

Introduzione a UML

Francesco Poggi
fpoggi@cs.unibo.it

(dal materiale di Paolo Ciancarini, Angelo Di Iorio e Gian Piero Favini)

A.A. 2016-2017

Modellare

- Un modello è un'**astrazione** che cattura le proprietà salienti della **realtà** che si desidera rappresentare.
- Idealizza una realtà complessa, individuandone i tratti importanti e separandoli dai dettagli, facilitandone la comprensione.
- La mente umana compie un'attività continua di modellazione, producendo schemi per comprendere e spiegare quello che viene percepito dai sensi.
- La realtà è un'*istanza* del modello.

Perché Modellare

- Per *comprendere* il soggetto in analisi.
- Per *conoscere* il soggetto in analisi (fissando ciò che si è compreso).
- Per *comunicare* la conoscenza del soggetto.

I progetti software sono complessi

- Il tipico progetto software raramente coinvolge un solo sviluppatore, e può coinvolgerne anche centinaia:
 - ▶ separare compiti e responsabilità
 - ▶ raggruppare le informazioni a diversi livelli di granularità
- Il tipico progetto software subisce un ricambio di personale nel corso della sua storia:
 - ▶ il progetto perde conoscenza
 - ▶ nuovi sviluppatori devono acquisirla
- Le caratteristiche del progetto spesso mutano col tempo:
 - ▶ necessità di comunicare con il cliente in termini chiari
 - ▶ prevedere ed adattarsi ai cambiamenti
 - ▶ stimarne l'impatto su costi, tempi e risorse di sviluppo

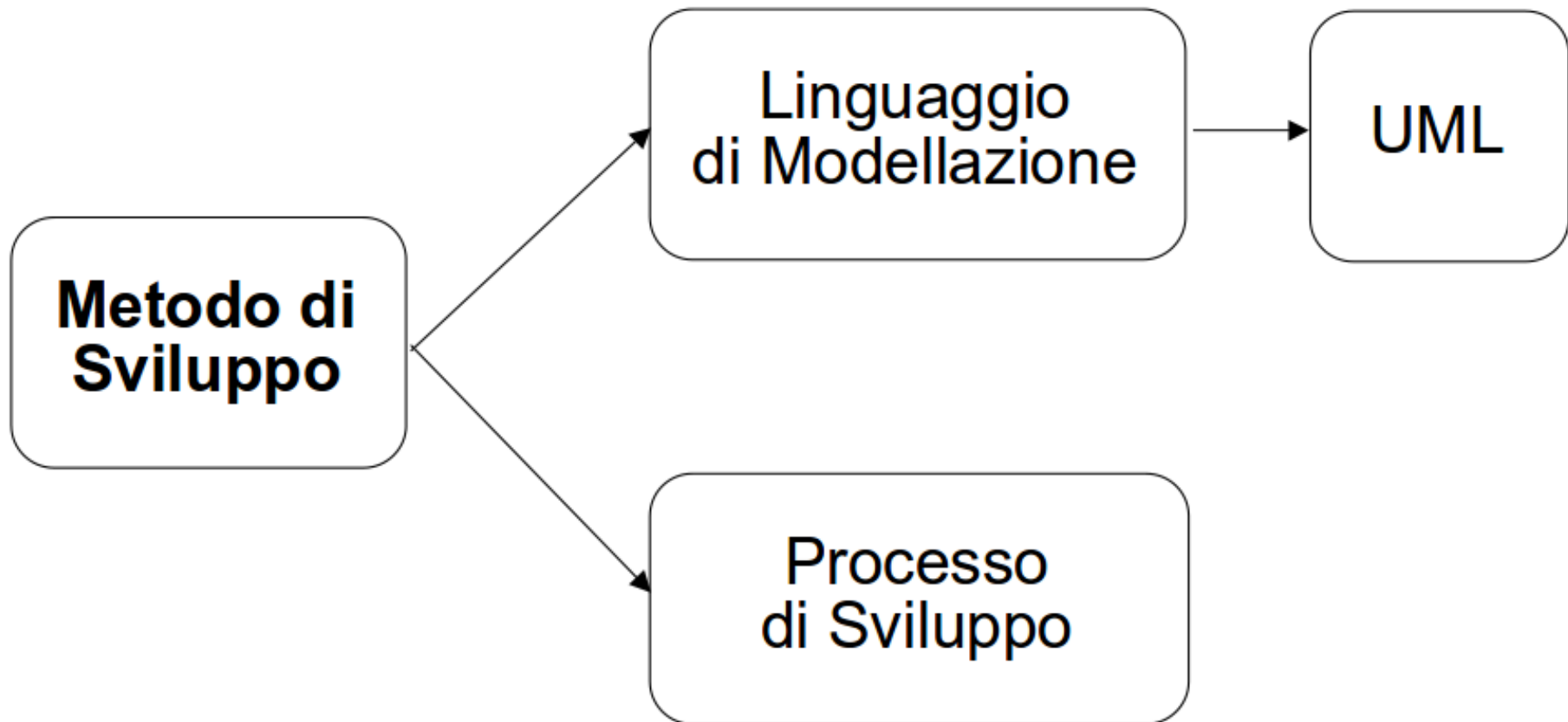
Come si può ragionare su questo se non si sa su *cosa* si sta ragionando?

I linguaggi di modellazione

- Un linguaggio di modellazione fornisce le primitive a cui ricondurre la realtà in esame.
- Permette di esprimere le **entità** che compongono un sistema complesso, le loro **caratteristiche** e le **relazioni** che le collegano.
- Nell'ambito di un progetto, il **linguaggio** di modellazione è normalmente distinto dal **processo** di sviluppo.

Metodo di sviluppo

- Metodo = Linguaggio + Processo
- **Il Linguaggio di Modellazione** è la notazione per esprimere le caratteristiche del progetto
- **Il Processo** consiste in una serie di passi da intraprendere per produrre il progetto



Che cos'è UML (1)

- **UML** (Unified Modeling Language): è un linguaggio **semiformale e grafico** (basato su diagrammi) per:

- ▶ specificare
- ▶ visualizzare
- ▶ costruire
- ▶ documentare

gli artefatti di un sistema software.

- **Artefatti:** sorgenti, eseguibili, documentazione, risultati di test, ecc.

Cos'è UML (2)

- Permette di:
 - ▶ modellare un dominio
 - ▶ scrivere i requisiti di un sistema software
 - ▶ descrivere l'architettura del sistema
 - ▶ descrivere struttura e comportamento di un sistema
 - ▶ documentare un'applicazione
 - ▶ generare automaticamente un'implementazione
- Gli stessi modelli UML sono quindi artefatti usati per sviluppare il sistema e comunicare con il cliente (ma anche con progettisti/sviluppatori/etc.)

Che cos'è UML (3)

- Si tratta di un linguaggio di modellazione usato per capire e descrivere le caratteristiche di un nuovo sistema o di uno esistente.
- Indipendente dall'ambito del progetto.
- Indipendente dal processo di sviluppo.
- Indipendente dal linguaggio di programmazione (progettato per essere abbinato alla maggior parte dei linguaggi object-oriented).
- Fa parte di un metodo di sviluppo, non è esso stesso il metodo.

L come Linguaggio

- Si tratta di un vero e proprio linguaggio, non di una semplice notazione grafica.
- Un modello UML è costituito da un insieme di elementi che hanno *anche* una rappresentazione grafica.
- Il linguaggio è semiformale perché descritto in linguaggio naturale e con l'uso di diagrammi, cercando di ridurre al minimo le ambiguità.
- Ha regole *sintattiche* (come produrre modelli legali) e regole *semantiche* (come produrre modelli con un significato).

