

Lezione 1

Introduzione al corso

Programmazione II

Corso di Laurea Triennale in Informatica

Prof. Misael Mongiovì

misael.mongiovi@unict.it












Prof. Misael Mongiovì



Afferenza	Università degli Studi di Catania Consiglio Nazionale delle Ricerche (CNR)
Istituti e Dipartimenti	Dipartimento di Matematica e Informatica Dipartimento di Scienze Umanistiche Istituto di Scienze e Tecnologie della Cognizione (ISTC-CNR)
Gruppo di lavoro	Semantic Technology Laboratory (STLab)
Sede CNR	Via Gaifami, 18 – Catania
e-mail	misael.mongiovi@unict.it misael.mongiovi@cnr.it
Web site	https://web.dmi.unict.it/docenti/misael.mongiovi https://www.istc.cnr.it/it/people/misael-mongiovi

- Interessi:
 - Intelligenza Artificiale
 - Analisi e gestione dei dati
 - In forma testuale: Elaborazione del Linguaggio Naturale
 - In forma di network: Graph mining e management
- Ricevimento (appuntamento per e-mail)
 - Martedì 17.00 – 18.00
 - Giovedì 17.00 – 18.00
 - Studio 346

Popolarità linguaggi di programmazione Febbraio 2024

 TIOBE <small>the software quality company</small>						
Feb 2024	Feb 2023	Change	Programming Language		Ratings	Change
1	1			Python	15.16%	-0.32%
2	2			C	10.97%	-4.41%
3	3			C++	10.53%	-3.40%
4	4			Java	8.88%	-4.33%
5	5			C#	7.53%	+1.15%
6	7	▲		JavaScript	3.17%	+0.64%
7	8	▲		SQL	1.82%	-0.30%
8	11	▲		Go	1.73%	+0.61%
9	6	▼		Visual Basic	1.52%	-2.62%
10	10			PHP	1.51%	+0.2%

Perché il C++

- Uno dei linguaggi più popolari
- Basato sul paradigma della programmazione orientata agli oggetti (OOP)
- Uno dei linguaggi più completi. Supporta costrutti quali:
 - Ereditarietà multipla
 - Puntatori
 - Template
 - Overloading degli operatori
 - Binding statico e dinamico
 - Namespace
- Possibilità di operare sia basso livello (efficienza) che ad alto livello (astrazione)
- Linguaggio a tipizzazione statica

Non solo C++

- Tecniche di programmazione
- Complessità computazionale
- Implementare algoritmi di ordinamento e ricerca
- Implementare strutture dati sfruttando la OOP

Obiettivi Formativi

- Acquisire metodi di programmazione
 - Imparare a ragionare da informatici
- Acquisire e sviluppare capacità di
 - Risolvere problemi utilizzando gli strumenti della OOP
 - Comprendere le proprietà fondamentali di diversi algoritmi e le strutture dati per essi
 - Implementare gli algoritmi studiati in C++ in modo da ottenere soluzioni affidabili ed efficienti

Perchè programmazione orientata agli oggetti: OOP



- **Incapsulamento:** implementare elementi specifici nascondendo i dettagli implementativi interni
- **Riutilizzo del codice:** Creare nuovi elementi estendendo elementi esistenti
- **Polimorfismo:** trattare oggetti differenti che condividono caratteristiche come se fossero dello stesso tipo
- **Manutenibilità:** Le modifiche a una parte specifica del sistema possono essere effettuate con un impatto minimo sulle altre parti

Canale MS Teams

- Codice del team:
4zm2sra



Testi

- Luis Joyanes Aguilar - Fondamenti di Programmazione in C++ - McGrawHill
- Deitel - C++ Fondamenti di programmazione - Apogeo
- T. H Cormen - Introduction to Algorithms - The MIT Press

Consigliati:

- Deitel - C++ Tecniche avanzate di programmazione - Apogeo
- Algoritmi in C++ (terza edizione) - Pearson Education Italia
- Effective C++ e More Effective C++ Addison Wesley

Attività formative

- 72 ore di lezione frontale
 - 48 ore di teoria
 - 24 ore di laboratorio con il Prof. Massimo Orazio Spata
- L'acquisizione dei metodi e delle competenze professionali della materia è sostenuta da
 - Frequenza delle lezioni
 - Studio individuale (tanto) e soprattutto esercitazione (tanta)
 - Studio del testo
 - Partecipazione al tutorato

Tutorato

- 36 ore
- Orari prestabiliti (si consulti aulario)

Propedeuticità

- Per sostenere l'esame di Programmazione II dovrete aver già superato:
 - Programmazione 1

Programma delle lezioni I

- **Parte 1: Introduzione al linguaggio C++**
 - Elementi di base
 - La programmazione strutturata
 - Le funzioni
 - Operatori ed espressioni
 - Stringhe
 - Flussi di file e stream

