

## Cenni di Modellazione UML

Un modello è un'estrazione che cattura le proprietà salienti della realtà. Se dovessimo costruire modelli completamente uguali alla realtà, avremmo molte caratteristiche del modello inutili. La realtà, quindi, è molto più dettagliata del nostro modello. Il nostro obiettivo, quindi, è di avere un modello semplificato, ma questo deve essere fedele agli aspetti che ci interessano.

### Perché modellare?

Un aspetto fondamentale è sicuramente quello della comprensione della "materia". Un altro aspetto importante è quello di modellare con lo stesso linguaggio. In modo da favorire la collaborazione. Inoltre il tipico progetto software ha un continuo ricambio di personale; è quindi essenziale modellare con uno schema preciso.

### I linguaggi di modellazione

Tramite delle primitive che il linguaggio offre, possono effettuare un'estrazione della realtà. Dobbiamo capire quindi come i costrutti del linguaggio sono composti. I processi, invece, descrivono i passi da intraprendere per ottenere un risultato finale.

### UML - Unified Model Language

E' basato su una visualizzazione grafica sviluppato per realizzare sistemi complessi. E' inoltre detto unified perché prima di UML tutte le tecniche di modellazione erano strutturate, ma adottate singolarmente.

Qualsiasi artefatto di cui abbiamo bisogno nel nostro sistema può essere modellato grazie ad UML. Esso si articola su un numero di costrutti molto ampio. Alcune delle soluzioni non sono sempre usate. UML è indipendente dal linguaggio di programmazione.

Si parla di un vero e proprio linguaggio: ogni aspetto grafico ha un significato ben preciso; alterare la notazione grafica ci porta a modelli privi di senso.

UML fornisce supporto a tutte le fasi dello sviluppo:

- **Analisi:** cosa deve fare il sistema
- **Progettazione:** ci consente di definire le soluzioni progettuali.
  - **Class diagram:** ci fa copiare la struttura del sistema
  - **Sequence Diagram:** ci fa copiare come gli elementi del sistema si relazionano tra loro.
- **Implementazione:**
- **Testing:**

I costrutti principali (class diagram e sequence diagram) sono gli stessi per tutte le fasi della progettazione. Dobbiamo quindi essere consapevoli di quale fase ci troviamo.

### Concetti fondamentali

- **Astrazione:** Usare le classi per astrarre le caratteristiche di un oggetto del mondo reale.
- **Incapsulamento:** Nascondere al mondo esterno i dettagli di implementazione di un dato oggetto; Ci interessa sapere solo come questo oggetto si interfaccia con l'esterno.
- **Ereditarietà:** Le classi possono specializzare altre classi.
- **Polimorfismo:** Invocare comportamenti diversi a seconda del tipo di invocazione.

### Diagrammi e viste

UML si organizza in diagrammi: organizzati in 4 viste + 1:

- **Vista logica:** Mette in risalto le viste logiche del sistema tramite classi ed oggetti.

- **Sviluppo:** Mostra l'organizzazione del sistema in blocchi strutturali: **Package, Sottosistemi**
- **Processi:** mostra i processi e thread in esecuzione del Sistema
- **Fisico:** mostra in quali luoghi del calcolatore il Sistema viene fisicamente installato.

Tutte queste viste vengono legate da una vista **use case** che fa da collegante tra tutte le viste.

## Diagrammi forniti da UML

**Struttura:**

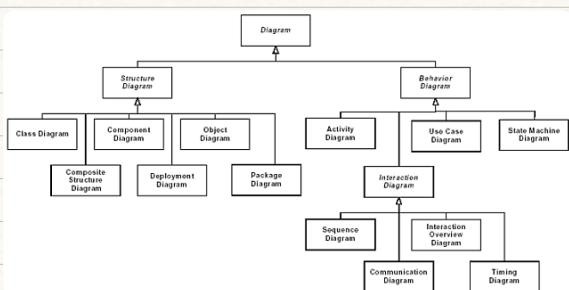
- Class Diagram
- Object Diagram
- Package Diagram
- Composite Structure Diagram
- Component Diagram
- Deployment Diagram

**Comportamento:**

- Use Case Diagram
- Activity Diagram
- State machine Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Diagram
- Timing Diagram

Diagrammi

Approfonditi in  
questa lezione.