Sintassi e semantica: esempio



- Usiamo un semplice diagramma dei Casi d'Uso come esempio.
- Regola sintattica: una relazione tra un attore e un caso d'uso può opzionalmente includere una freccia.
- Regola semantica: la freccia significa che la prima interazione si svolge nel senso indicato dalla freccia.

U come Unificato

- UML rappresenta la sintesi di vari approcci metodologici fusi in un'unica entità.
- L'intento era di prendere il meglio da ciascuno dei diversi linguaggi esistenti e integrarli. Per questo motivo, UML è un linguaggio molto vasto.
- Questa è una delle critiche principali mosse a UML ("vuole fare troppe cose").

Breve storia (1)

- Agli inizi degli anni '90 vi era una proliferazione di linguaggi e approcci alla modellazione object-oriented. Mancava uno standard accettato universalmente per la creazione di modelli software.
- C'era la sensazione che la quantità di differenti soluzioni ostacolasse la diffusione dello stesso metodo object-oriented.
- Nel 1994 due esperti di modellazione della *Rational Software Corporation*, Grady Booch e James Rumbaugh, unificarono i propri metodi, **Booch** e **OMT** (Object Management Technique).
- Nel 1995 si unisce anche Ivar Jacobson con il suo **OOSE** (Object Oriented Software Engineering), dopo l'acquisizione della sua compagnia Objectory da parte di Rational.

The 'three amigos'

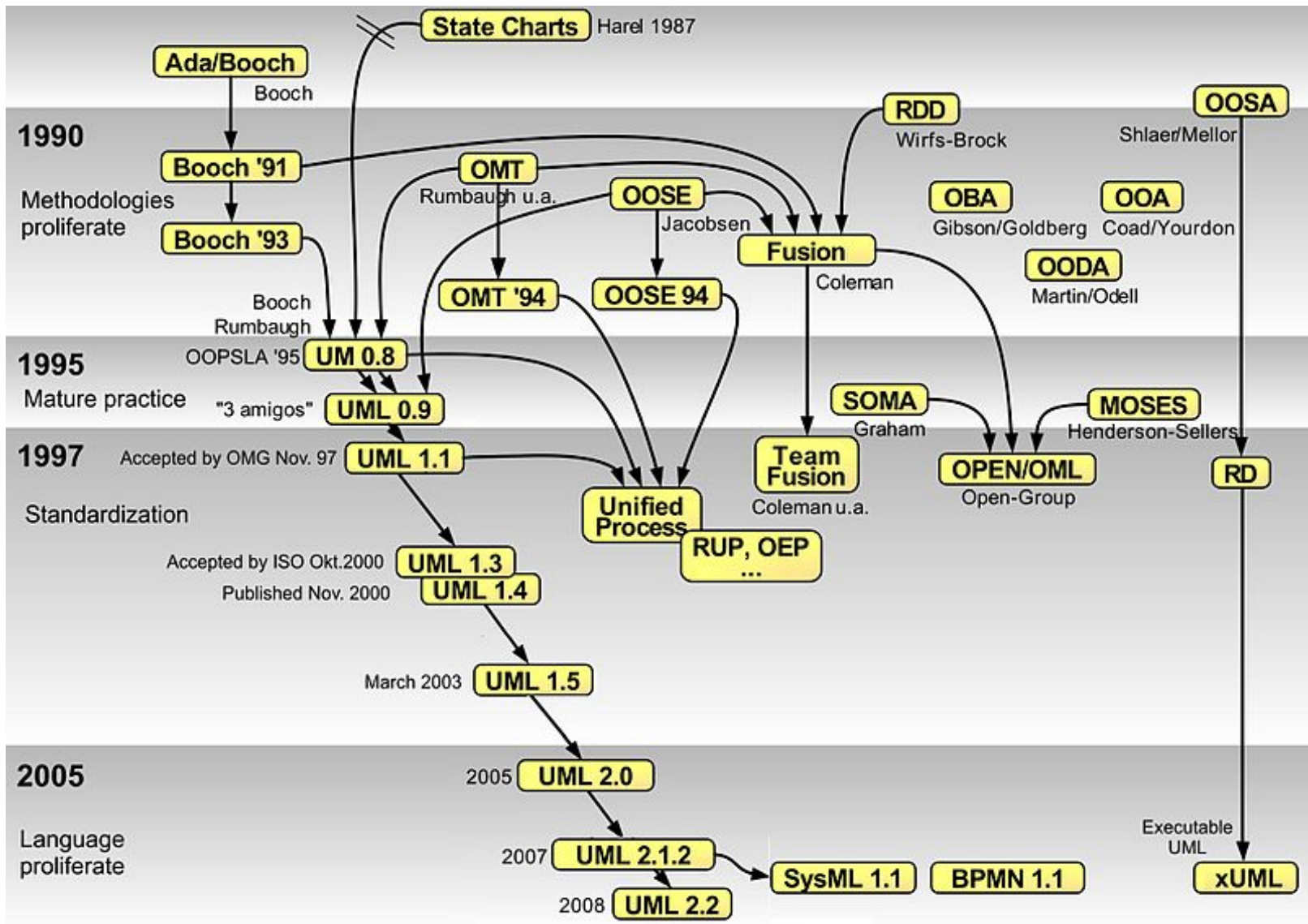
- **Booch:** definizione e classificazione di nozioni base
- **Rumbaugh:** modelli e notazioni diagrammatiche
- **Jacobson:** modello di processo Objectory



Grady Booch, Jim Rumbaugh, Ivar Jacobson

Breve storia (2)

- Nel 1996, Booch, Rumbaugh e Jacobson, sono incaricati da Rational di dirigere la creazione dell'Unified Modeling Language e RUP.
- (Nel 2001 Rational verrà acquisita da IBM)
- Nel 1997, la specifica UML 1.0 viene presentata all'**OMG** (Object Management Group), un consorzio di grandi aziende interessate allo sviluppo di standard e tecnologie basate su oggetti.
- Nel novembre dello stesso anno, una versione arricchita, UML 1.1, viene approvata dall'OMG.
- Seguono aggiornamenti: 1.2 (1998), 1.3 (1999), 1.4 (2001), 1.5 (2003), e la major revision 2.0 (2005).
- L'ultima versione è la 2.5 rilasciata a dicembre 2013.



Object-oriented

- UML non è solo un linguaggio per la modellazione, ma un linguaggio per la modellazione **orientata agli oggetti**.
- Questo include sia l'*analisi* che la *progettazione* orientata agli oggetti (OOA e OOD, rispettivamente):
 - ▶ **analisi**: capire **cosa** deve fare il sistema, senza occuparsi dei dettagli implementativi
 - ▶ **progettazione**: capire **come** il sistema raggiunge il suo scopo, come viene implementato
- UML offre strumenti di modellazione OO in entrambi gli ambiti; frammenti differenti di UML sono impiegati in diverse fasi del processo di sviluppo (anche se UML stesso non fornisce indicazioni sul suo utilizzo).

Object-oriented

- OO: un paradigma che sposta l'enfasi della programmazione dal codice verso le **entità** su cui esso opera, gli **oggetti**.
- Una rivoluzione copernicana cominciata dagli anni '60 e proseguita, lentamente, fino ad affermarsi negli anni '90.
- I principi cardine furono proposti separatamente e solo successivamente integrati in unico paradigma.
- Principi OO comunemente accettati sono: **Abstraction, Encapsulation, Inheritance, Polymorphism**.

