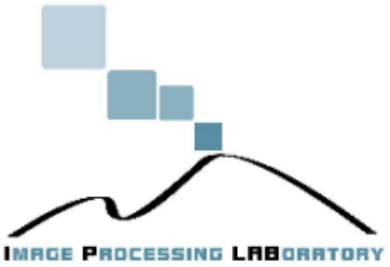
Introduzione alla OOP

Alessandro Ortis

www.dmi.unict.it/ortis/



L’importanza dell’astrazione

Il mondo in cui viviamo è costituito da sistemi molto complessi, in cui oggetti diversi interagiscono tra loro e cambiano il loro modo di agire in funzione di quello che accade.

È difficile gestire una realtà complessa, allo stesso modo è difficile costruire sistemi complessi come ad esempio software di grandi dimensioni.

Un modo per gestire la complessità è l’**astrazione**.

L’importanza dell’astrazione

Esempi di astrazione…

L’importanza dell’astrazione

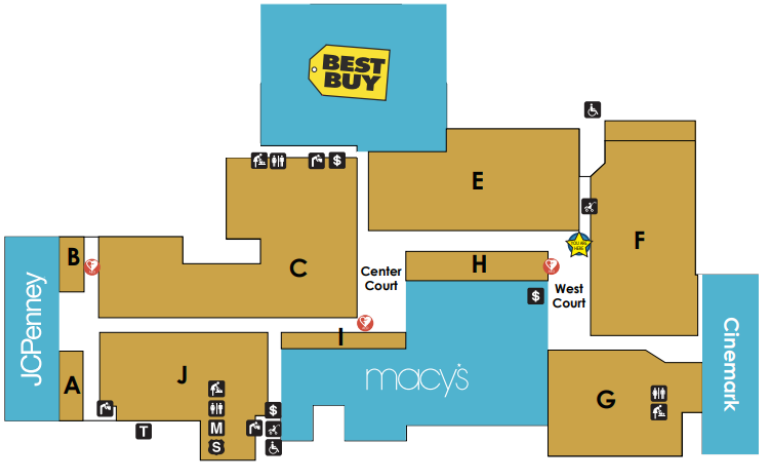
Esempi di astrazione…

- Una piantina stradale rappresenta una astrazione di una città.

L’importanza dell’astrazione

Esempi di astrazione…

- Una piantina stradale rappresenta una astrazione di una città.



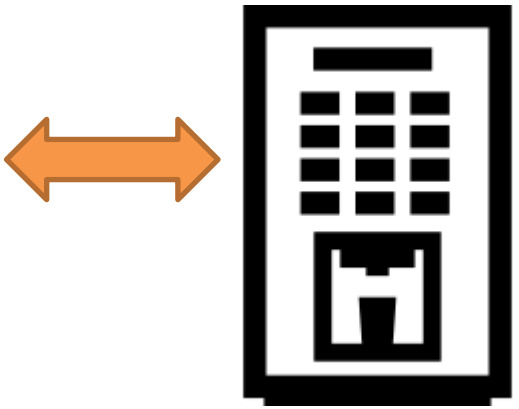
L’importanza dell’astrazione

Esempi di astrazione…

- Una piantina stradale rappresenta una astrazione di una città.

- Come faccio per avere un caffe ?

1. Temperatura acqua
2. Quantità caffe
3. Finezza della macina
4. …



L’importanza dell’astrazione

L’**astrazione** è un procedimento che consente di semplificare la realtà che vogliamo modellare. La semplificazione avviene concentrando l’attenzione solo sugli elementi importanti del sistema complesso che stiamo considerando.

L’importanza dell’astrazione

Si tratta di un concetto fondamentale nella programmazione ad oggetti.

Gli oggetti in un linguaggio OOP forniscono la funzionalità di astrarre, cioè di nascondere i dettagli implementativi interni.

Quando si creano dei programmi mediante un linguaggio ad oggetti, la capacità di astrarre, cioè la capacità di semplificare delle entità complesse in ***oggetti caratterizzati dalle caratteristiche e dalle funzionalità essenziali*** per gli scopi preposti, può risultare determinante.

Evoluzione sino alla OOP

I linguaggi di programmazione permettono di scrivere un programma in una forma comprensibile all'elaboratore.

Sono caratterizzati da una **sintassi,** cioè un insieme di costrutti e di regole. Questi costrutti rappresentano una **astrazione delle possibili operazioni che il processore può eseguire**.

I linguaggi hanno avuto un'evoluzione a partire dagli anni '50: i **linguaggi macchina** sono stati abbandonati per utilizzare i linguaggi più vicini al linguaggio naturale. Questo sviluppo è stato possibile grazie al processo di astrazione.