Tassonomie di UML

## Class Diagram

Un Class Diagram è usato per modellare:

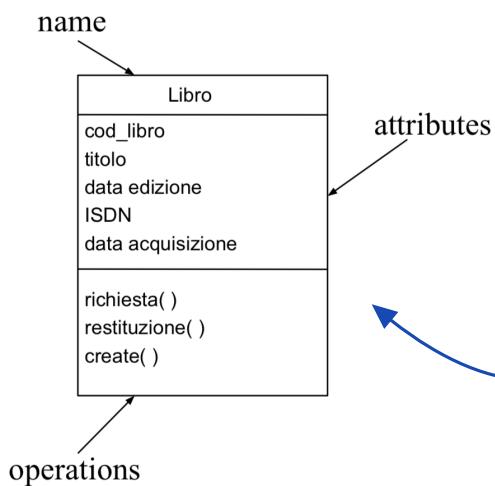
- **Dominio del Sistema:** Come gli elementi di quel sistema interagiscono e sono strutturati
- **Astrazioni Visuali:** Astrazioni da considerare
- **Schema concettuale di un DB:** molto usato per i modelli di DB ad oggetti
- **Architettura e Struttura**

Quando costruiamo un CD dobbiamo dare un **nome significativo** alla nostra classe. Deve quindi comunicare il suo **Scopo**.

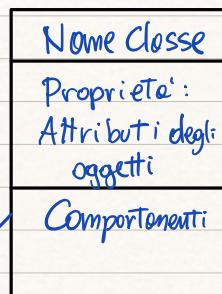
Quando creo un modello, questo è estremamente ricco; fare un diagramma **unico** è impensabile per questione di dimensioni. Di conseguenza i **diagrammi** ci consentono di realizzare delle viste parziali di un **modello molto più grande**. Quindi, quando costruiamo un diagramma pensiamo che stiamo comunicando un **esposto a qualcuno**: non possiamo quindi creare dei diagrammi con migliora di dettagli, ma dobbiamo concentrarci su solo alcuni dettagli fondamentali.

## Concetto di Classe

Il class Diagram si fonda sul concetto di classe:

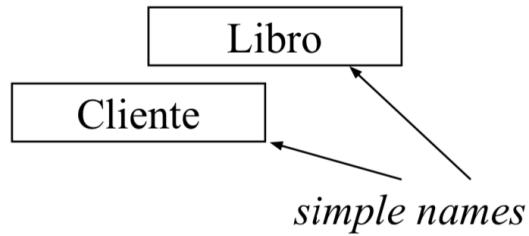


Una classe in UML è rappresentata con un rettangolo diviso in 3 comparti:

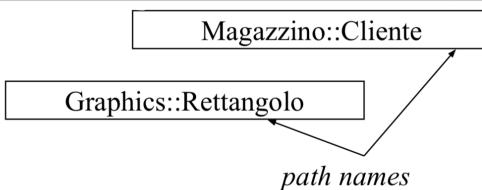


Dopo aver definito la classe è possibile instanziare oggetti di quella det. Classe.

## Rappresentazione Sintetica



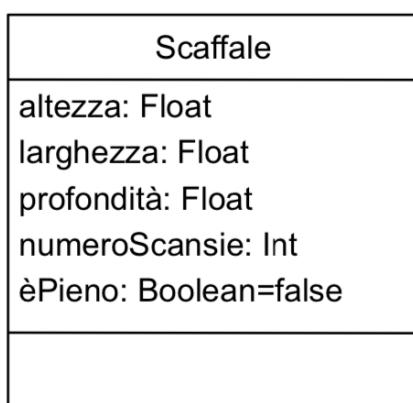
E' possibile anche rappresentare una classe usando il nome della classe. Se esiste ambiguità si possono usare nomi qualificati. Possiamo quindi inserire una classe all'interno di un package:



Nomi con Package

## Attributi

Gli attributi specificano il tipo dell'attributo e la sua molteplicità.