Principi OO

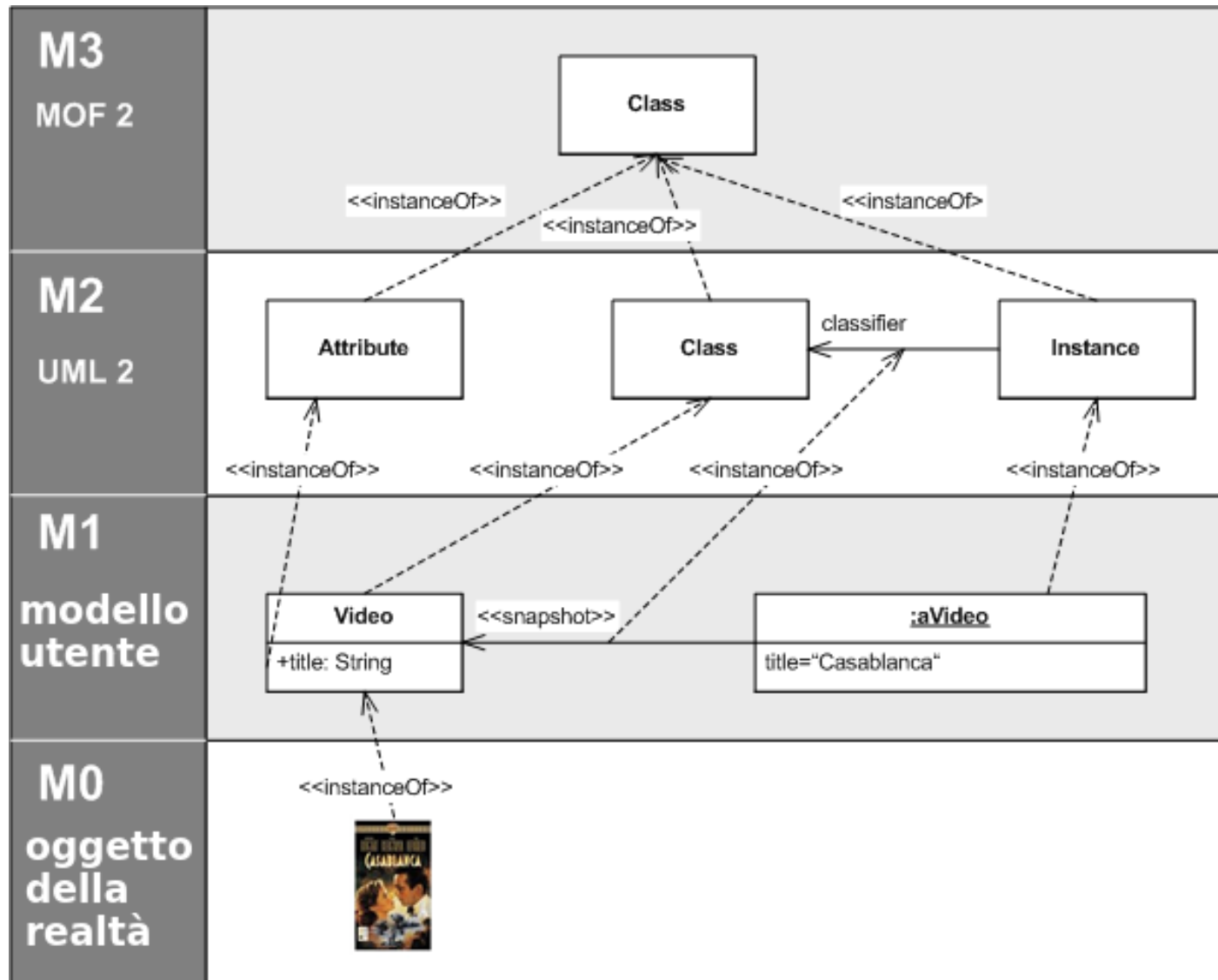
- **Abstraction** : usare *classi* per astrarre la natura e le caratteristiche di un oggetto, che è un'istanza della propria classe di appartenenza.
- **Encapsulation** : nascondere al mondo esterno i dettagli del funzionamento di un oggetto; gli oggetti hanno accesso solo ai dati di cui hanno bisogno.
- **Inheritance** : classi possono specializzare altre classi ereditando da esse e implementando solo la porzione di comportamento che differisce.
- **Polymorphism** : invocare comportamento diverso in reazione allo stesso messaggio, a seconda di quale oggetto lo riceve.

Meta-Object Facility

- Si può dire che in UML tutto è un oggetto.
- La relazione classe/istanza costituisce le fondamenta stessa del linguaggio.
- UML stesso, il linguaggio, è un'*istanza*!
- UML fa parte di un'architettura standardizzata per la modellazione di OMG chiamata **MOF** (Meta-Object Facility).
- Si può vedere MOF come un linguaggio per creare linguaggi, uno dei quali è UML.

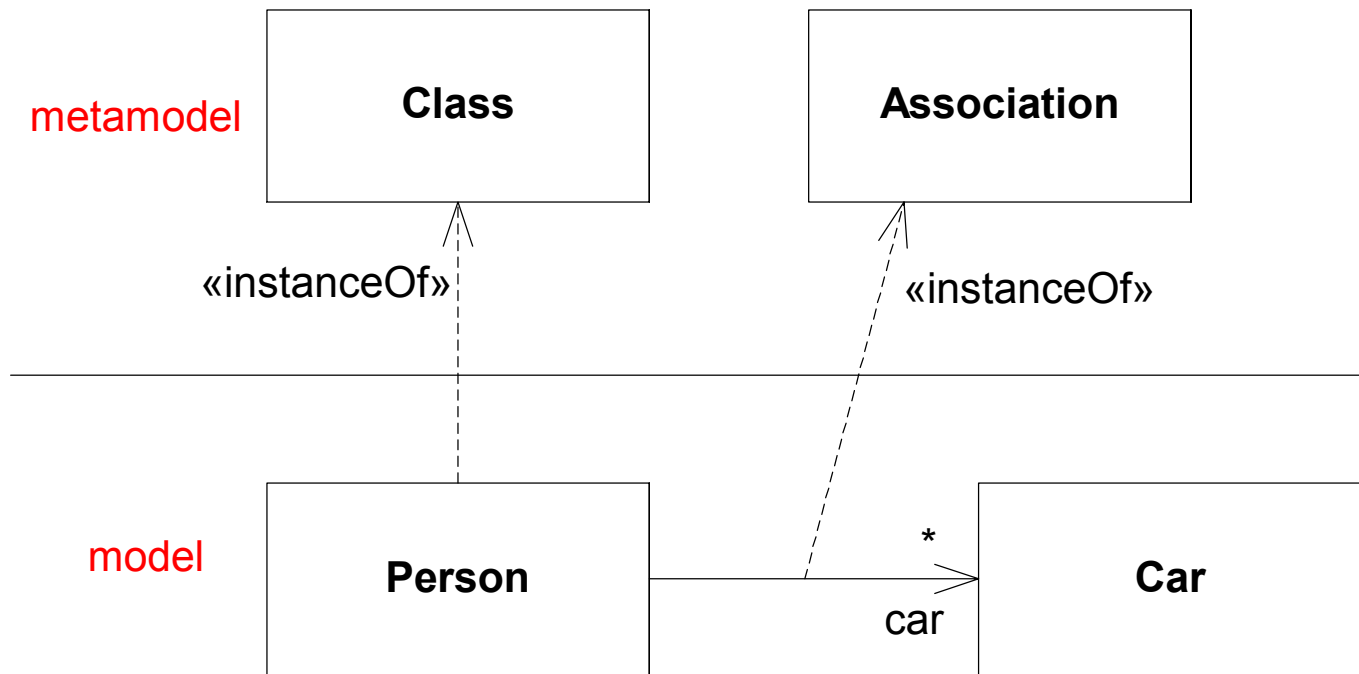
Il metamodello UML

- MOF ha 4 livelli: M0, M1, M2 e M3. Ogni livello è un'istanza di un elemento del livello superiore.
 - ▶ un elemento di M0 è la realtà da modellare
 - ▶ un elemento di M1 è un modello che descrive la realtà
 - ▶ un elemento di M2 è un modello che descrive modelli (*metamodello*); **UML è qui**
 - ▶ un elemento di M3 è un modello che descrive metamodelli (*meta-metamodello*); **MOF è qui**
- OMG usa MOF per definire altri linguaggi oltre a UML.
- Tutti i linguaggi basati su MOF e i modelli con essi prodotti possono essere serializzati e scambiati tramite lo standard **XMI** (XML Metadata Interchange).

