Indice

7 – Modellazione

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, facilitando 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 può essere vista come un'**istanza** del modello.

Esistono **tre motivi** per cui è necessaria la modellazione:

1. Per **comprendere** il soggetto in analisi.
2. Per **conoscere** il soggetto in analisi, fissando ciò che si è compreso.
3. Per **comunicare** la conoscenza del soggetto.

Si **approfondisce** adesso il **motivo** dell'**uso della modellazione**.

Il tipico progetto software raramente coinvolge un solo sviluppatore, solitamente sono vari team che formano in tutto centinaia di sviluppatori. Da un numero così elevato nasce il bisogno di separare compiti, responsabilità e raggruppare le informazioni a diversi livelli di granularità.

Inoltre, solitamente il progetto software subisce un ricambio di personale nel corso della sua storia. Questo provoca che il nuovo personale dovrà spendere del tempo per conoscere i dettagli ed entrare nella giusta mentalità.

Nel tempo, durante lo sviluppo, possono nascere sicuramente nuove idee che provocano il cambiamento delle caratteristiche del progetto. Le nuove peculiarità devono essere comunicate al cliente in modo non ambiguo (si è già affrontato questo tema nei primi capitoli dello sviluppo). L'azienda deve essere capace di prevedere ed adattarsi a questi cambiamenti, stimando inoltre l'impatto su costi, tempi e risorse di sviluppo.

Per effettuare "brainstorming" su questi temi, è necessario un modello chiaro e semplice.

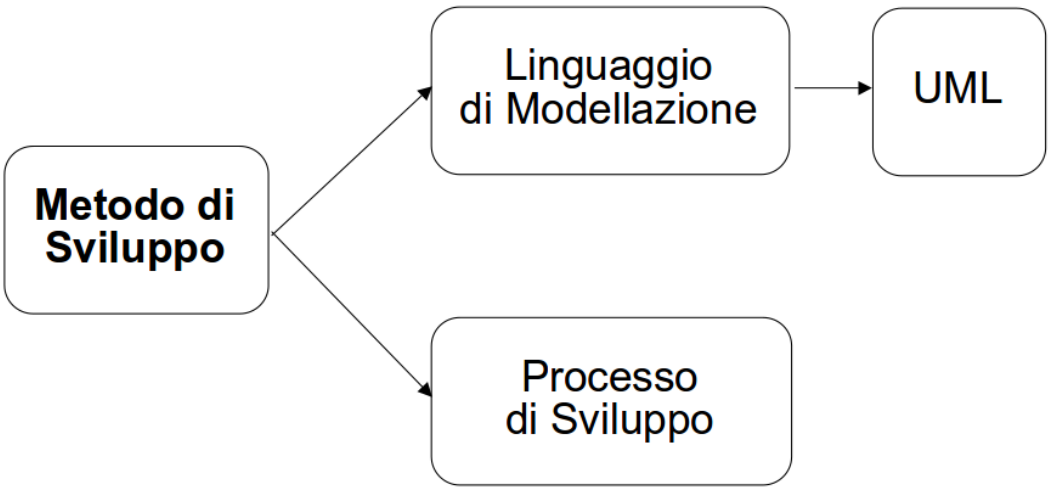
Ecco i motivi della nascita dei **linguaggi di modellazione**. Un linguaggio di modellazione **fornisce le primitive a cui ricondurre la realtà** presa in esame. Consente di esprimere le **entità** che compongono un sistema complesso, le loro **caratteristiche** e le **relazioni** che le collegano (relazioni viste nel capitolo precedente). Nell'ambito di un progetto, il **linguaggio di modellazione** è normalmente **distinto** dal **processo di sviluppo**.

Tuttavia, la **modellazione** prevede anche la parte di sviluppo. Infatti, è possibile scrivere una sorta di equazione:

$$Metodo = Linguaggio + Processo$$

In cui il $Linguaggio$ è il **linguaggio di modellazione** (**notazione per esprimere le caratteristiche del progetto**).

In cui il $Processo$ è il **processo** (**serie di passi da intraprendere per produrre il progetto**).



8 – UML – Approfondimento

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

1. **Specificare**;
2. **Visualizzare**;
3. **Costruire**;
4. **Documentare**.

gli artefatti di un sistema software. Per **artefatti** si intendono: sorgenti, eseguibili, documentazione, risultati di test, ecc.