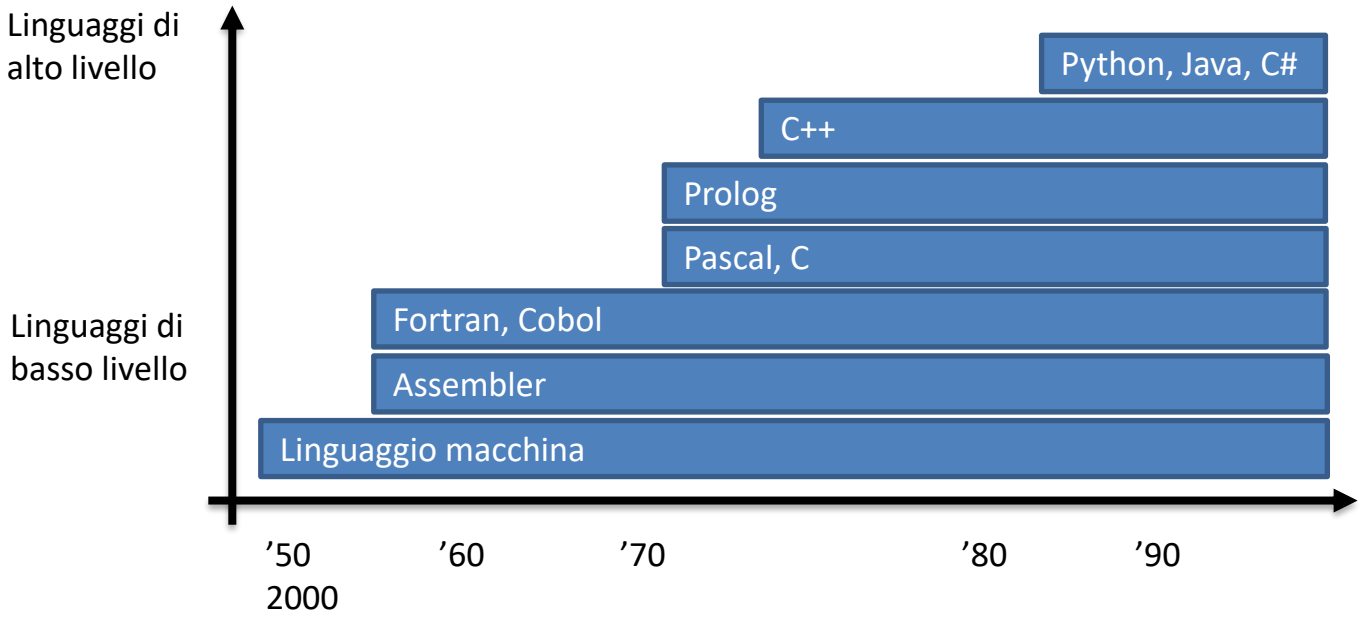
Evoluzione sino alla OOP

Dal 1950 ad oggi sono stati creati moltissimi linguaggi di programmazione, classificabili in diversi modi.

Linguaggio	Anno	Utilizzo principale	Paradigma
Fortran	1954	Calcolo numerico	Imperativo
Cobol	1959	Applicazioni gestionali	Imperativo
Pascal	1971	Usi generali	Imperativo
Prolog	1972	Intelligenza artificiale	Logico
C	1974	Usi generali, programmazione di sistema	Imperativo
C++	1979	Usi generali	Ad oggetti
Python	1991	Usi generali	Ad oggetti
Java	1995	Usi generali	Ad oggetti

Evoluzione sino alla OOP

Dal 1950 ad oggi sono stati creati moltissimi linguaggi di programmazione, classificabili in diversi modi.



Evoluzione sino alla OOP

Linguaggi di basso livello

I primi programmatori avevano come unico strumento il **linguaggio macchina**: potevano comporre istruzioni tramite sequenze di 0 e 1.

Il livello di astrazione è stato elevato con il **linguaggio Assembler**, nel quale le istruzioni venivano indicate con un nome simbolico (es., mov).

Le uniche operazioni che si possono fare con tali linguaggi sono: caricare valori nei registri, operazioni aritmetiche, confronti, salto ad una particolare istruzione.

Evoluzione sino alla OOP

Linguaggi di alto livello

Utilizzano parole chiave in inglese, che facilitano il lavoro di programmazione e di comprensione del codice.

Il programma viene poi processato da un **interprete** o un **compilatore** che si preoccupano di tradurlo in linguaggio macchina.

I vantaggi sono evidenti (es., sviluppo e debug più rapidi, apprendimento rapido) tuttavia nell'uso di questi linguaggi il codice può risultare meno efficiente ed occupare più memoria.

Programmazione procedurale vs. OOP

Nello sviluppo software, usando la metodologia della **programmazione procedurale**, l'interesse principale è rivolto alla sequenza di operazioni da svolgere: si crea un modello indicando le procedure da eseguire in maniera sequenziale per arrivare alla soluzione.

Lo spostamento di attenzione dalle procedure agli oggetti ha portato all'introduzione della **programmazione ad oggetti**. Gli oggetti sono intesi come entità che hanno un loro stato e che possono eseguire certe operazioni.

L'algoritmo perde importanza a vantaggio del concetto di **sistema.**

Programmazione procedurale vs. OOP

Un **algoritmo** è un insieme di istruzioni che a partire dai dati di input permettono di ottenere i risultati di output.

Un algoritmo deve essere riproducibile, deve avere una durata finita e non deve essere ambiguo. Il modo di programmare pone attenzione sulla **sequenza di esecuzione**.

Un **sistema** è una parte del mondo che si sceglie di considerare come un intero, composto da **componenti**. Ogni componente è caratterizzata da **proprietà** rilevanti, e da **azioni** che creano interazioni tra le proprietà e le altre componenti.

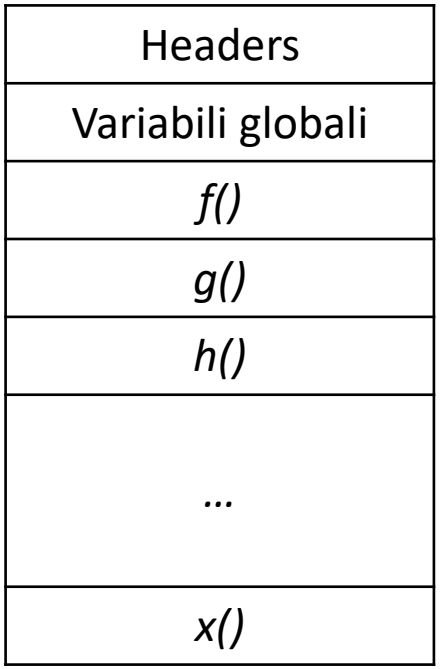
Programmazione procedurale vs. OOP

I linguaggi procedurali hanno dei limiti nel creare componenti software riutilizzabili.