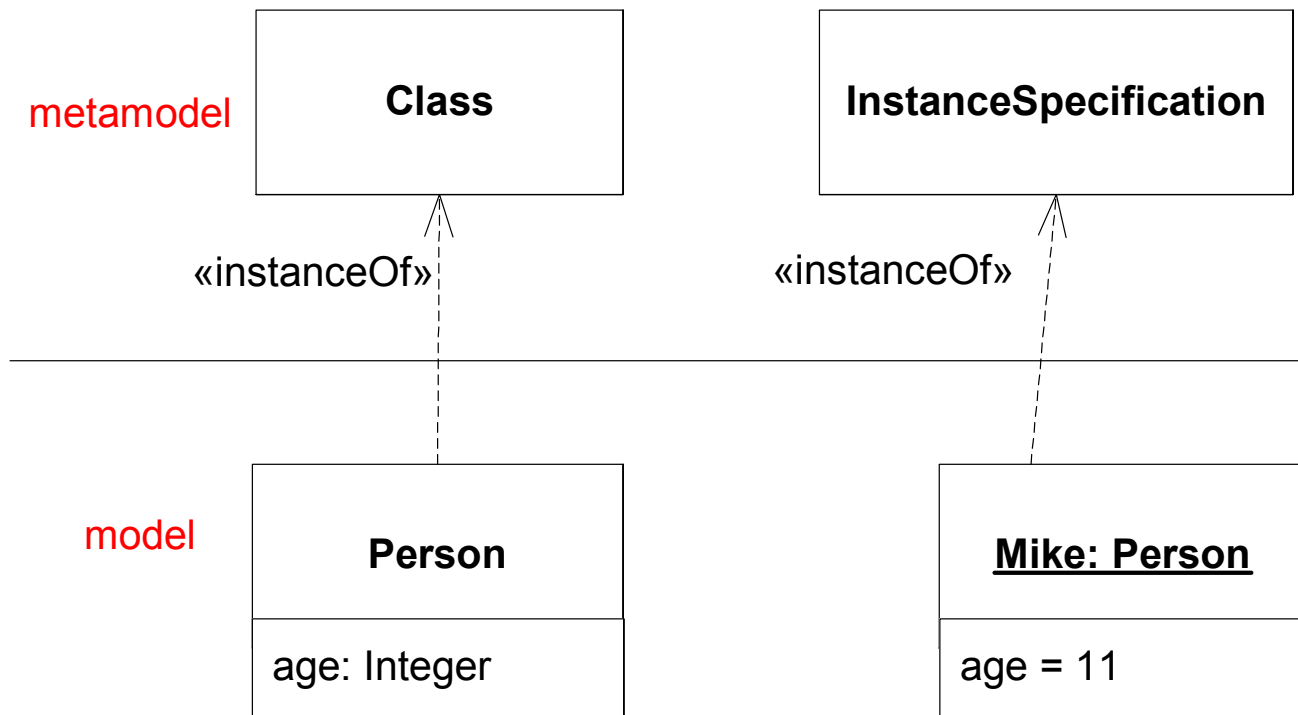


Semplificando...

- Si può dire che UML è definito tramite un modello UML (o piuttosto, usando un modello M3 che usa primitive comuni a UML).
- In qualunque momento, un oggetto al livello Mx è un'istanza di uno del livello superiore.
- Il meta-metamodello di livello M3 è progettato per essere istanza di se stesso, quindi non esiste M4.
- **Chi usa UML crea modelli di livello M1**; tuttavia, è bene sapere che esistono anche i livelli superiori.



M1 e M2 a confronto (usando un diagramma UML).



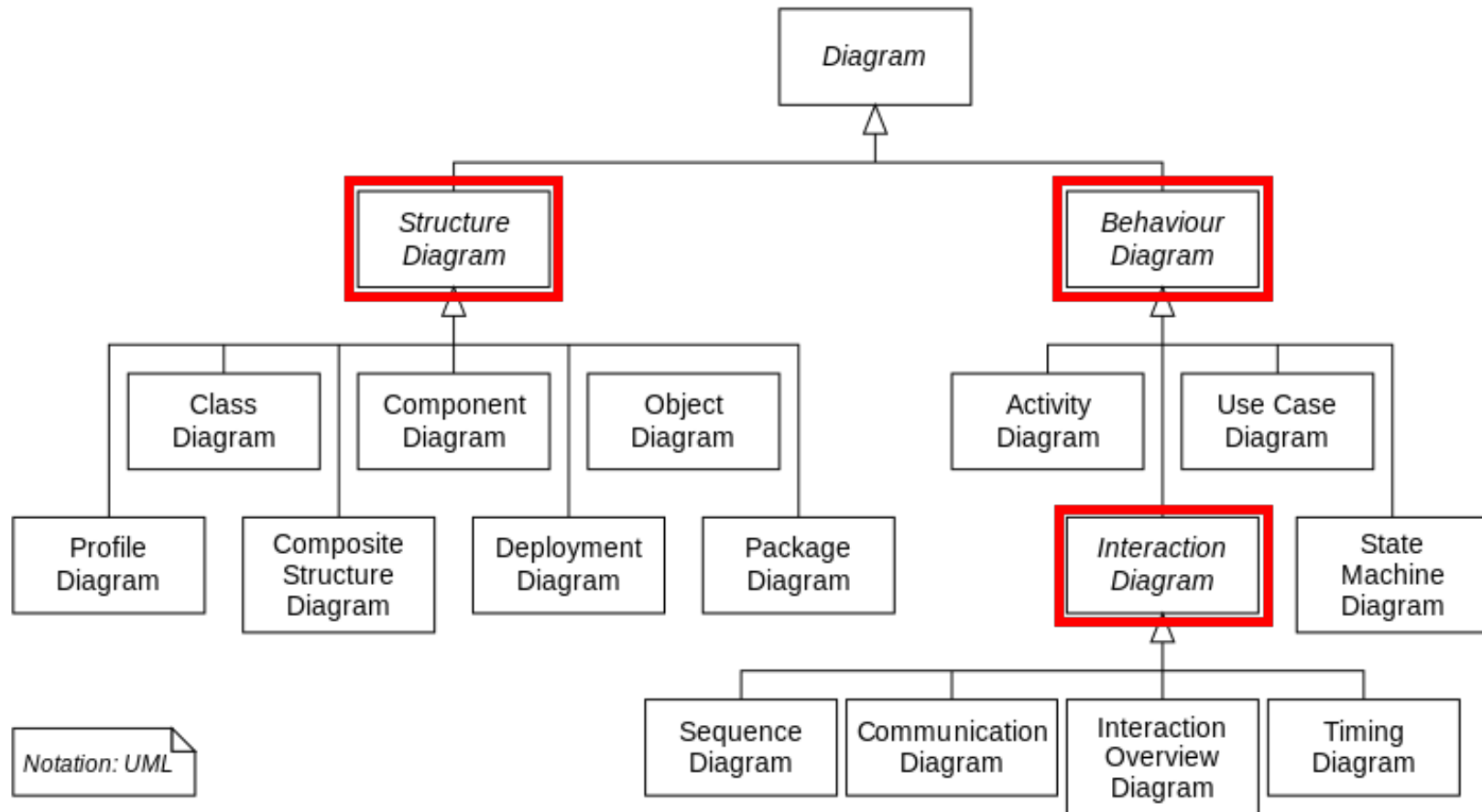
Altro esempio con M1 (modello utente) e M2 (metamodello UML)