Questo strumento viene **usato per**:

- 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, ecc.).

In altre parole, questo linguaggio viene **usato per capire** e **descrivere** le caratteristiche di un nuovo sistema o di uno esistente. Indipendente dall'ambito del progetto, dal processo di sviluppo o dal linguaggio di programmazione (anche se progettato per essere abbinato ai linguaggi *object-oriented*). Fa parte di un metodo sviluppo, non è esso stesso il metodo.

La L di UML si riferisce a "linguaggio", poiché viene considerato un vero e proprio **linguaggio**, non 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).

Un **esempio** per capire la differenza tra regole sintattiche e regole semantiche.

Quindi:

- La **regola sintattica** è la relazione tra un attore e un caso d'uso opzionalmente incluso con una freccia;
- La **regola semantica** rappresenta la freccia che significa che la prima interazione si svolge nel senso indicato dalla freccia.

La U di UML si riferisce a "Unificato", poiché essa **rappresenta la sintesi di vari approcci metodologici fusi in un'unica entità**. L'obbiettivo quando venne creato, era di prendere il meglio da ciascuno dei diversi linguaggi esistenti e integrarli. Per questo motivo, UML è un linguaggio molto vasto. Una delle **critiche** mosse contro questo linguaggio fu "vuole fare troppe cose" (invidiosi).

8.1 – MOF e XML

UML non è solo un linguaggio per la modellazione, ma un linguaggio per la modellazione **orientata agli oggetti**. Quindi include l'**analisi** e 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.

Vengono offerti da UML degli strumenti di modellazione orientati agli oggetti in entrambi gli ambiti. Inoltre, diversi frammenti di UML sono impiegati in diverse fasi del processo di sviluppo.

Il linguaggio ad oggetti, ovviamente sposta l'enfasi della programmazione del codice verso l'**entità** su cui esso opera, ovvero gli **oggetti**.

I **principi** del linguaggio orientato agli oggetti sono:

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

Affermare che in UML **tutto è un oggetto**, non è errato. Infatti, la relazione classe e istanza costituisce le fondamenta stessa del linguaggio. UML stesso è un'istanza! Inoltre, fa parte di un'architettura standardizzata per la modellazione chiamata **MOF** (**Meta-Object Facility**). Quest'ultimo è un linguaggio per creare linguaggi, per esempio UML.

Il MOF ha **4 livelli** in cui ogni istanza appartiene a un elemento del livello superiore:

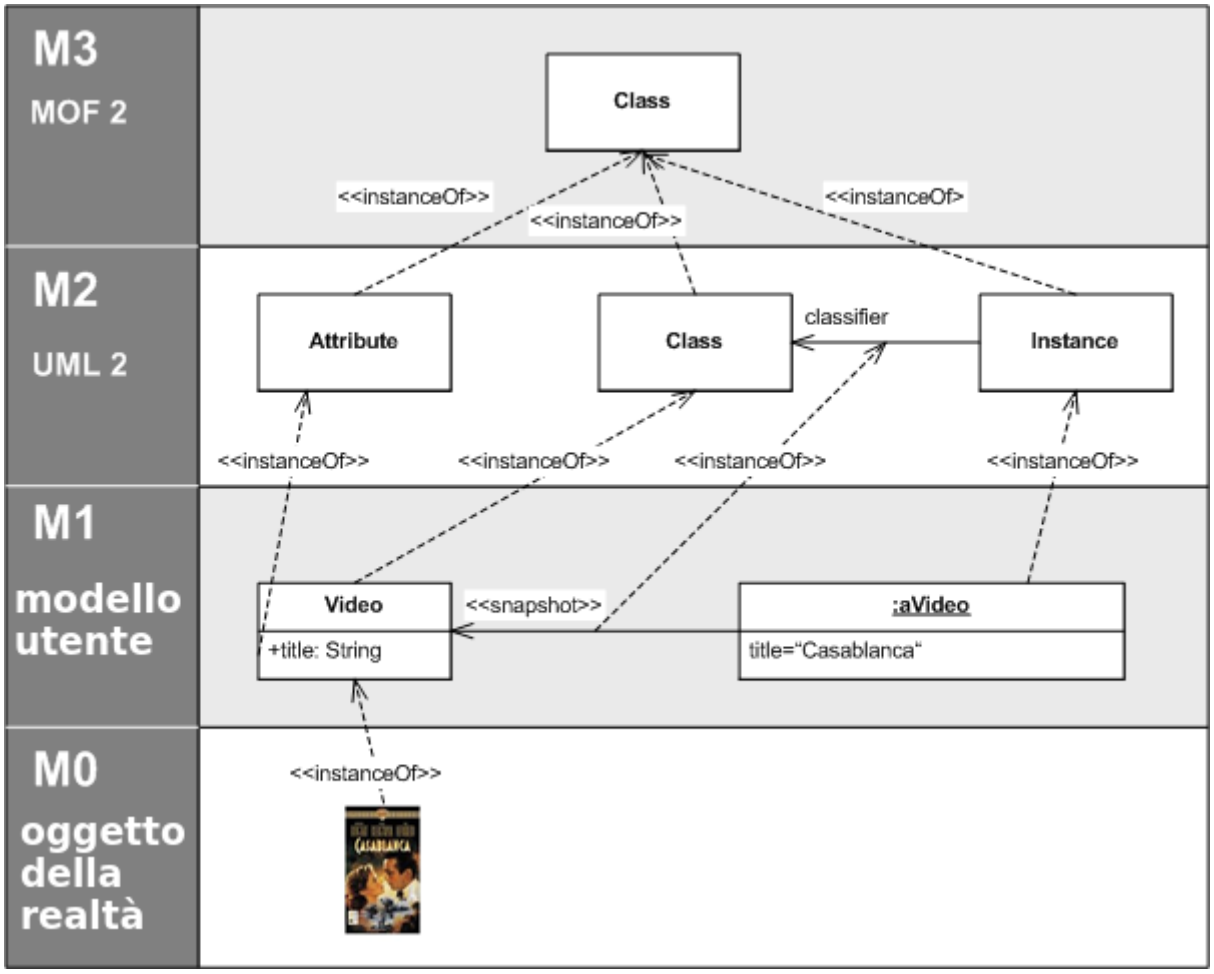
- **M0**: è la realtà da modellare;
- **M1**: è un modello che descrive la realtà;
- **M2**: è un modello che descrive modelli (metamodello), per esempio UML;
- **M3**: è un modello che descrive metamodelli (meta-metamodello), per esempio MOF.

Quindi, un oggetto al livello Mx è un'istanza di uno del livello superiore. Ovviamente, il meta-metamodello di livello M3 è progettato per essere istanza di sé stesso, ergo non esiste M4.

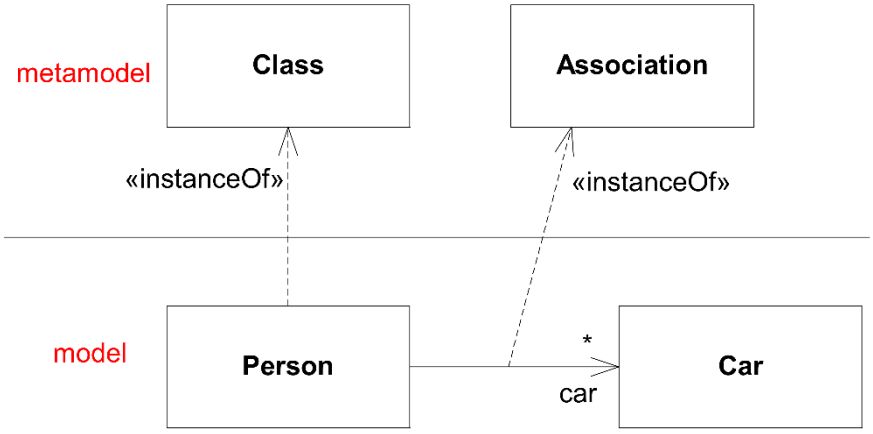
In genere, chi **usa UML crea modelli di livello M1**.

Tutti i linguaggi basati su MOF e i modelli con essi prodotti possono essere serializzati e scambiati tramite lo standard **XMI** (**XML Metadata Interchange**).