Programma delle lezioni II

- **Parte 2: Programmazione avanzata a oggetti**
 - Classi e Oggetti
 - Classi derivate
 - Templates
 - Sovraccaricamento degli operatori

Programma delle lezioni III

- **Parte 3: Strutture Dati**
 - Liste
 - Pile e code
 - Alberi
 - Grafi
- **Parte 1, 2 e 3:**
 - Tantissimi esercizi

Modalità d'esame



Modalità d'esame



1 Prova scritta:

- Durata 1 h
- 30 quesiti a risposta multipla su piattaforma ExamBox in uno dei laboratori del dmi

Modalità d'esame



2 Prova pratica

- Realizzazione di un progetto software su piattaforma ExamBox in uno dei laboratori del dmi
- Compilazione mediante g++ su una macchina generica

Modalità d'esame



3 Orale

- Compilazione ed esecuzione del progetto
- Quesiti sull'implementazione
- Quesiti su tutto il programma didattico

Raccomandazioni

- E' fondamentale acquisire quello che chiamiamo **pensiero computazionale**
 - La capacità di tradurre un problema in una sequenza di istruzioni atte a risolverlo
 - E' alla base della programmazione in qualsiasi linguaggio
- **Perché?**
 - Allena la nostra capacità di **problem solving**. Utile in qualsiasi ambito
 - Il pensiero computazionale è diventato una competenza sempre più importante per molte professioni
 - **Saper programmare** è fondamentale per un informatico
- **Come** si acquisisce il pensiero computazionale?
 - Facendo **pratica**, pratica, pratica, pratica... (svolgendo le esercitazioni e gli homework)
 - Con una forte motivazione: **provare**, **riprovare**, riprovare... anche quando la strada sembra in salita...
 - Evitando le soluzioni semplici: es. farci dare la soluzione dal collega o da uno strumento basato su AI
- Vi supporteremo al meglio, ma siete voi che dovete crederci

Raccomandazioni pratiche

- **Seguire** attentamente le lezioni
- **Non distrarsi** durante la lezione frontale (quando il docente parla)
- Se qualcosa non è chiara chiedere al docente di rivederla
- Svolgere le **esercitazioni** in classe
- Quando il docente richiama la vostra attenzione, anche durante le esercitazioni, interrompere qualsiasi attività in corso ed **ascoltare il docente**
- **Esercitarsi a casa**
- Ascoltare attentamente la **soluzione agli homework** proposta dal docente. Se non è chiara, chiedere al docente di rispiegarla
- Tenere in mente che un problema può avere più soluzioni diverse, tutte altrettanto valide
 - Se il programma termina senza errori e fornisce la risposta corretta, questo è un buon segno ma non una prova di correttezza, in quanto ad es. cambiando i dati di input il programma potrebbe non funzionare più

Domande?

Introduzione alla OOP

(Object Oriented Programming – Programmazione orientata agli oggetti)

Sommario

- Il concetto di astrazione
- Oggetti e classi
- Programmazione orientata agli oggetti OOP
 - Programmazione procedurale vs. OOP
 - Processo di sviluppo Object Oriented
- Concetti fondamentali della OOP

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...

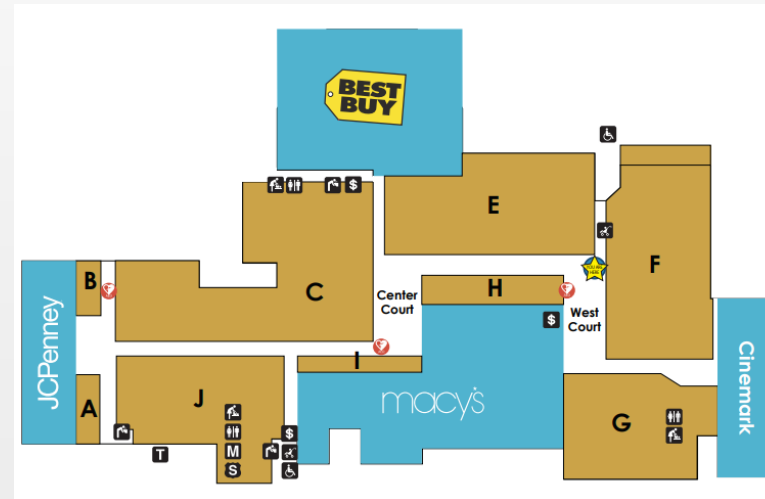
- Una piantina stradale rappresenta una astrazione di una città.



L'importanza dell'astrazione

Esempi di astrazione...