Questa classe ci dice che sarà composta da diversi attributi, tra cui ad esempio èPieno, di tipo intero ed inizializzato a False.

## Operazioni

SensoreTemperatura
reset()
setAlarm(t:Temperatura)
leggiVal():Temperatura

Le operazioni sono inserite nel 3° Comparto e possono avere n parametri; è in questo Comparto che referenziamo le altre classi presenti nel modello. Questa caratteristica ci permette di collegare diverse classi mediante diverse relazioni.

In questo caso si nomina un'altra classe (Temperatura).

All'interno di questo comporto possono essere ancora più precisi ed usare degli Stereotipi che ci consentono di organizzare i gruppi di operazioni che hanno un ruolo ben preciso:

Agente Finanziario
.....
<<costruttore>>
new()
new(p: Polizza)
.....
<<query>>
èSolvibile(o:Ordine)
èEmesso(o:Ordine)

Questo meccanismo è un meccanismo intrinseco di UML e si possono applicare a qualsiasi elemento (classi, metodi, ecc.). Gli Stereotipi consentono di alterare il significato che un determinato elemento ha.

Uno Stereotipo ci chiarisce la Semantico con cui devo interpretare quell'elemento.

## Classificatore

Un classificatore è un meccanismo che descrive fattori strutturali. È possibile inserire vari elementi che riguardano la visibilità delle strutture interne. Questi elementi ci dicono un dato elemento se può essere visibile a tutti, o se solo in alcuni contesti:

Libro
+ cod_libro
+ titolo
# data edizione
ISDN
#richiesta()
restituzione()
+ create()

Se ad esempio troviamo il simbolo +, allora la visibilità è pubblica; Quando invece troviamo il simbolo #, la classe è visibile solo nello stesso package (protected). Se non c'è presente alcun simbolo, la classe è public.

## Multiplicità

NetworkController
consolePort [2..*]: Port

ControlRod
4

La classe NetworkController può essere instanziato una Sola volta (particolare pattern)

Per ogni attributo possiamo specificare la multiplicità indicando un bound inferiore e superiore, e questo specifica il numero di istanze in stanzieribili per quell'attributo.

L'attributo consolePort deve almeno essere instanziato 2 volte, con un max di oo.

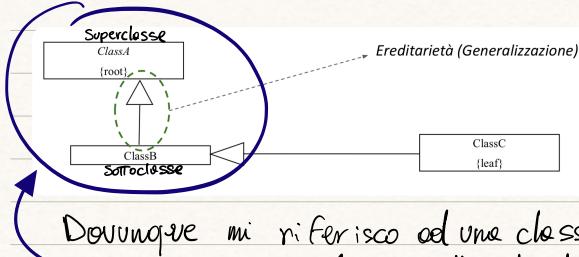
control Rod deve avere 4 istanze Al massimo.

# Attributi ed operazioni

[visibilità] nome [[molteplicità]] [:tipo] [= valore iniziale] [{stringa di proprietà}] ← → Sintassi Completa

- **Changeable**: nessuna limitazione per la modifica del valore dell' attributo.
- **addOnly**: Si possono aggiungere valori per attributo, ma non cambiati o rimossi.
- **Frozen**: Il valore non puo' essere modificato dopo la sua inizializzazione.

## Ereditarietà



E' possibile derivare da concetti piu' astratti (**Superclassi**) delle **Sottoclassi** che rappresentano concetti piu' specifici.

Dovunque mi riferisco ad una classe A, posso trattare anche oggetti di classe B; in quanto i comportamenti di B sono un'estensione di A.

Gli oggetti di una **sottoclasse**, saranno sempre considerati anche oggetti della **Superclasse**.

Non dobbiamo abusare dell'ereditarietà: deve esserci un'effettiva specializzazione della superclasse.