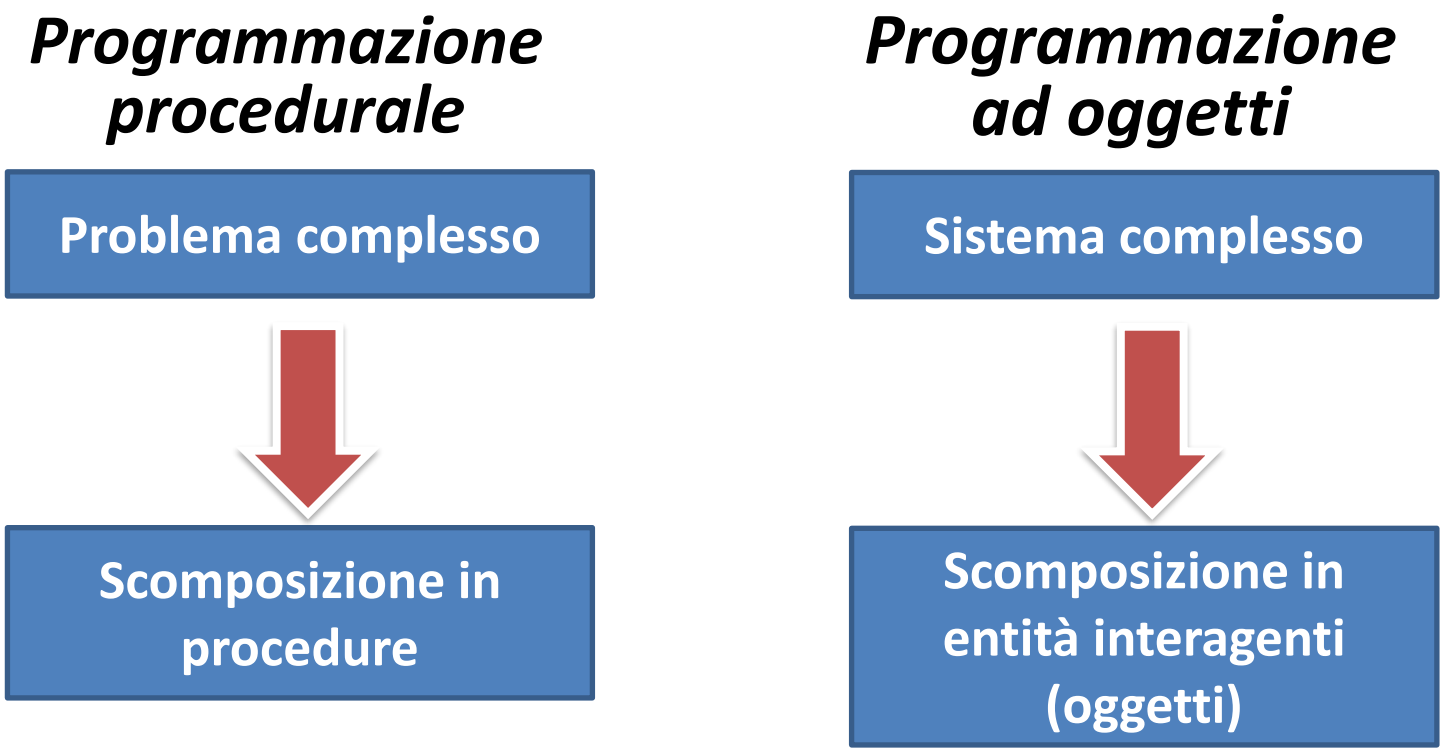
I programmi sono fatti da funzioni, che rappresentano codice riutilizzabile, ma che spesso fanno riferimento a headers e/o variabili globali che devono essere importate insieme al codice delle funzioni.

I linguaggi procedurali non si prestano bene alla modellazione di concetti ad alti livelli di astrazione, utili per rappresentare entità complesse che interagiscono in un sistema reale.

In altre parole, i linguaggi procedurali separano le strutture dati e gli algoritmi



Programmazione procedurale vs. OOP



Programmare ad oggetti (OOP)

Esempio: interfaccia grafica (GUI) di un PC

Componenti

Finestre (proprietà: dimensione, posizione)

Bottoni (proprietà: colore, testo)

Correlazioni

Premendo un bottone si può aprire una finestra (e quindi definire la sua posizione e la sua dimensione)

Programmare ad oggetti (OOP)

Possiamo individuare tre fasi ‘’Object-Oriented’’ (OO):

- Analisi (OOA): identificazione dei requisiti funzionali, delle classi e delle loro relazioni logiche.
- Design (OOD): specifica delle gerarchie tra classi, e delle loro interfacce e comportamenti.
- Programmazione (OOP): implementazione del design, test ed integrazione.

La OOP è il momento in cui si scrive effettivamente il codice.

Programmare ad oggetti (OOP)

La metodologia OO è un modo di pensare al problema in termini di sistema, quindi parte dall'analisi del problema e dalla progettazione della sua soluzione.

Durante la fase di analisi si crea un **modello del sistema**, individuando gli **elementi** di cui è formato e i **comportamenti** che devono avere.

In questa fase non interessano le modalità con le quali i comportamenti vengono implementati, ma soltanto gli **oggetti** che compongono il sistema e **le interazioni tra essi**.

Programmare ad oggetti (OOP)

L'elemento base della OOP è l'**oggetto.**

Un oggetto può essere definito elencando sia le sue caratteristiche, sia il modo con cui interagisce con l'ambiente esterno, cioè i suoi comportamenti.