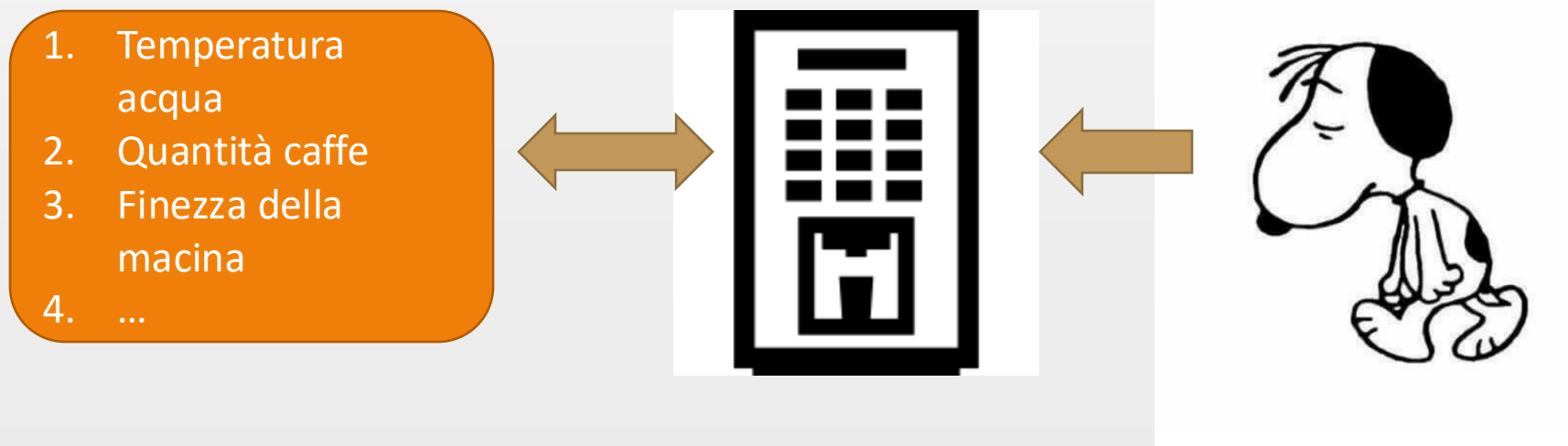
- Una piantina stradale rappresenta una astrazione di una città.



L'importanza dell'astrazione

Esempi di astrazione...

- Una piantina stradale rappresenta una astrazione di una città.
- Come faccio per avere un caffè ?



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.

Oggetti

Si tratta di un concetto fondamentale nella programmazione ad oggetti.

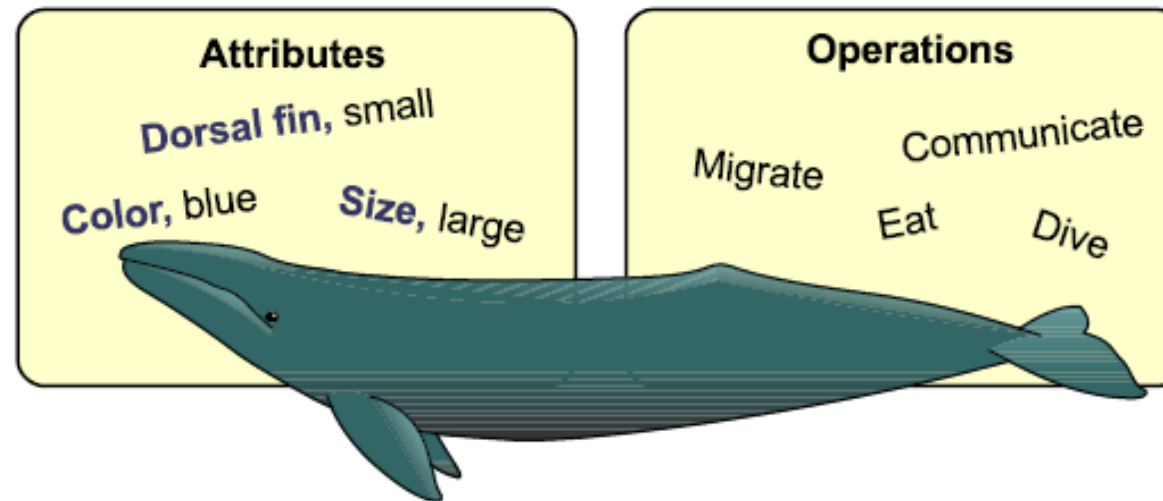
Gli **oggetti** in un linguaggio OOP forniscono la funzionalità di astrarre, cioè di nascondere i dettagli implementativi interni.

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.