**Comportamento Polimorfico**: tutti i comportamenti delle sottoclassi che sussurrano quelli delle superclassi, generano un comportamento Polimorfico. Se ho un oggetto di A, ocui invio un messaggio, quell'oggetto si comporterà come viene specificato all'interno della classe di riferimento A.

Se ho un oggetto di B, che riceve lo stesso messaggio di A, il comportamento sarà diverso.

## Relazioni



da Freccia punte al genitore

\* E' importante rappresentare la freccia esattamente in questo modo!

In UML è possibile anche usare l'ereditarietà multiple: un singolo Figlio può ereditare da più genitori.

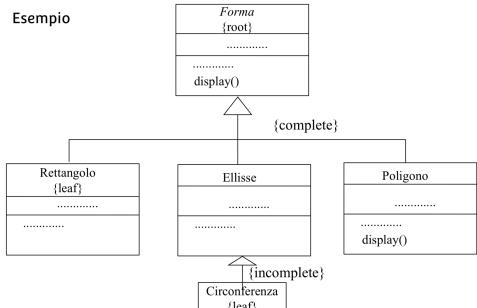
**UML** definisce alcuni stereotipi fondamentali per la generalizzazione:

**1 Stereotipo**: Implementazione se la sottoclasse implementa l'implementazione della superclasse ma non la rende pubblica e non supporta la sua interfaccia. Questo significa usare l'**'ereditarietà privata'**

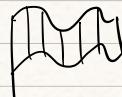
### 4 Vincoli

- **Complete**: tutte le sottoclassi sono state specificate e nessun'altra sottoclasse è permessa. (Se ho aggiunto ricade in una delle sottoclassi)
- **incomplete**: non tutte le classi sono state create; ne sono messe altre.
- **Disjoint**: gli ogg. del genitore non possono avere più di un figlio come tipo.
- **Overlapping**: ogg. del genitore possono avere più di un figlio come tipo

### Esempio



Abbiamo 3 sottoclassi della classe Forma. La s.c. Rettangolo è dichiarata come **leaf**, quindi non possono derivare altre sottoclassi, mentre da Ellisse e Poligono si

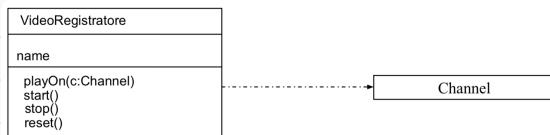


00: 52

# Relazioni

## • Dipendenza

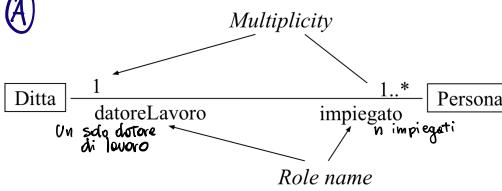
Collega due elementi: basta che un elemento ne nomini un altro per avere una dipendenza:



Siccome VideoRegistratore nomina la classe Channel, allora avremo una dipendenza.

## • Associazione

(A)



Multiplicity

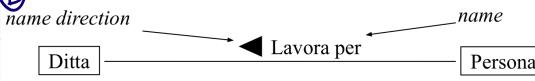
Role name

E' una relazione strutturale tra oggetti chiamati Link con cui gli oggetti possono dialogare

ES: Se ho due classi A e B, nel momento in cui una deve comunicare con l'altra, c'è bisogno di avere visibilità con l'altra classe; I relativi oggetti devono avere visibilità dell'oggetto dell'altra classe.

Quando c'è un'associazione tra due classi, viene inserito un attributo che ci consente di specificare una delle due parti da raggiungere.

(B)



Multiplicity

Role name

Quando non ci sono frecce l'associazione è bidirezionale.

Questo vuol dire che in Ditta devo avere un attributo che mi consente di raggiungere Persona, e viceversa.

Inoltre, l'attributo si trova in entrambi gli oggetti, e ciascun capo deve onorare la molteplicità. Questo significa che quando un ogg Ditta viene formato, avremo al suo interno un Insieme di impiegati (davanti 1..\*).

