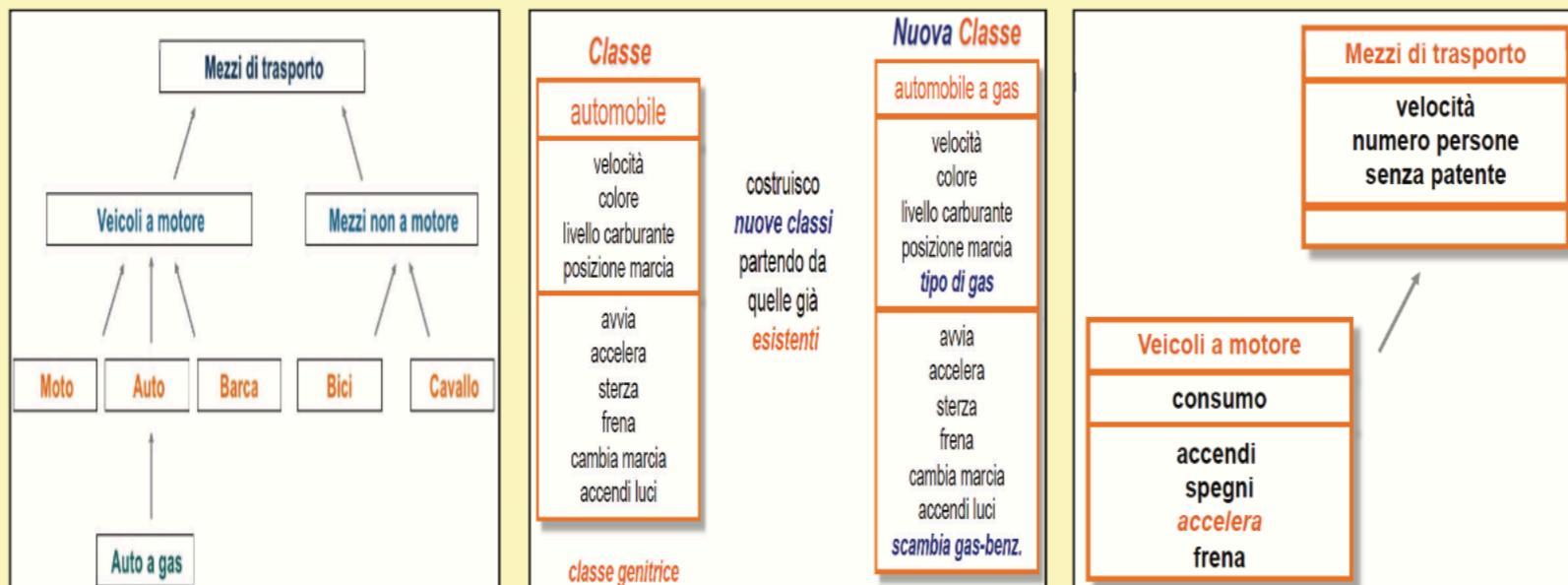
OOP - proprietà ereditarietà

Principali proprietà

2. Ereditarietà

Uno strumento molto efficace nella programmazione ad oggetti è l'**ereditarietà**, che consente di:

definire una **nuova classe** che mantiene le proprietà di una classe già esistente ,
ma **aggiunge nuovi attributi e metodi**.