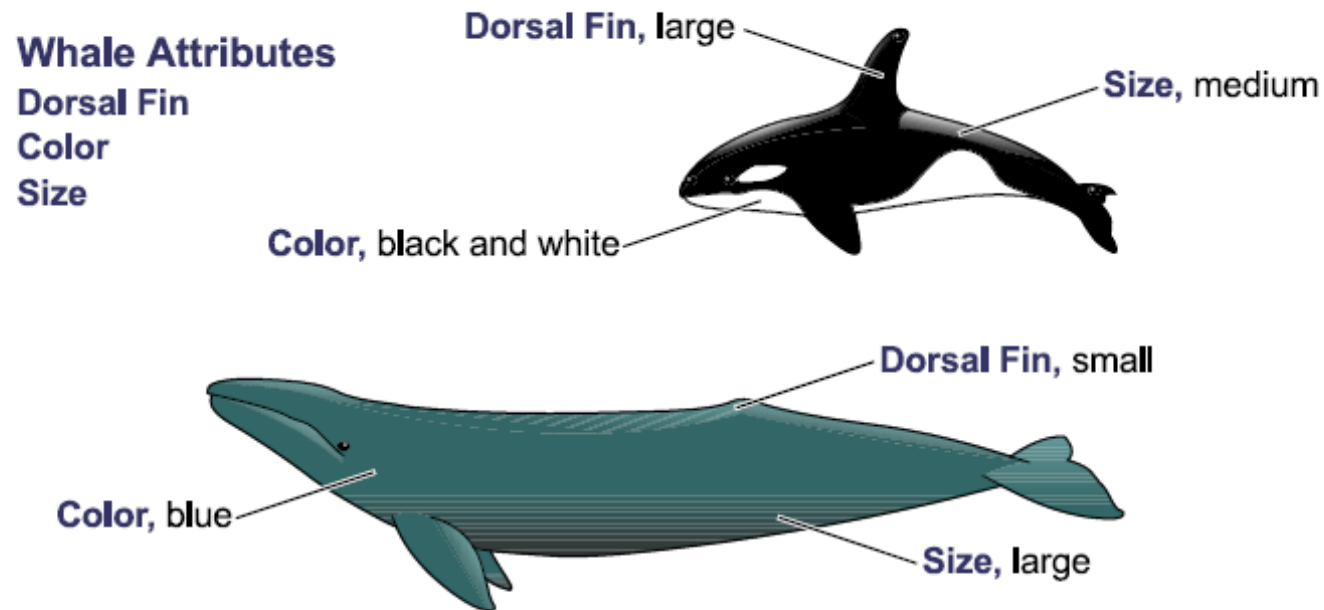
Identificare gli Oggetti

- Gli oggetti possono essere fisici oppure modelli concettuali.
- Gli oggetti possono avere *attributi* (caratteristiche), come size, name, shape, e via dicendo.
- Gli oggetti possono avere *operazioni* (ovvero le operazioni che essi possono compiere), come ad esempio settare un valore, mostrare a screen un risultato, oppure incrementare il valore di una variabile come ad esempio la velocità.