I diagrammi e le viste

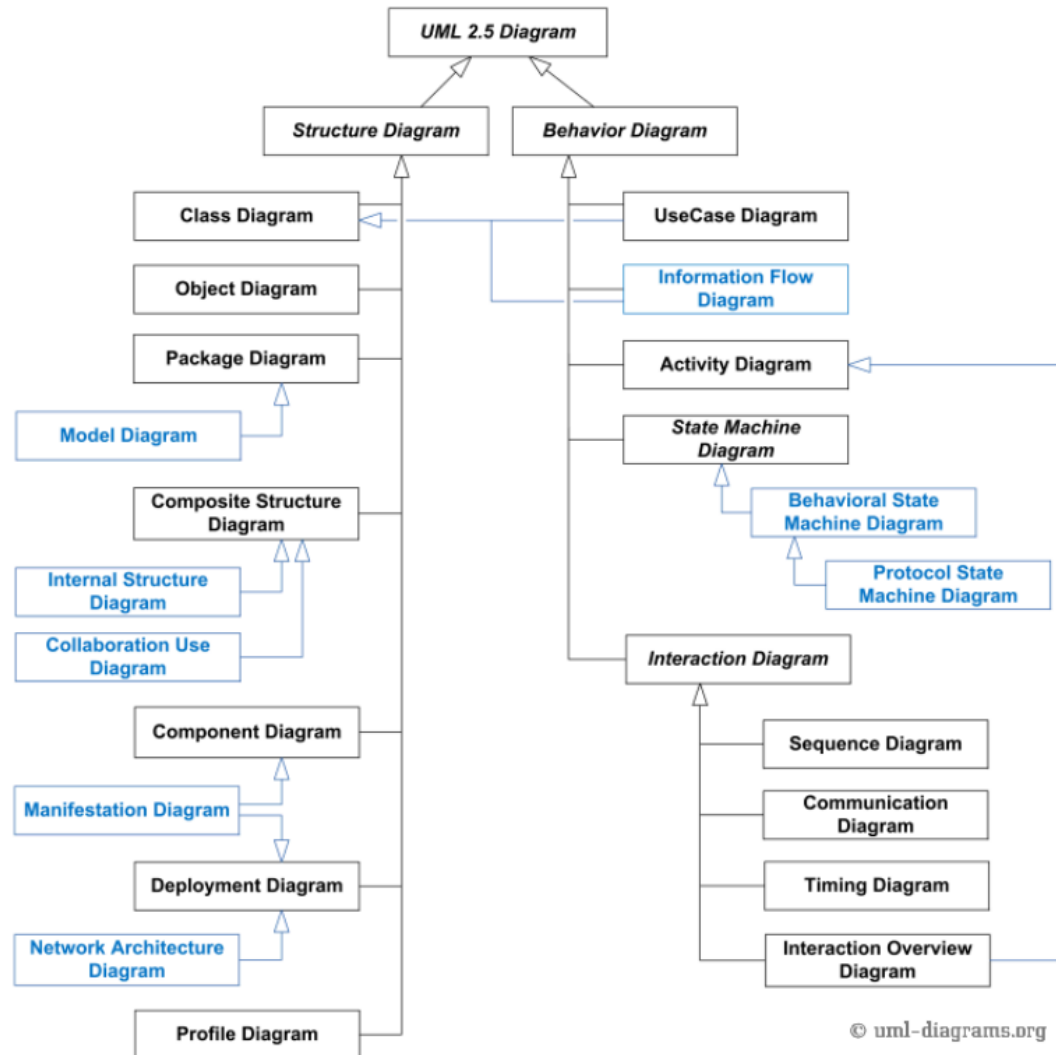
- Un **diagramma** è la **rappresentazione grafica di una parte del modello**.
- Fornisce una **vista** di un sistema o una sua parte, cioè ne mette in risalto diverse proprietà.
- 4+1 viste (Kruchten, 1995):
 - ▶ **Logical**: mette in risalto la scomposizione logica del sistema tramite classi, oggetti e loro relazioni
 - ▶ **Development**: mostra l'organizzazione del sistema in blocchi strutturali (package, sottosistemi, librerie, ...)
 - ▶ **Process**: mostra i processi (o thread) del sistema in funzione, e le loro interazioni
 - ▶ **Physical**: mostra come il sistema viene installato ed eseguito fisicamente
 - ▶ **Use case**: (+1) la vista che agisce da 'collante' per le altre; spiega il funzionamento desiderato del sistema

- UML 2.5 definisce 14 diagrammi (contro i 9 di UML 1.x), divisi in categorie:
 - ▶ **Diagrammi di Stato:** mostrano la *struttura statica* del sistema e delle sue parti a diversi livelli di astrazione ed implementazione, e delle relazioni che intercorrono fra di essi;
 - ▶ **Diagrammi di Comportamento:** mostrano il *comportamento dinamico* degli oggetti di un sistema, che può essere descritto come una serie di cambiamenti al sistema nel tempo.
 - ★ **Diagrammi di Interazione:** mostrano come un certo comportamento viene realizzato dalla collaborazione delle entità in gioco.

Tassonomia dei 14 diagrammi di UML 2



Gerarchia
dei
diagrammi
(Work in
progress 2.5)