L'attributo che viene quindi inserito in ditta sarà una collezione di impiegati.

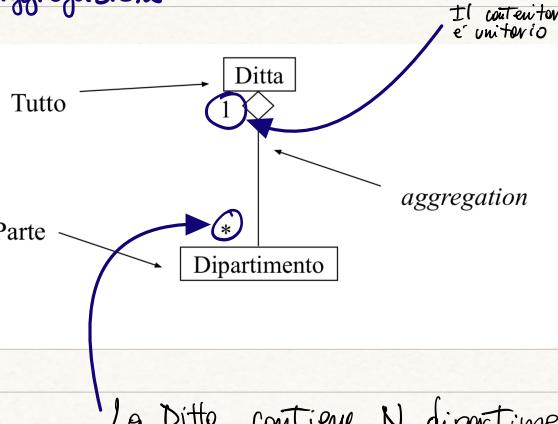
Se ci poniamo in Persona allora avremo un attributo Ditta che sarà però Unico (davanti ad 1).

Partecipa quindi all'associazione 1 istanza di Ditta e più istanze di Persona.

(B) In questo caso le frecce indicano il verso con cui si legge il nome! Quando le frecce non sono presenti si può leggere da entrambi i capi. In questo caso, un capo non vede l'altro: potrei voler modellare il sistema in modo che la ditta vede le persone, ma le singole persone non comunicano con la ditta.

Ditta potrà riferirsi a persone, ma non viceversa.

## • Aggregazione



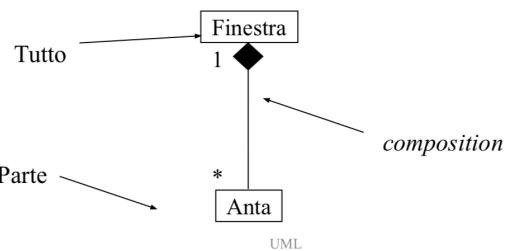
E' un particolare tipo di associazione. Serve per modellare relazioni tutto-parte: da un lato abbiamo un contenitore, e dall'altro abbiamo classi che rappresentano elementi contenuti nel contenitore. Il rombo si mette dal lato del contenitore; Solitamente il contenitore ha molteplicità unitaria.

La Ditta contiene N dipartimenti  $\Rightarrow$  La Ditta aggrega n dipartimenti

Questa relazione è più forte di una associazione ed è legata alla relazione di "contenimento" descritta precedentemente: è una relazione di contenimento debole: la ditta esiste a prescindere dai dipartimenti, così come i dip. esistono a prescindere dalla Ditta.



## • Composizione

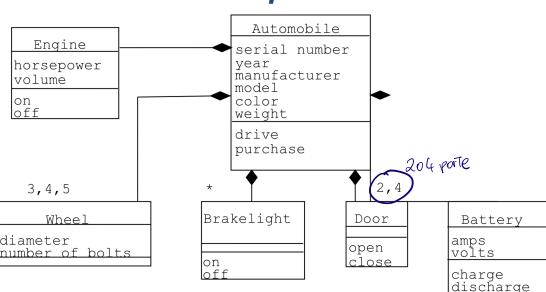


Quando usiamo la composizione possiamo dire che l'elemento, (la finestra) "si compone di" delle sue ante.

Quando uso la composizione, siccome la finestra è fatta dalle sue ante, esiste solo se esistono anche le sue ante. Se distruggo le sue ante, distruggo anche la finestra.

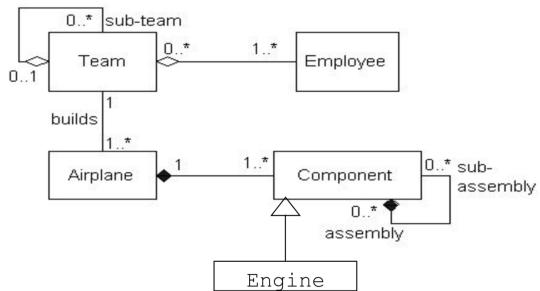
Inoltre, le ante (in questo caso) appartengono solo a quella finestra, non possono quindi essere condivise da altre composizioni.