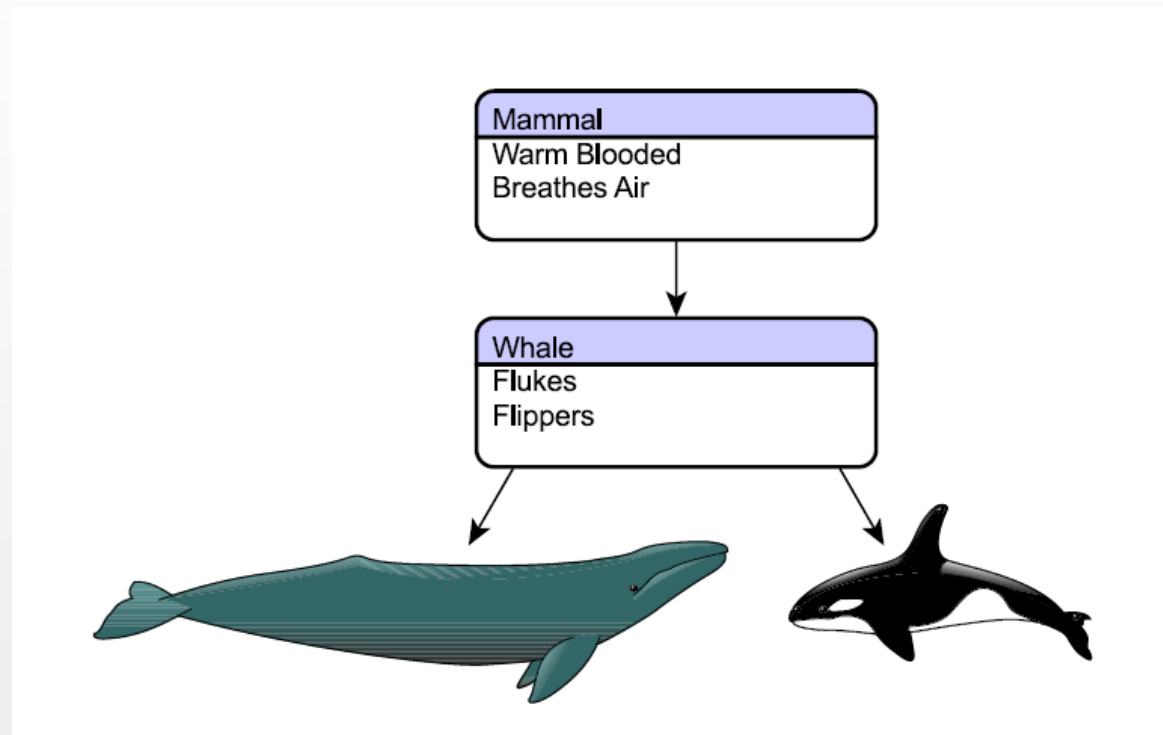
Identificare gli Oggetti



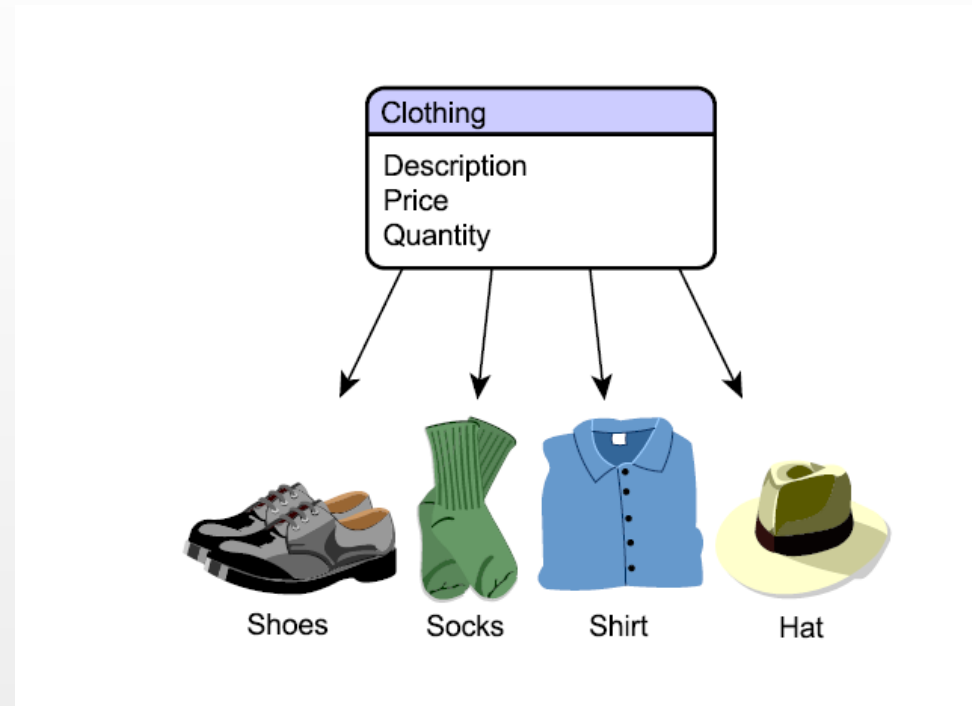
Progettare le Classi



Superclassi e Sottoclassi



Testare le relazioni comuni tra sottoclassi



Sommario

- Il concetto di astrazione
- Oggetti e classi
- Programmazione orientata agli oggetti OOP
 - Programmazione procedurale vs. OOP
 - Processo di sviluppo Object Oriented
- Concetti fondamentali della OOP

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

Headers
Variabili globali
$f()$
$g()$
$h()$
...
$x()$

Programmazione procedurale vs. OOP

Programmazione procedurale

Problema complesso



Scomposizione in
procedure

Programmazione ad oggetti

Sistema complesso



Scomposizione in
entità interagenti
(oggetti)

Programmare ad oggetti (OOP)

Esempio: interfaccia grafica (GUI) di un PC

Componenti

Finestre (proprietà: dimensione, posizione)

Bottoni (proprietà: colore, testo)

Interazioni

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, dei componenti 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.

Analisi

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 **elementi** che compongono il sistema e **le interazioni tra essi**.

Design

È importante creare programmi che siano **flessibili**, ovvero facili da estendere.

Durante il **design**, lo scopo è avere sw che sia **altamente coeso** ma allo stesso tempo con un accoppiamento lasco (**loose coupling**), ovvero che un cambiamento su una componente non implichi modifiche su un'altra.

Questo si ottiene **decomponendo il problema** in moduli, determinando le relazioni tra essi, identificando dipendenze e forme di comunicazione.

Oggetti

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

Classi

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.

