Programmazione a Oggetti (OOP)

In Breve...

- **OOP** Object Oriented Programming
- **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
- **EREDITARIETA'** estendere classi ereditando tutte le caratteristiche in una nuova classe (sottoclasse)
- **POLIMORFISMO** implementare comportamenti diversi per uno stesso metodo nelle sottoclassi
- **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

Astrazione

I software di grandi dimensioni, come ad esempio i giochi, possono essere molto difficili da creare, qui entra in gioco l'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.**

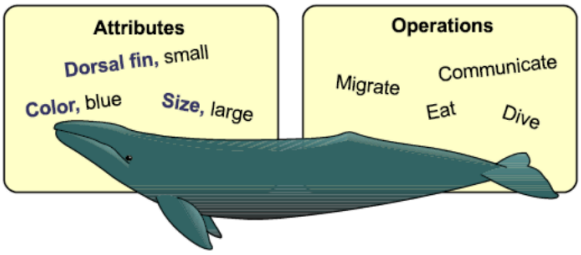
Esempi:
Una cartina rappresenta l'astrazione di una città:

Oggetti e classi

- 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.

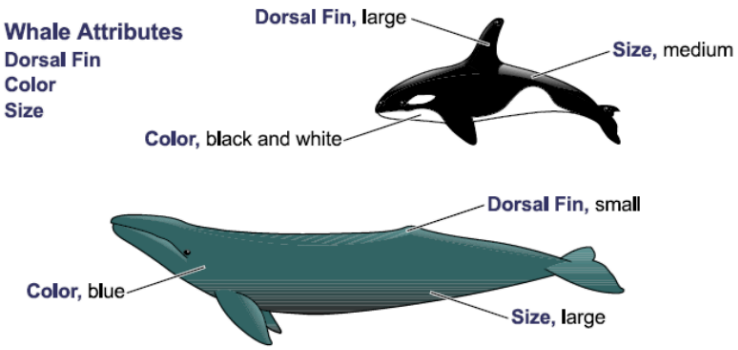
• Gli oggetti possono essere fisici oppure modelli concettuali.
• Gli oggetti possono avere attributi (caratteristiche), come size, name, shape, e via dicendo.
• Gli oggetti possono avere operazioni (ovvero le operazioni che essi possono compiere), come ad esempio settare un valore, mostrare a screen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

ESEMPIO:

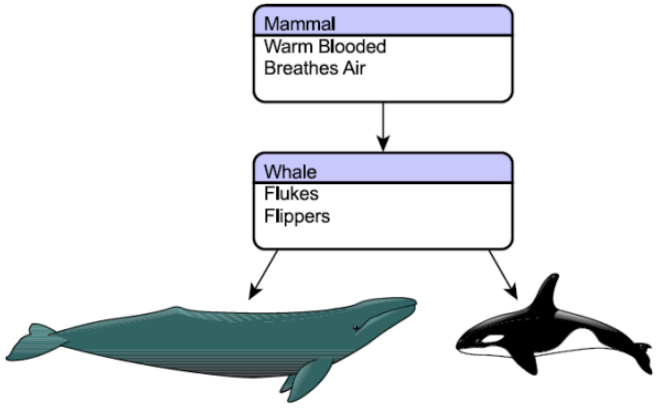


Le **classi** sono un insieme di oggetti dello stesso tipo. Quindi...
Una classe incapsula sia le caratteristiche (attributi) sia i comportamenti (metodi) degli oggetti che rappresenta.

ESEMPIO:



Posso creare delle **superclassi** e delle **sottoclassi**
ESEMPIO:



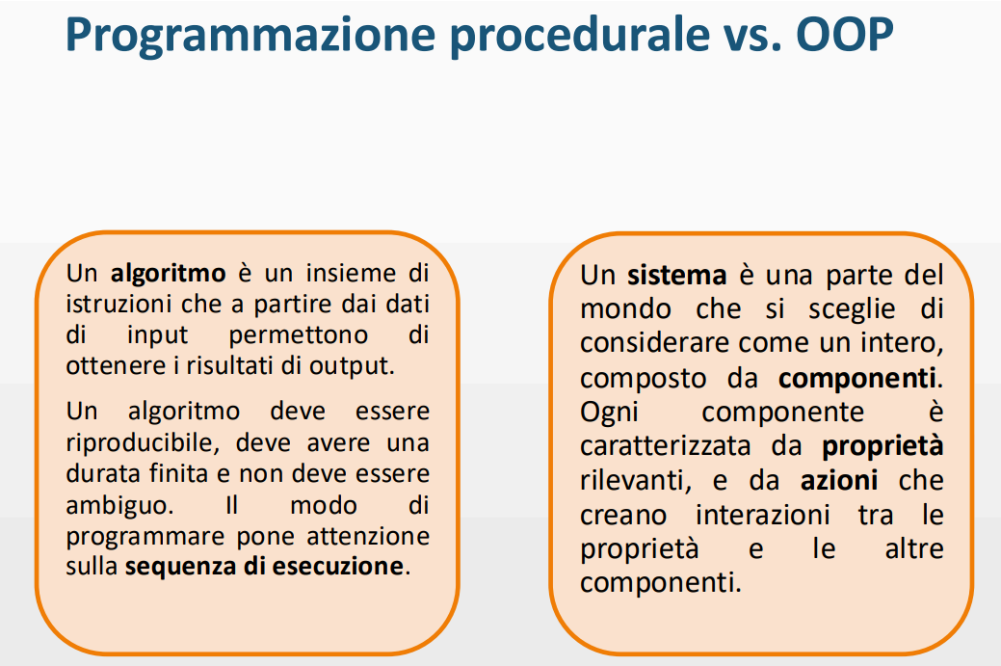
Programmazione orientata agli oggetti OOP