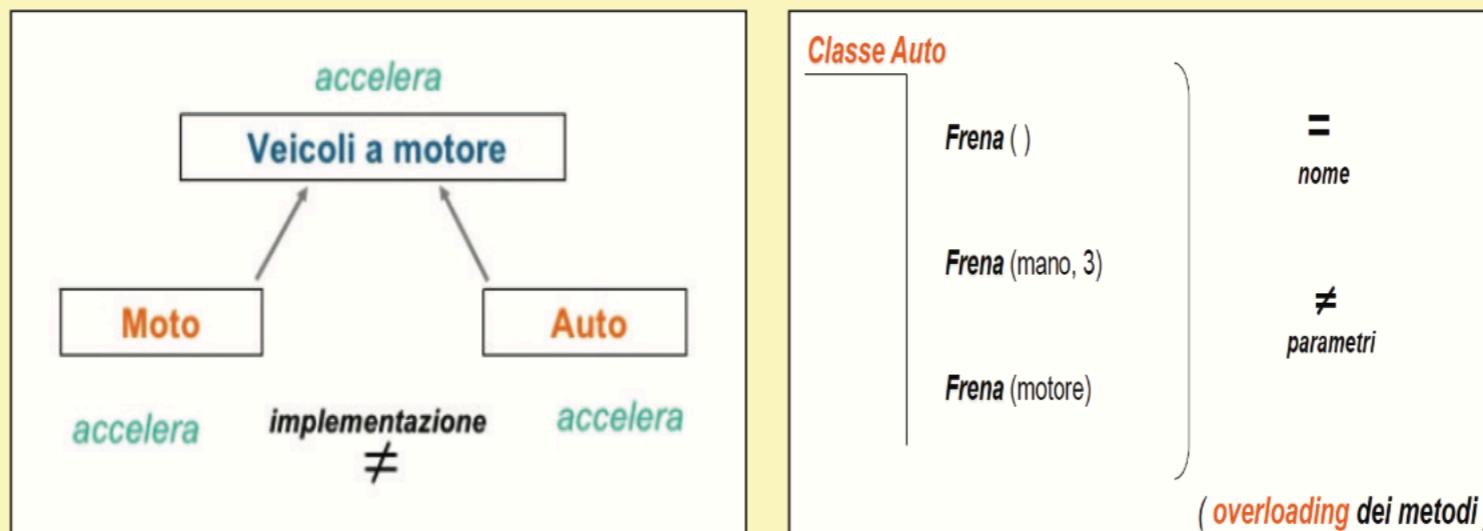
OOP - proprietà polimorfismo

Principali proprietà

3. Polimorfismo

Il **polimorfismo** è la possibilità di richiamare un unico metodo che avrà un comportamento diverso in base al tipo di oggetto su cui viene applicato.

Questo è reso possibile grazie all'**ereditarietà** e all'**overloading**: possiamo definire un metodo con lo stesso nome su due classi. Richiamando il nome del metodo otterremo risultati diversi in base al tipo di oggetto su cui è stato richiamato.



OOP - rappresentazione formale di un ADT - classe

Classi e oggetti, attributi e metodi nei diagrammi UML

Tra i vari formalismi grafici che UML mette a disposizione per descrivere i diversi aspetti di un sistema software a molteplici livelli di dettaglio, uno in particolare fornisce una notazione grafica per formalizzare le classi e le relazioni che intercorrono tra di esse: il **diagramma delle classi** (class diagram):

È inoltre possibile classificare le componenti:

simbolo

private -

pubbliche +

protette #

premettendo ai singoli nomi rispettivamente i simboli:

«-» «+» «#»

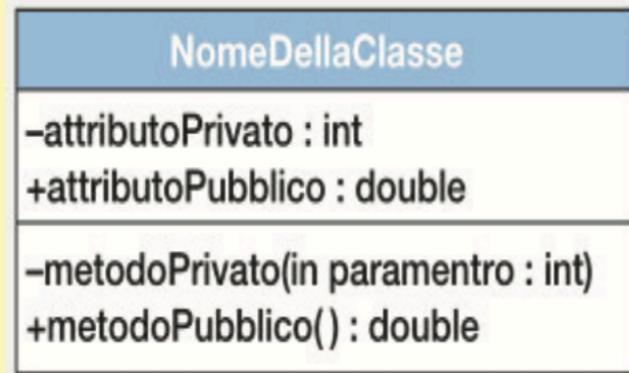
NomeDellaClasse
-attributo_1
-attributo_1
-...
-attributo_n
+metodo_1()
+metodo_2()
+...()
+metodo_n()

Proprietà

Operazioni

OOP - rappresentazione formale di un ADT - classe

UML: diagramma delle classi



Per ogni metodo, oltre al suo livello di visibilità (pubblica o privata) è necessario definire:

- i **nomi** e i **tipi** degli eventuali parametri e il **loro ruolo** (in/out/in-out);
- il **tipo dell'eventuale valore restituito** (che viene scritto dopo le parentesi e il simbolo ‘:’).

Attività

Realizzare il diagramma di classe per le seguenti entità (ADT)

- Automobile (diapositiva 22)
- Televisore (diapositiva 26)
- Contatore di persone
 - caratteristiche: numero di persone contate
 - comportamenti: aggiunge una persona, toglie una persona, azzera il contatore
- Studente: quali sono le proprietà? Quali sono i metodi?

Disegnare gli UML: draw.io, altro ...