### Composition



In questo esempio abbiamo un modello di un'auto avendo un certo livello di dettaglio. Tutte queste composizioni fanno sì che questi elementi siano strettamente legati ad un'automobile.

### Esempio Aggregazione e Composizione:



Un team **aggrega** sottoteam; questa è una aggregazione, ed un team **aggrega** gli impiegati. Dal punto di vista della modellizzazione è più usare un'aggregazione più che una composizione.

Il team **costruisce** aereoplani; 1 team viene allocato alla costruzione di un aereo. In questo caso un aereo **si compone** dei suoi componenti. Un componente può a sua volta essere composto da vari sottocomponenti.

### Meta-Object Facility

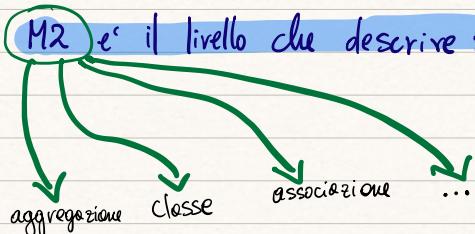
Il linguaggio UML è basato su alcune assunzioni, per cui tutto è un oggetto. La relazione classe-istanza costituisce le fondamenta del linguaggio; quindi ogni concetto UML è un oggetto appartenente ad una classe. UML fa parte di un'architettura standardizzata per la modellazione chiamata **MOF**. Il linguaggio MOF ci consente non solo di creare modelli con UML, ma anche grazie ad altri linguaggi.

Il MOF ha 4 livelli:

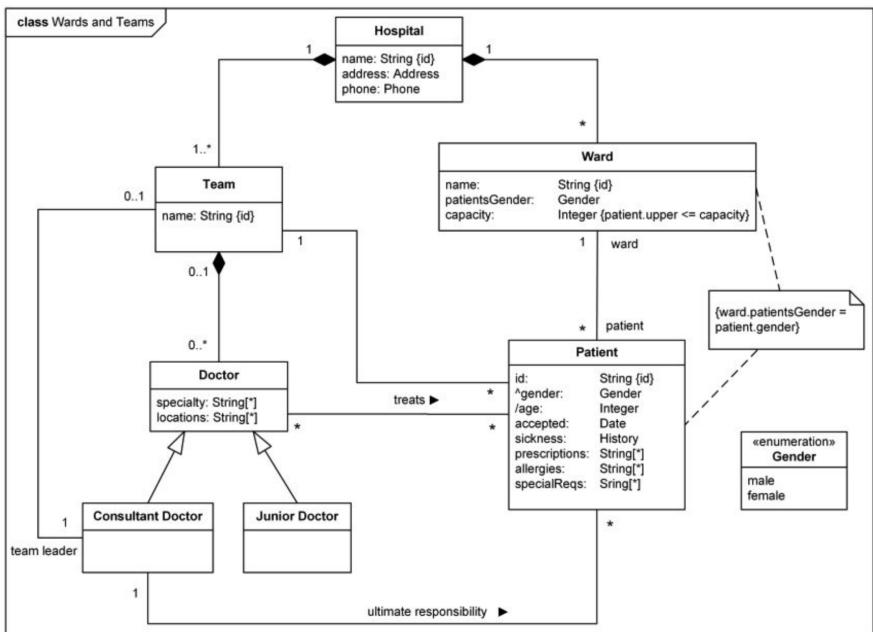
**M0** è il livello della realtà da modellare; troviamo gli oggetti fisici che dobbiamo modellare.

**M1 - M3** ci sono i layer di modellazione che appartengono allo stack della MOF.

**M2** è il livello che descrive modelli. Questo è il livello dove abbiamo scelto di usare UML.



## UML Class Diagram - Un esempio



Esempio di  
Class Diagram