- Le **caratteristiche** rappresentano gli elementi che caratterizzano l'oggetto, utili per descrivere le sue proprietà e definirne lo stato.
- I **comportamenti** rappresentano le funzionalità che l'oggetto mette a disposizione: chi intende utilizzare l'oggetto deve attivare i comportamenti dell'oggetto stesso

Programmare ad oggetti (OOP)

Una **classe** incapsula sia le **caratteristiche** (attributi) sia i **comportamenti** (metodi) degli oggetti che rappresenta.

Inoltre fornisce una **interfaccia pubblica** per poter utilizzare (interagire con) gli oggetti definiti dalla classe.

In altre parole, la OOP combina **strutture dati e algoritmi** in entità software ‘’impacchettate’’ dalla definizione di una classe.

Programmare ad oggetti (OOP)

Esempio: analizziamo l'oggetto ''automobile''

Caratteristiche: velocità, colore, numero di porte, livello del carburante, posizione della marcia

Comportamenti: accelera, fermati, gira (a destra o sinistra), cambia marcia, rifornisciti

Programmare ad oggetti (OOP)

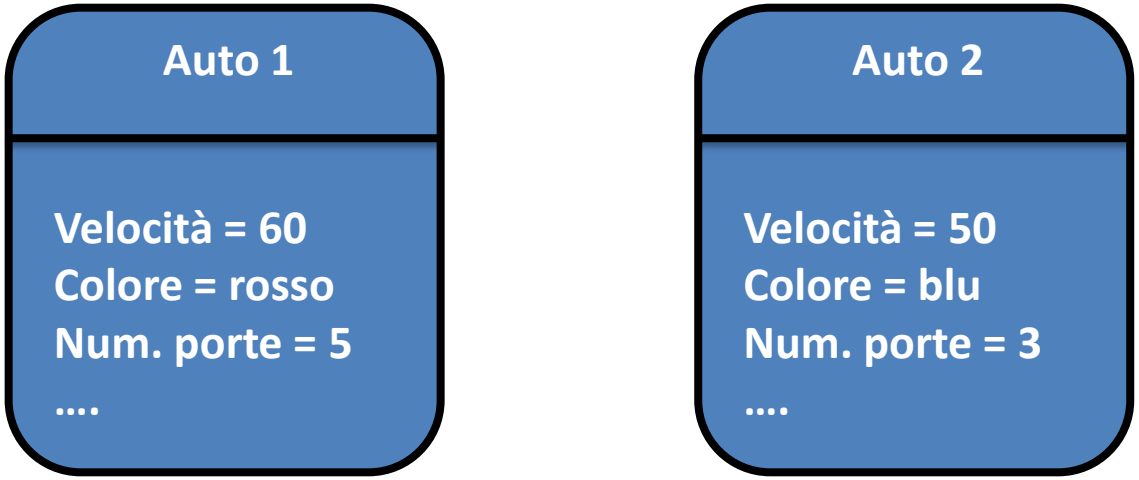
Esempio: analizziamo l'oggetto ''automobile''

Caratteristiche: velocità, colore, numero di porte, livello del carburante, posizione della marcia

Comportamenti: accelera, fermati, gira (a destra o sinistra), cambia marcia, rifornisciti

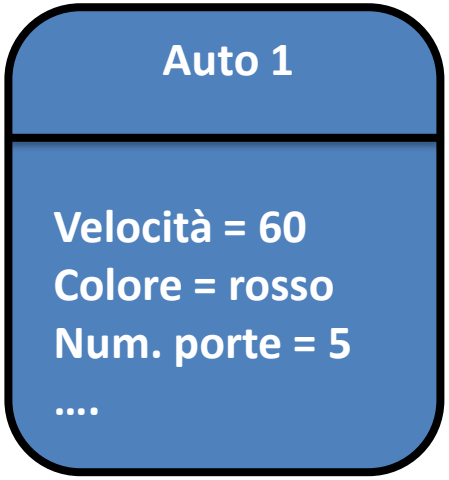
Chi intende utilizzare questo oggetto agisce ***attivando i suoi comportamenti***, questi possono concretizzarsi con delle azioni o con il ***cambiamento dello stato*** dell'oggetto cioè delle sue caratteristiche.

Programmare ad oggetti (OOP)



Programmare ad oggetti (OOP)

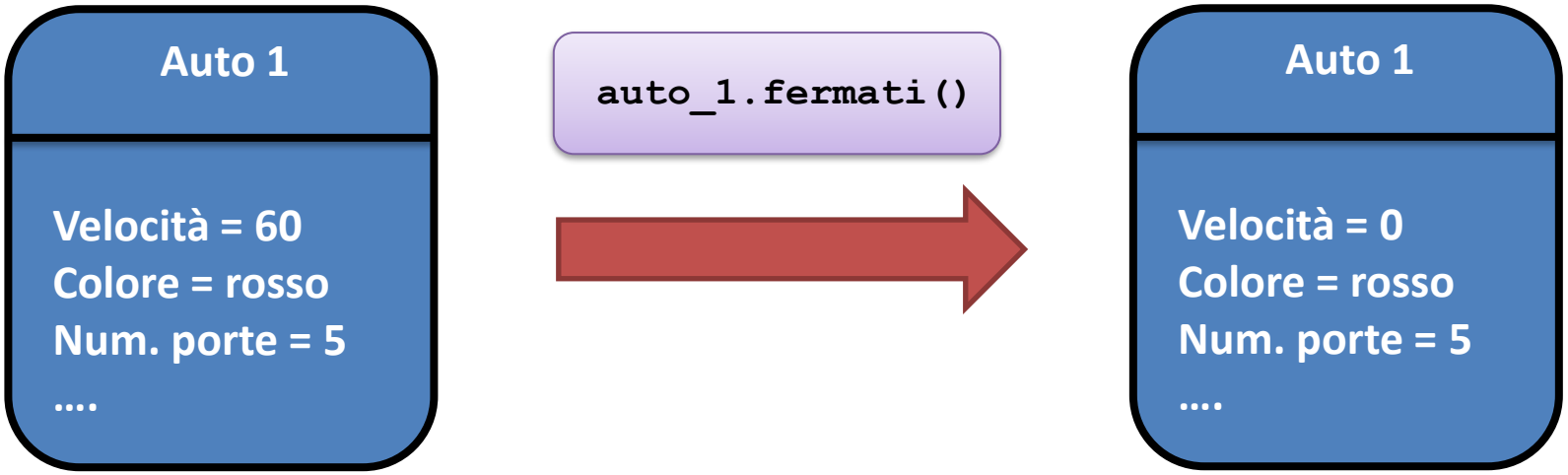
Mediante il metodo ‘’fermati()’’ posso modificare lo stato dell’oggetto auto_1.




```
auto_1.fermati()
```