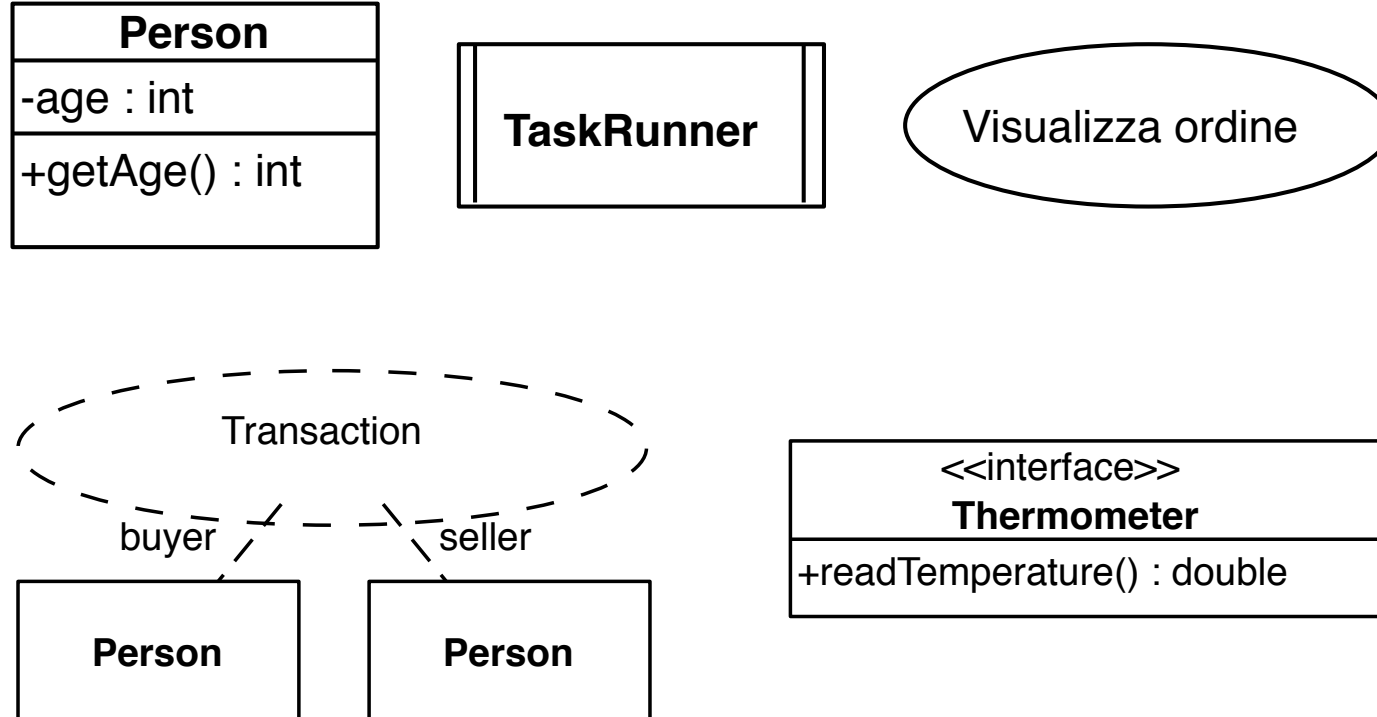
- **NB:** gli elementi in blu non sono parte della tassonomia ufficiale di UML 2.5.

Entità UML

- UML prevede diversi tipi di entità che possono essere organizzati in quattro categorie:
 - ▶ Strutturali
 - ▶ Comportamentali
 - ▶ Informative
 - ▶ Raggruppamento e contenimento
- Una lista delle entità usate da ogni singolo diagramma è disponibile all'url:
`www.uml-diagrams.org/uml-25-diagrams.html`
- Segue una panoramica delle entità principali ma andremo in dettaglio quando analizzeremo i diversi diagrammi

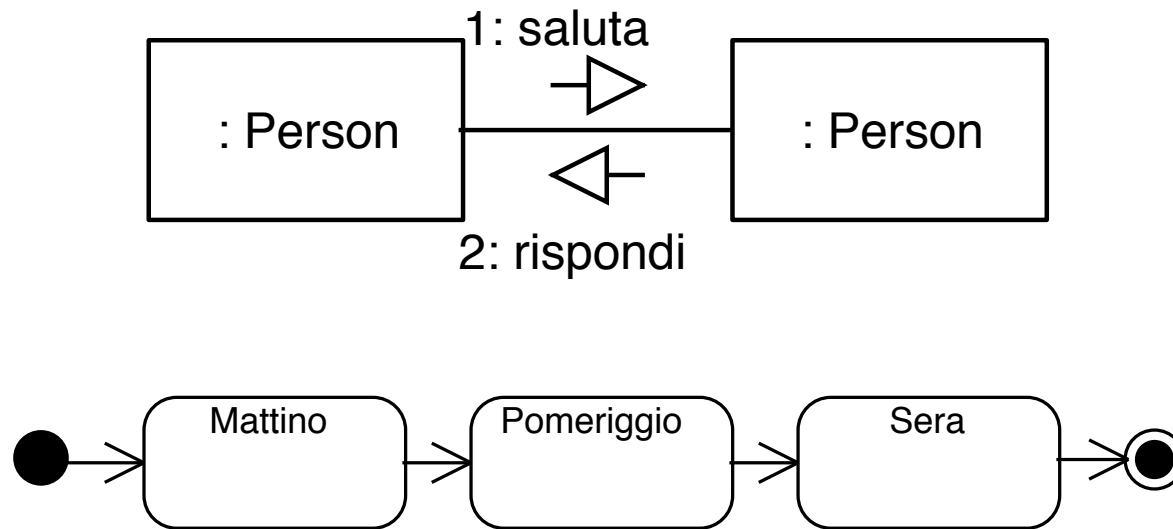
UML: entità strutturali

- Definiscono le "cose" (*classificatori, things*) del modello
- Alcuni esempi: classi, classi attive, use-case, collaborazioni, interfacce



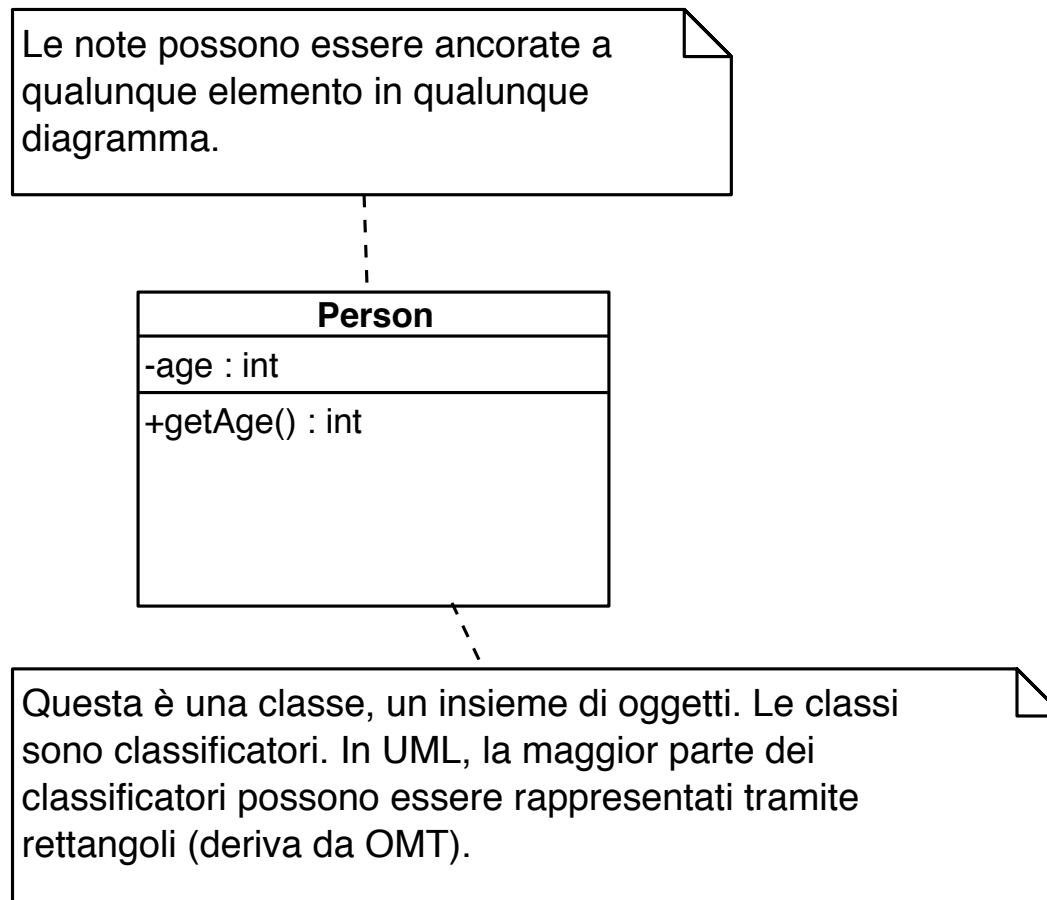
UML: entità comportamentali

- Descrivono il "behavior": interazioni, collaborazioni (communication), scambi di messaggi, transizioni di stato, etc.