- **Programmazione procedurale vs. OOP**

  Nello sviluppo software, usando la metodologia della programmazione procedurale (come in **C**), 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 (come in **C++**). 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.



- 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 breve...

Programmazione a oggetti:
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)

Possiamo individuare 3 fasi principali nella **programmazione a oggetti OOP**

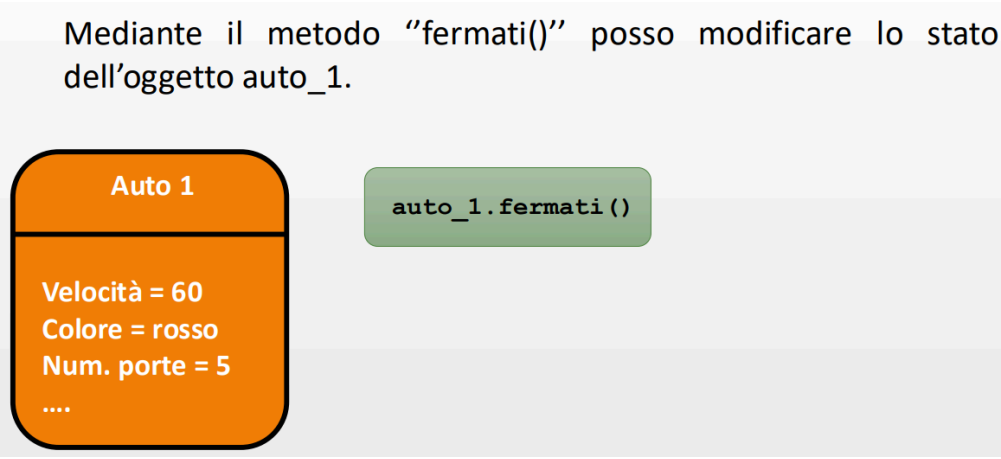
1. **Analisi**: identificazione dei requisiti funzionali e delle loro relazioni. (analisi del problema e identificazione dei vari componenti da utilizzare)
2. **Design**: specificazione delle gerarchie tra le classi e delle loro interfacce e comportamenti
3. **OOP**: momento dell'effettiva programmazione e test

ANALISI
Questa è la fase in cui si analizza il sistema in cerca di una soluzione. Cercando i componenti e capendo come essi devono interagire (può essere paragonato ad una fase di progettazione)

DESIGN
Questa fase ha il compito di rendere il programma software flessibile.
Quindi il programma deve essere coeso ma allo stesso tempo deve gestire il fatto che un cambiamento ad un componente non implichi altri cambiamenti al di fuori di quel componente

OOP
Qui si programma e si eseguono i test

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



Concetti fondamentali della OOP

Caratteristiche
• Incapsulamento e occultamento

• Ereditarietà
• Polimorfismo

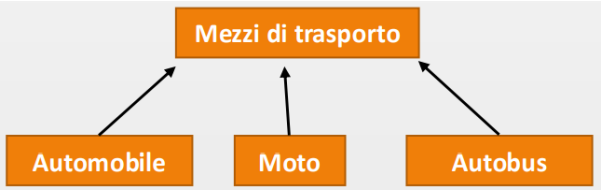
INCAPSULAMENTO E OCCULTAMENTO
Il termine incapsulamento indica la proprietà degli oggetti di incorporare al loro interno le caratteristiche ed i comportamenti dell'oggetto.

L'occultamento consiste nel nascondere all'esterno i dettagli implementativi dei metodi di un oggetto.

In pratica...
• 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.

EREDITARIETA'
• 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.

Le classi generate si chiamano **sottoclassi** mentre quelle che generano si chiamano **superclassi** e possiamo rappresentarle con un grafo:



La nuova classe si differenzia dalla sopraclasse in due modi:
• Per **estensione**: aggiungendo nuovi attributi e metodi
• Per **ridefinizione**: modificando i metodi ereditati, (override, overload)

POLIMORFISMO
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).

• 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.