Programmare ad oggetti (OOP)

Mediante il metodo ‘’fermati()’’ posso modificare lo stato dell’oggetto auto_1.



Programmare ad oggetti (OOP)

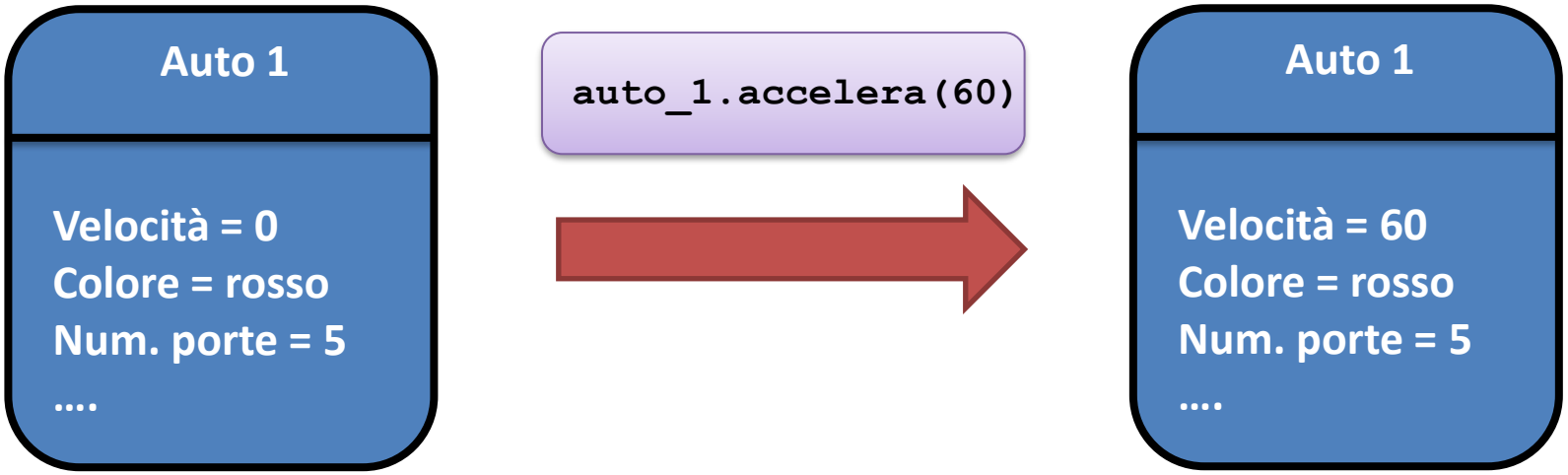
Programmare ad oggetti (OOP)

Mediante il metodo ‘’accelera()’’ posso modificare lo stato dell’oggetto auto_1.


```
auto_1.accelera(60)
```

Programmare ad oggetti (OOP)

Mediante il metodo ‘’accelera()’’ posso modificare lo stato dell’oggetto auto_1.



Programmare ad oggetti (OOP)

Un **oggetto** quindi è formato da **attributi** e **metodi**.

Un **programma ad oggetti** è caratterizzato dalla presenza di **tanti oggetti che comunicano** e interagiscono tra loro (**modello di sistema**).

Nella OOP l'interazione tra oggetti avviene con un meccanismo chiamato **scambio di messaggi**. Un oggetto, inviando un messaggio ad un altro oggetto, può richiederne l'esecuzione di un metodo.

Programmare ad oggetti (OOP)

Un messaggio è costituito da tre parti:

- Un destinatario → l'oggetto verso il quale il messaggio è indirizzato
- Un selettore → identifica il metodo che si vuole attivare
- Un elenco di argomenti → l'insieme dei parametri che vengono passati all'oggetto quando si richiede l'attivazione del metodo

```
destinatario.selettore(elenco di argomenti)

auto_1.fermati()
auto_1.accelera(60)
auto_1.gira(destra)
```

Programmare ad oggetti (OOP)

Adesso che sappiamo cosa è la programmazione OO e cos'è un oggetto, vediamo ora le caratteristiche fondamentali della OOP che la rendono così importante:

- Incapsulamento
- Interfaccia di un oggetto
- Classe
- Ereditarietà
- Polimorfismo
- Collegamento dinamico

Programmare ad oggetti (OOP)

Il termine ***incapsulamento*** indica la proprietà degli oggetti di incorporare al loro interno sia gli attributi che i metodi, cioè le caratteristiche ed i comportamenti dell'oggetto.

Si dice che gli attributi e i metodi sono *incapsulati* nell'oggetto. In questo modo tutte le informazioni utili che riguardano un oggetto sono ben localizzate.

L'incapsulamento non va confuso con l'***information hiding***, che consiste nel nascondere all'esterno i dettagli implementativi dei metodi di un oggetto.