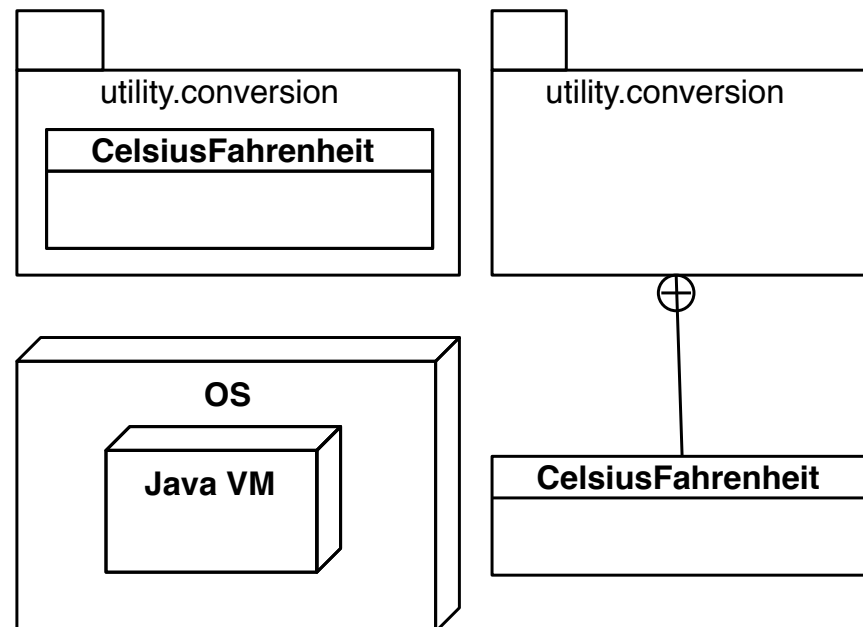
Entità informative

- Uno degli scopi principali della modellazione è la leggibilità: un diagramma non leggibile e informativo serve a poco...
- Le *note* UML non hanno effetti sul modello ma migliorano la leggibilità.



UML: entità di raggruppamento e contenimento

- I *package* raggruppano altri elementi e forniscono loro un *namespace* (che permette poi di identificare ogni elemento con il suo nome).
- In UML, moltissimi elementi possono contenere altri elementi al loro interno, formando una struttura gerarchica, rappresentabile graficamente in vari modi.

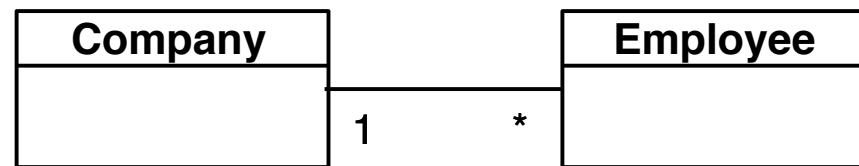


Relazioni

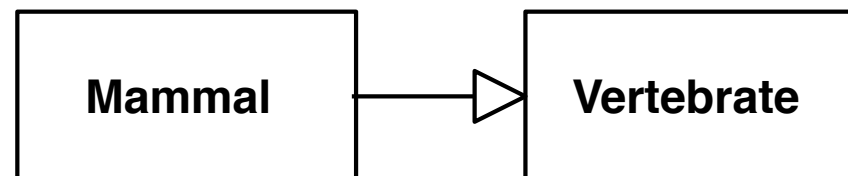
- Gli elementi del modello possono essere collegati da **relazioni**.
- Rappresentate graficamente tramite linee.
- Possono avere un nome.
- Quattro sottotipi fondamentali:
 - ▶ **Association**
 - ▶ **Generalization**
 - ▶ **Dependency**
 - ▶ **Realization**

Association e generalization

- Un'associazione descrive l'esistenza di un nesso tra le istanze di classificatori (*things*) e ha varie caratteristiche, alcune opzionali.

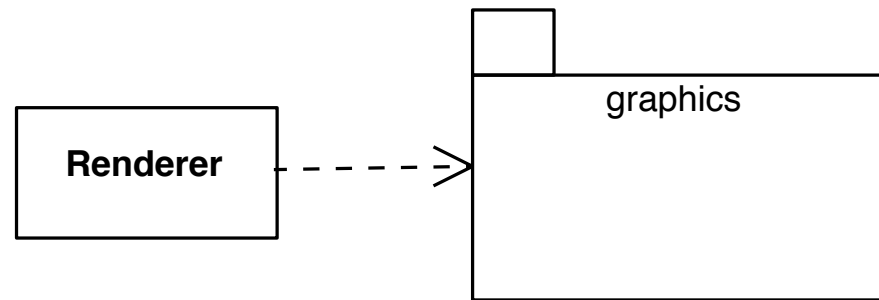


- La generalizzazione è una relazione tassonomica da un elemento specializzato verso un altro, più generale, dello stesso tipo.
 - ▶ Il figlio è sostituibile al genitore dovunque appaia, e ne condivide struttura e comportamento.

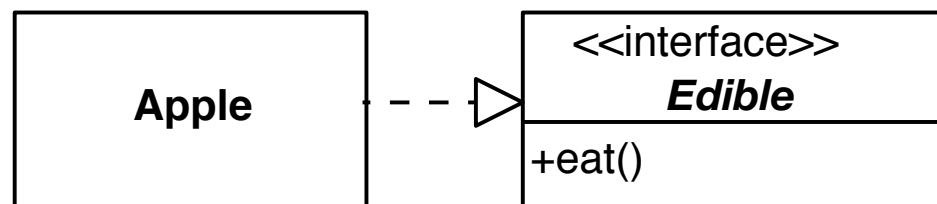


Dipendenze e realizzazioni

- Una dipendenza è una relazione semantica: indica che il *client* dipende, semanticamente o strutturalmente, dal *supplier* (variazioni alla specifica del supplier possono cambiare quella del client).



- Anche la realizzazione è una relazione semantica: il supplier fornisce una specifica, il client la realizza (es: implementazione di interfacce, templates, etc.)



Le frecce in UML

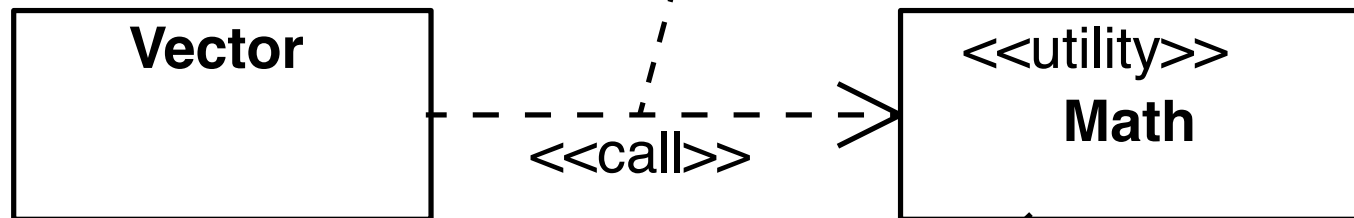
- Un metodo mnemonico per ricordarsi il verso di tutte le frecce in UML è il seguente:
- In UML, tutte le frecce vanno **da chi sa verso chi non sa** (dell'esistenza dell'altro).
- In una generalizzazione, il figlio sa di estendere il genitore, ma il genitore non sa di essere esteso.
- In una dipendenza, chi dipende sa da chi dipende, ma non vice-versa.
- In una realizzazione, chi implementa conosce la specifica, ma non il contrario.