Esempio di OOP

Esempio: analizziamo l'oggetto "automobile"

Caratteristiche: ...

Esempio di OOP

Esempio: analizziamo l'oggetto "automobile"

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

Esempio di OOP

Esempio: analizziamo l'oggetto "automobile"

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

Comportamenti:

Esempio di 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

Esempio di OOP

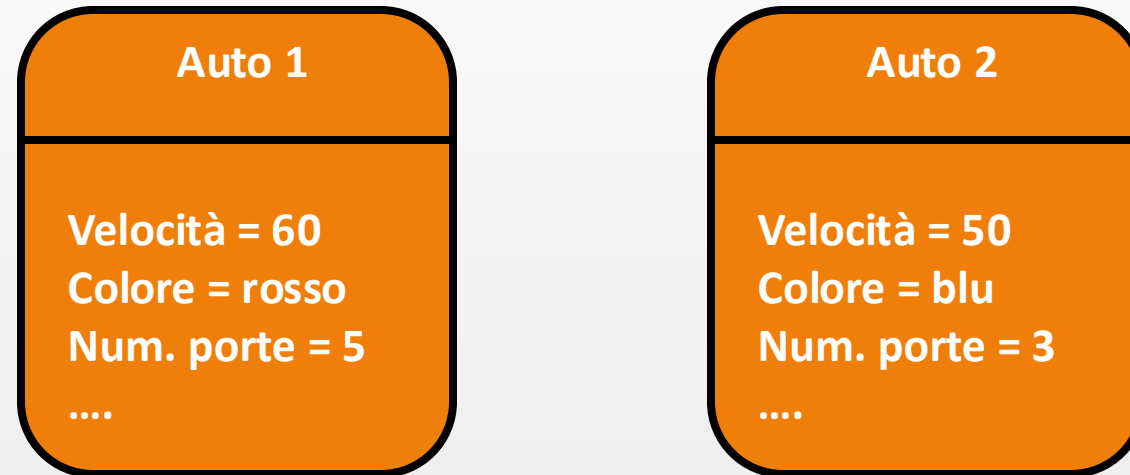
Esempio: analizziamo l'oggetto "automobile"

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.

Esempio di OOP



Esempio di OOP

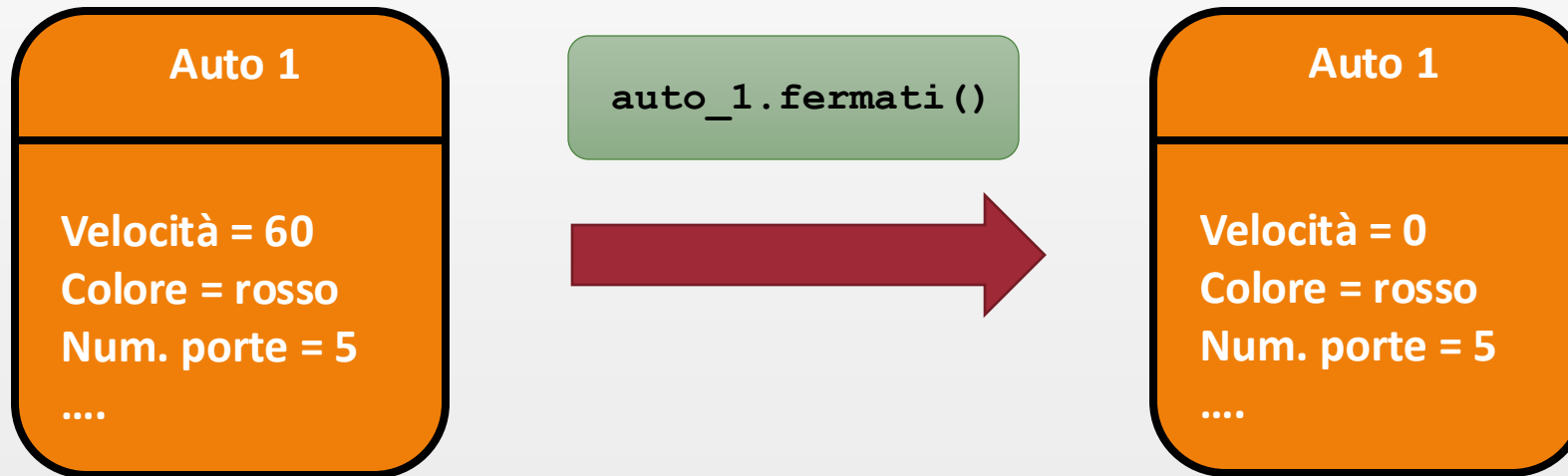
Mediante il metodo “fermati()” posso modificare lo stato dell’oggetto auto_1.