Programmare ad oggetti (OOP)

Riassumendo possiamo dire che:

Un **oggetto** è costituito da un insieme di **metodi** e **attributi incapsulati** nell'oggetto. Gli oggetti interagiscono sfruttando i **messaggi**, che costituiscono **l'interfaccia** dell'oggetto. L'interfaccia non consente di vedere come sono implementati i metodi, ma permette il loro utilizzo.

Le classi ci permettono di definire dei **tipi di dati astratti** (TDA).

Programmare ad oggetti (OOP)

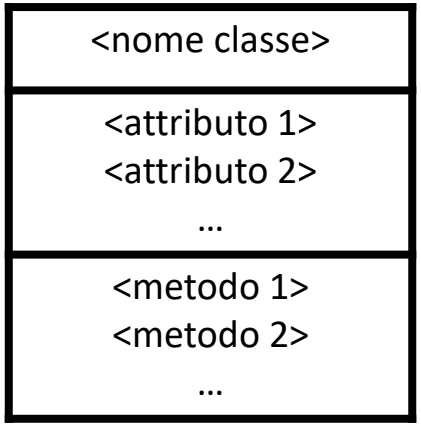
I linguaggi OO permettono di definire una sezione *pubblica* e una sezione *privata* dell'interno dell'oggetto a cui possono fare parte sia gli attributi che i metodi.

Nella **sezione pubblica** vengono messi gli attributi e i metodi che si vuole rendere visibili all'esterno e quindi utilizzabili dagli altri oggetti. Questi costituiscono l'interfaccia.

Nella **sezione privata** ci sono gli attributi e i metodi che non sono accessibili e che vengono usati solo internamente all'oggetto per implementare i suoi comportamenti.

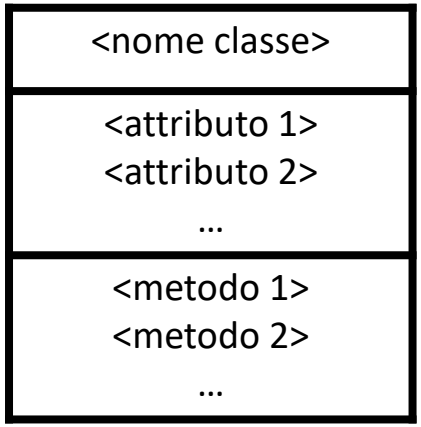
Programmare ad oggetti (OOP)

Per rappresentare graficamente le classi di un progetto software basato sulla OOP si usa il **diagramma delle classi.** La struttura generale è la seguente.



Programmare ad oggetti (OOP)

Possiamo quindi rappresentare la classe automobile con il diagramma della classe:



Programmare ad oggetti (OOP)

Non sempre occorre partire dal nulla nel costruire una classe, soprattutto se si dispone già di una classe che è simile a quella che si vuole costruire. In questo caso si può pensare di **estendere** la classe già esistente per adattarla alle nostre necessità.

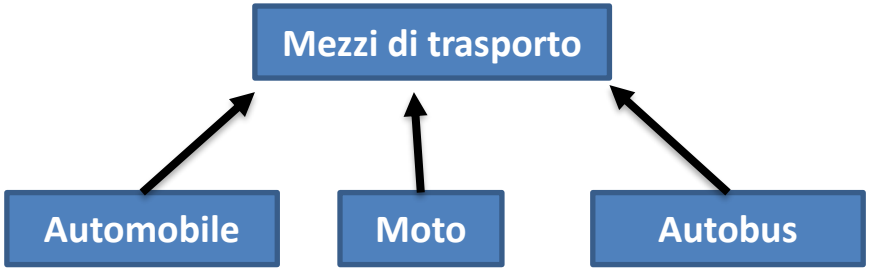
L'**ereditarietà** è lo strumento che permette di costruire nuove classi utilizzando quelle già sviluppate.

Quando una classe viene creata in questo modo, riceve tutti gli attributi ed i metodi della classe generatrice (li eredita). La classe generata sarà quindi costituita da tutti gli attributi e i metodi della classe generatrice più tutti quelli nuovi che saranno definiti.

Programmare ad oggetti (OOP)

La classe che è stata derivata prende il nome di **sottoclasse**, mentre la classe generatrice si chiama **sopraclasse**.

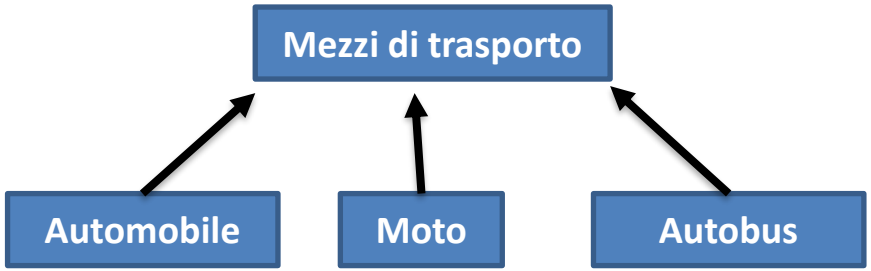
Queste relazioni individuano una gerarchia che si può descrivere usando un **grafo di gerarchia**.