Stereotipi

- Un'altra primitiva di UML comune ad ogni diagramma.
- Rende un diagramma più informativo arricchendo la semantica dei costrutti UML.
- Uno stereotipo è una parola chiave tra virgolette e abbinata ad un elemento del modello.
- Es. «import», «utility», «interface».

Stereotipi in un diagramma delle classi

Questa dipendenza tra Vector e Math ha lo stereotipo «call»; significa che operazioni di Vector invocano operazioni di Math.



Lo stereotipo «utility» indica che questa è una classe utility, cioè una collezione di variabili globali e operazioni statiche usate da altre parti del sistema.

Stereotipi come estensioni di UML

- Gli stereotipi forniscono significato aggiuntivo ai costrutti UML.
- Possono essere usati per adattare UML a particolari ambiti e piattaforme di sviluppo.
- Stereotipi, vincoli e regole aggiuntivi vengono raccolti in *profili*, che costituiscono uno dei principali meccanismi di estensione di UML.

OCL (1)

- **Object Constraint Language** (OCL) è un linguaggio, approvato e standardizzato da OMG, per la specifica di vincoli. Si usa assieme ad UML e a tutti i linguaggi dell'architettura MOF.
- Non è obbligatorio, ma aggiunge rigore formale al modello.
- Specifica condizioni che devono essere soddisfatte dalle istanze di una classe: i vincoli sono espressioni booleane considerate *true*.
- Un tool UML può usare OCL
 - ▶ per validare il modello
 - ▶ per generare codice

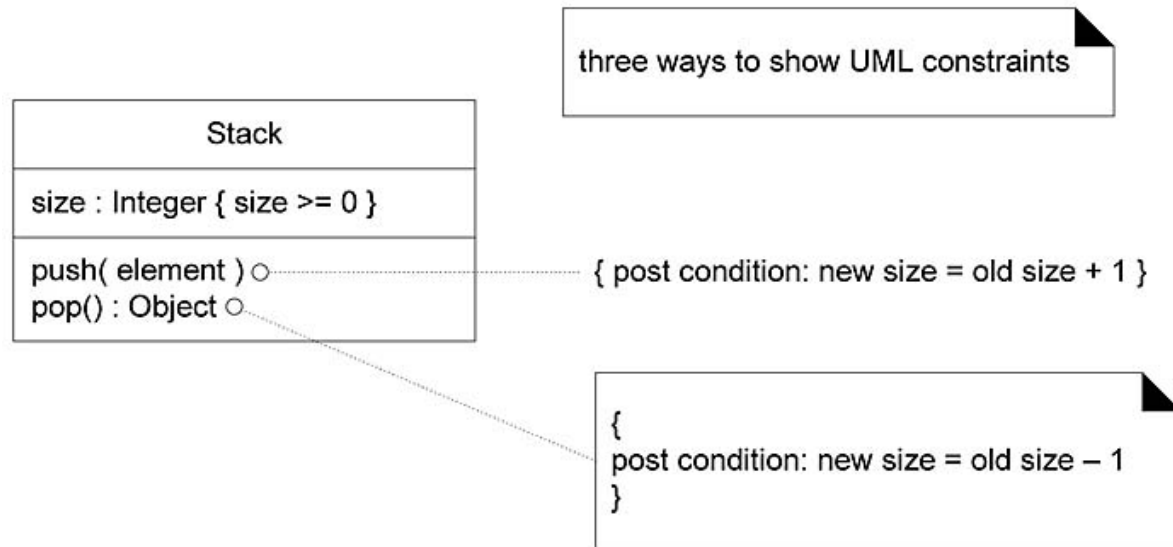
OCL (2)

- Un **vincolo** OCL opera in un determinato **contesto**, specificando proprietà soddisfatte da tutte le istanze di quel contesto.
- Il contesto può essere una classe, un suo attributo o una sua operazione.
- I vincoli hanno un tipo che descrive l'ambito della loro **validità**.
- **context** Car **inv**:
fuel>=0
- Questo vincolo si applica alla classe Car ed è un invariante (sempre valido).

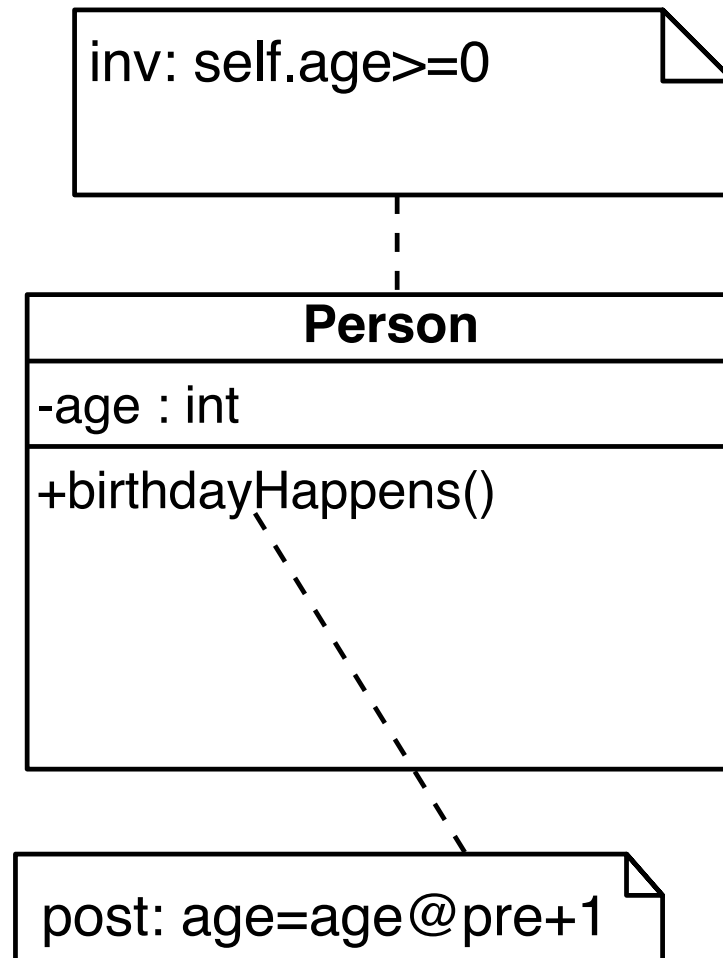
Tipi di vincoli in OCL

- **inv:** Invariante, sempre valido nel contesto.
- **pre:** Nel contesto di un'operazione, una preconditione per la sua esecuzione.
- **post:** Nel contesto di un'operazione, una postcondizione vera dopo l'esecuzione.
- **body:** Definisce una query nel contesto.
- **init:** Definisce il valore iniziale nel contesto.
- **derive:** Definisce un attributo derivato del contesto.

OCL nei diagrammi (1)



OCL nei diagrammi (2)



(In una postcondizione, '@pre' permette di accedere al valore precedente di un attributo.)

Conclusioni

- Modellare è indispensabile in qualunque progetto non banale.
- UML è un linguaggio general-purpose per la modellazione.
- Non è perfetto, ma è potente e costituisce uno standard diffuso ed accettato.
- Costruire un buon modello è difficile.

Alcuni tool

- *ArgoUML*: <http://argouml.tigris.org/>
- *Papyrus*: <http://eclipse.org/papyrus>
- *Umbrello*: <http://uml.sourceforge.net/>
- *Eclipse*: www.eclipse.org/uml2/
- ...