`auto_1.fermati()`

Esempio di OOP

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



Esempio di OOP



Esempio di OOP

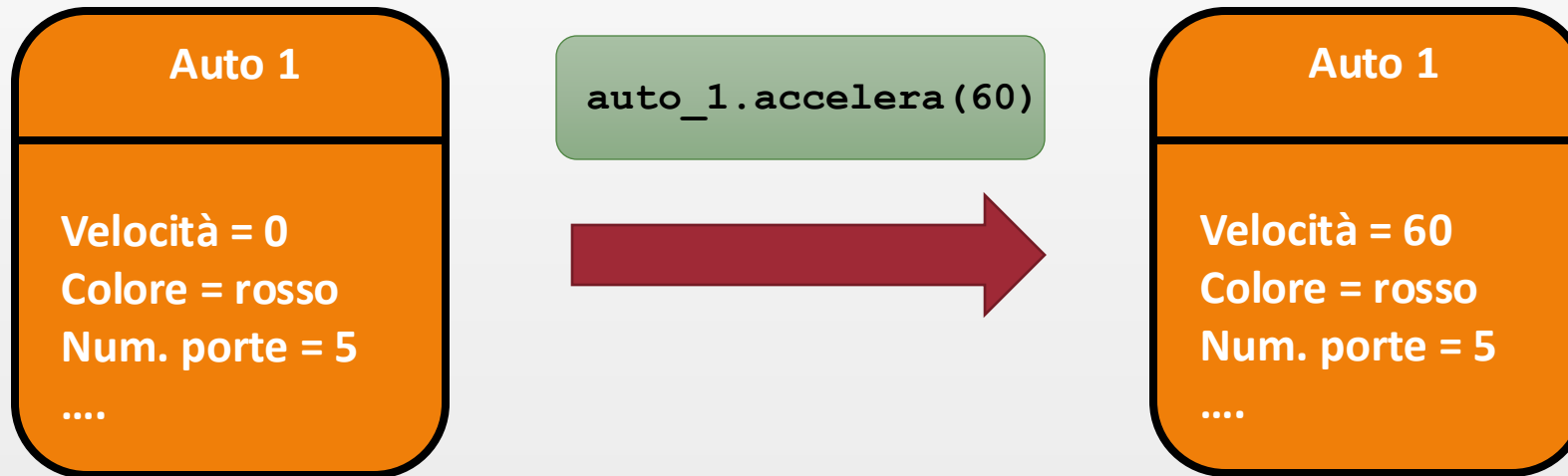
Mediante il metodo “accelera()” posso modificare lo stato dell’oggetto auto_1.



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

Esempio di OOP

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



Riassumendo

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.

Sommario

- Il concetto di astrazione
- Oggetti e classi
- Programmazione orientata agli oggetti OOP
 - Programmazione procedurale vs. OOP
 - Processo di sviluppo Object Oriented
- **Concetti fondamentali della OOP**

Concetti fondamentali della OOP

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

- Incapsulamento e occultamento
- Ereditarietà
- Polimorfismo

Incapsulamento e occultamento

- 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.
- L'***occultamento*** consiste nel nascondere all'esterno i dettagli implementativi dei metodi di un oggetto.
- Un **oggetto** è costituito da un insieme di **metodi** e **attributi incapsulati** nell'oggetto. Gli oggetti interagiscono sfruttando i **metodi**, 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**.

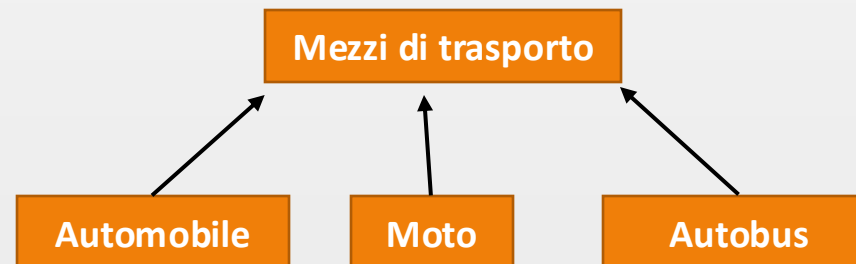
Ereditarietà

- 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.

Ereditarietà – gerarchie di classi

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

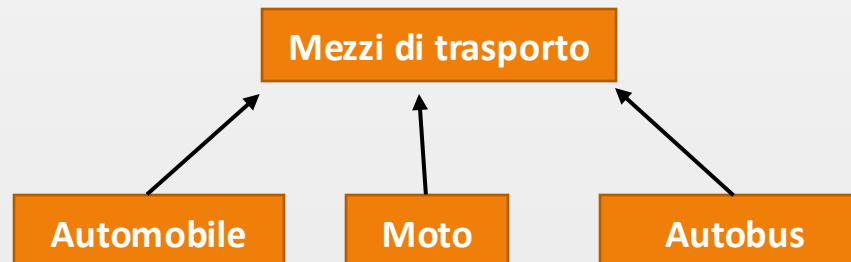
Queste relazioni individuano una gerarchia che si può descrivere usando un **grafo di gerarchia**.