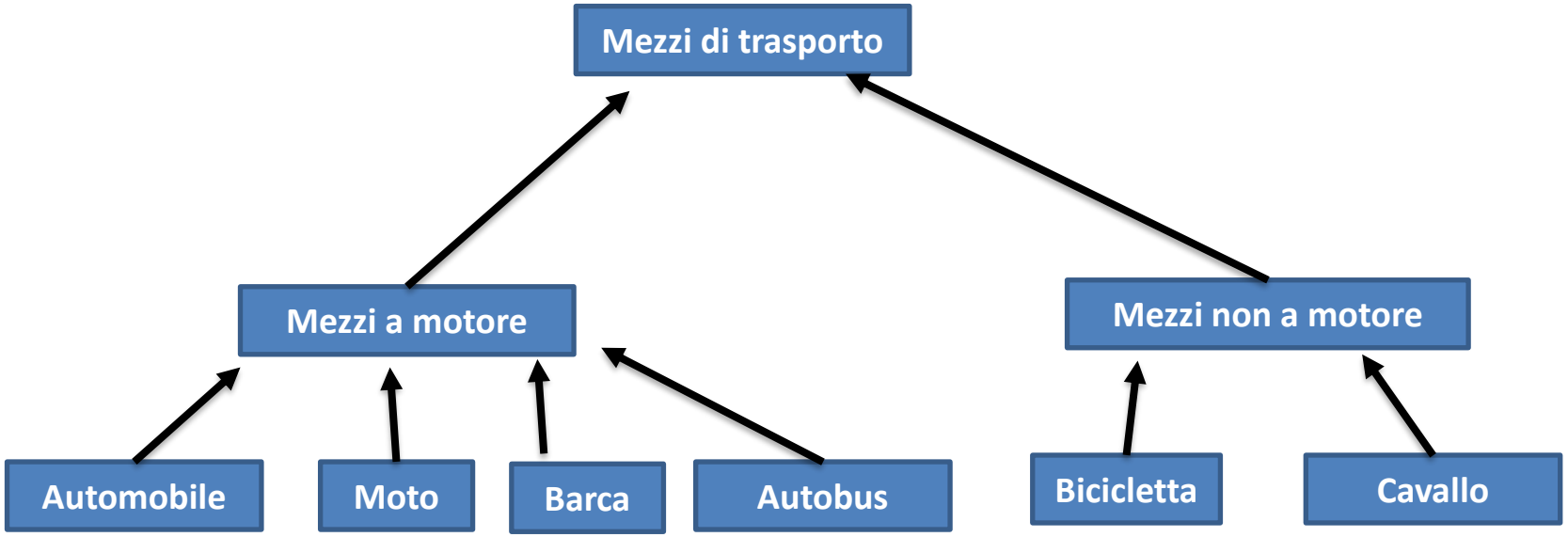
Programmare ad oggetti (OOP)

La nuova classe si differenzia dalla sopraclasse in due modi:

- Per estensione: aggiungendo nuovi attributi e metodi
- Per ridefinizione: modificando i metodi ereditati, specificando una implementazione diversa di un metodo (override, overload)

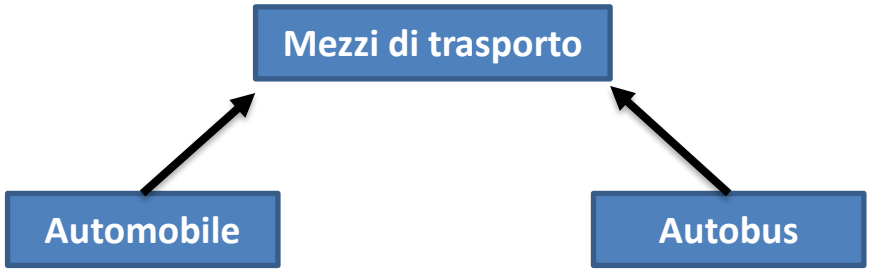


Programmare ad oggetti (OOP)

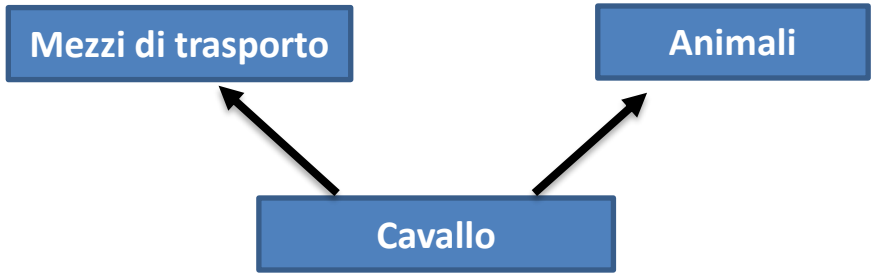


Programmare ad oggetti (OOP)

Ereditarietà singola



Ereditarietà multipla



Programmare ad oggetti (OOP)

Association (relazione d'uso)

Diciamo che una classe A utilizza una classe B se un oggetto di classe A è in grado di inviare dei messaggi ad un oggetto di classe B oppure se un oggetto di classe A può creare, ricevere o restituire oggetti di classe B.

Programmare ad oggetti (OOP)

Association (relazione d'uso)

Diciamo che una classe A utilizza una classe B se un oggetto di classe A è in grado di inviare dei messaggi ad un oggetto di classe B oppure se un oggetto di classe A può creare, ricevere o restituire oggetti di classe B.

Programmare ad oggetti (OOP)

Aggregation (relazione di contenimento)

Un oggetto di classe A contiene un oggetto di classe B se B è una proprietà (attributo) di A.

In sostanza, l'aggregazione è una forma di associazione più forte: una classe ne aggrega un'altra se esiste tra le due classi una relazione di tipo "intero-parte".

Ad esempio la classe Azienda aggrega la classe Persona perché una ditta (che costituisce l'"intero") è composta da persone (che costituiscono la "parte").