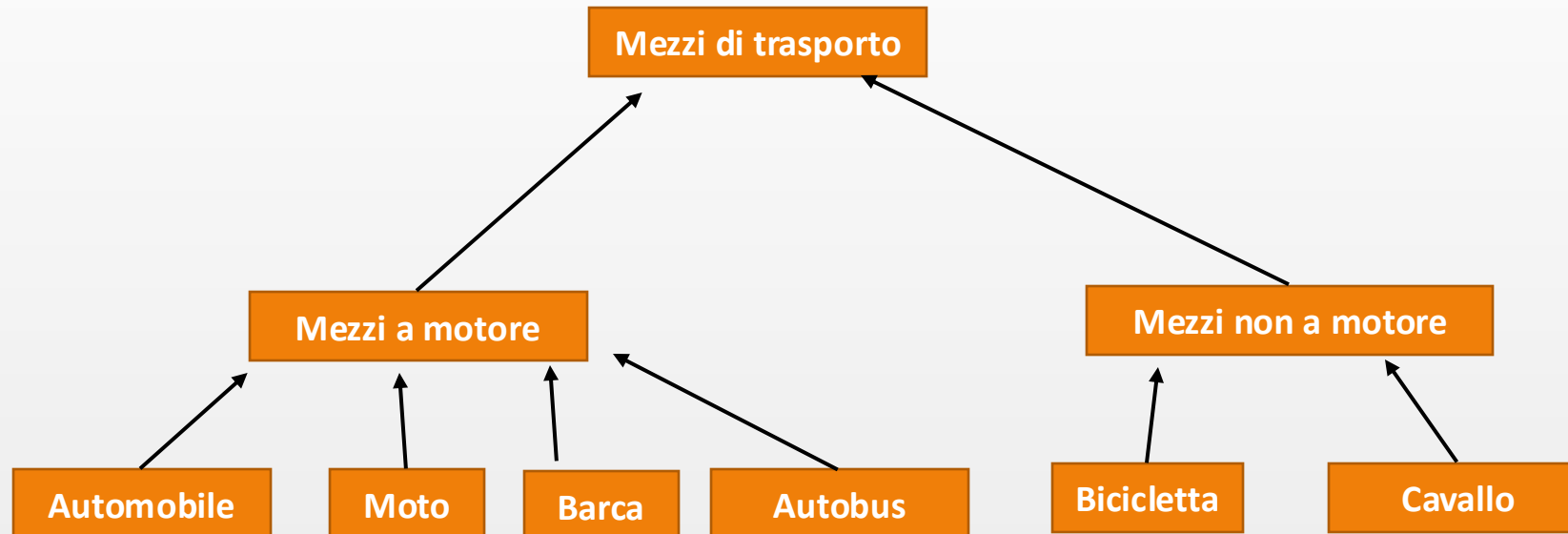
Ereditarietà - overriding

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**)



Ereditarietà a più livelli

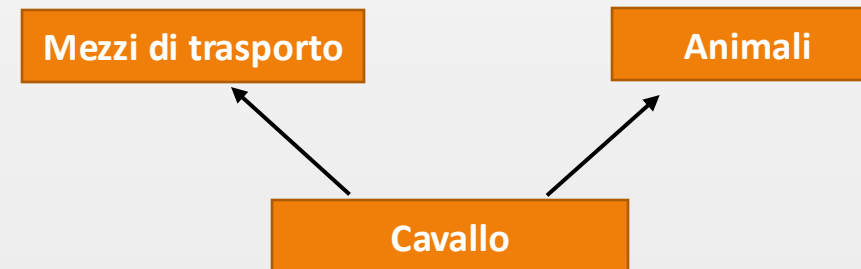


Ereditarietà multipla

Ereditarietà singola



Ereditarietà multipla



Polimorfismo

- 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).

Polimorfismo e binding dinamico

- 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 **binding 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.

Riassumendo

- **OOP**: Object Oriented Programming; Programmazione orientata agli oggetti
- **Astrazione**: capacità di considerare gli elementi importanti di un sistema in relazione alla specifica applicazione.
- **Incapsulamento e occultamento**: raccogliere **attributi** e **metodi** in una stessa **classe**, esporre alcuni metodi come **interfaccia** e nascondere i dettagli implementativi
- **Ereditarietà**: estendere classi ereditando tutte le caratteristiche in una nuova classe (**sottoclasse**)
- **Polimorfismo**: implementare comportamenti diversi per uno stesso metodo nelle sottoclassi