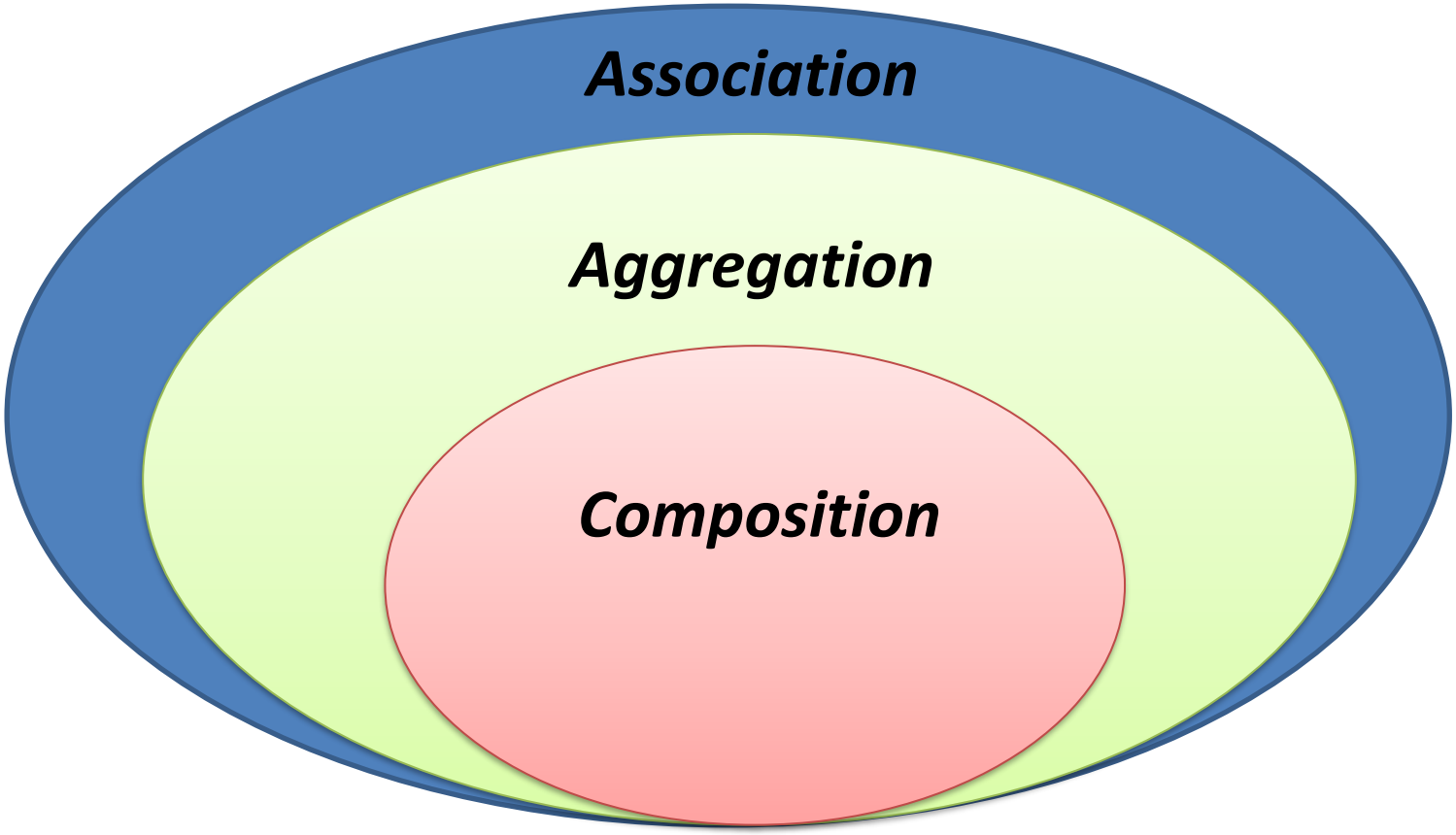
Programmare ad oggetti (OOP)

Composizione

La composizione è una forma di aggregazione ancora più forte che indica che una ‘’parte’’ può appartenere ad un solo ‘’intero’’ in un certo istante di tempo.

Ad esempio uno pneumatico può far parte di una sola automobile in un certo istante, mentre, al contrario una persona potrebbe lavorare contemporaneamente per più aziende.

Programmare ad oggetti (OOP)



Programmare ad oggetti (OOP)

Considerando una relazione di ereditarietà, le sottoclassi hanno la possibilità di ridefinire i metodi ereditati (mantenendo lo stesso nome) oppure lasciarli inalterati perché già soddisfacenti.

Il **polimorfismo** indica la possibilità per i metodi di assumere forme, cioè implementazioni, diverse all'interno della gerarchia delle classi.

Esempio: tutti i veicoli a motore possiedono il metodo ''accelera''. Le sottoclassi ''automobile'' e ''moto'' è probabile che lo ridefiniscano per adeguarlo alle particolari esigenze (es. pedale vs. manopola).

Programmare ad oggetti (OOP)

Durante l'esecuzione del programma, un'istanza della classe ''veicoli a motore'' può rappresentare sia una ''automobile'' che una ''moto''.

Quando viene richiesta l'attivazione del metodo ''accelera'' è importante garantire che, tra tutte le implementazioni, venga scelta quella corretta.

Il **collegamento dinamico** è lo strumento utilizzato per la realizzazione del polimorfismo. È dinamico perché l'associazione tra l'oggetto e il metodo corretto da eseguire è effettuata a ***run-time***, cioè durante l'esecuzione del programma.

Programmare ad oggetti (OOP)

Abbiamo visto le proprietà e le caratteristiche principali della OOP. Questo paradigma di programmazione ha offerto un modo nuovo e potente per scrivere programmi.

Per programmare ad oggetti serve una nuova metodologia con cui affrontare i problemi, che possiamo riassumere nel seguente modo:

- Identificare gli oggetti che caratterizzano il modello del problema
- Definire le classi indicando gli attributi e i metodi
- Stabilire come gli oggetti interagiscono con gli altri