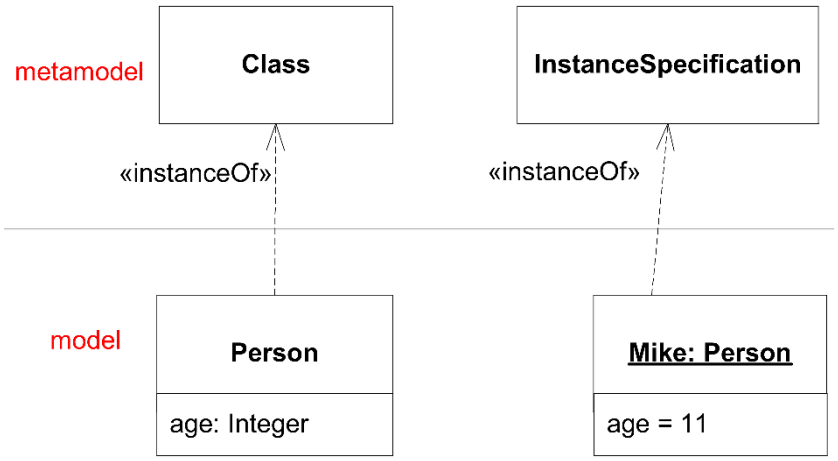
Un **esempio** dei livelli MOF.



Il livello M1 e M2 a confronto usando un diagramma UML:



Un altro esempio con M1 (modello utente) e M2 (metamodello UML):



8.2 – Diagrammi e 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.

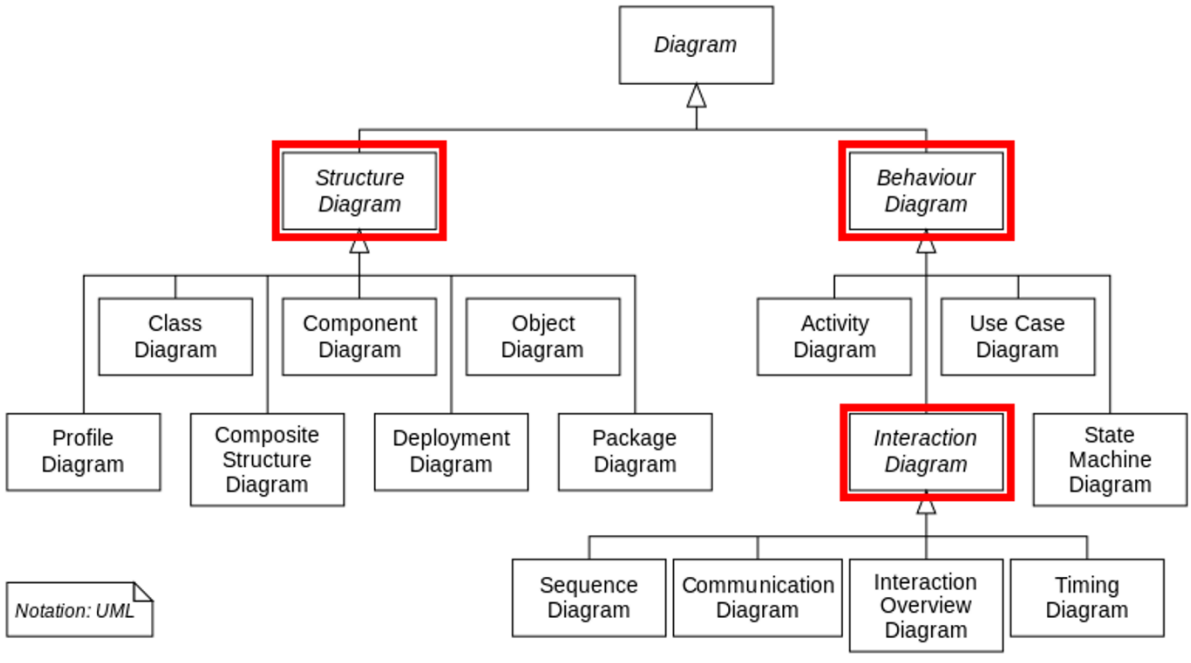
Le **viste** sono 4 più 1:

1. **Logical**: mette in risalto la scomposizione logica del sistema tramite classi, oggetti e loro relazioni;
2. **Development**: mostra l'organizzazione del sistema in blocchi strutturali (package, sottosistemi, librerie, …);
3. **Process**: mostra i processi (o thread) del sistema in funzione, e le loro interazioni;
4. **Physical**: mostra come il sistema viene installato ed eseguito fisicamente;
5. **Use case** (+1) la vista che agisce da "collante" per le altre. In altre parole, spiega il funzionamento desiderato del sistema.

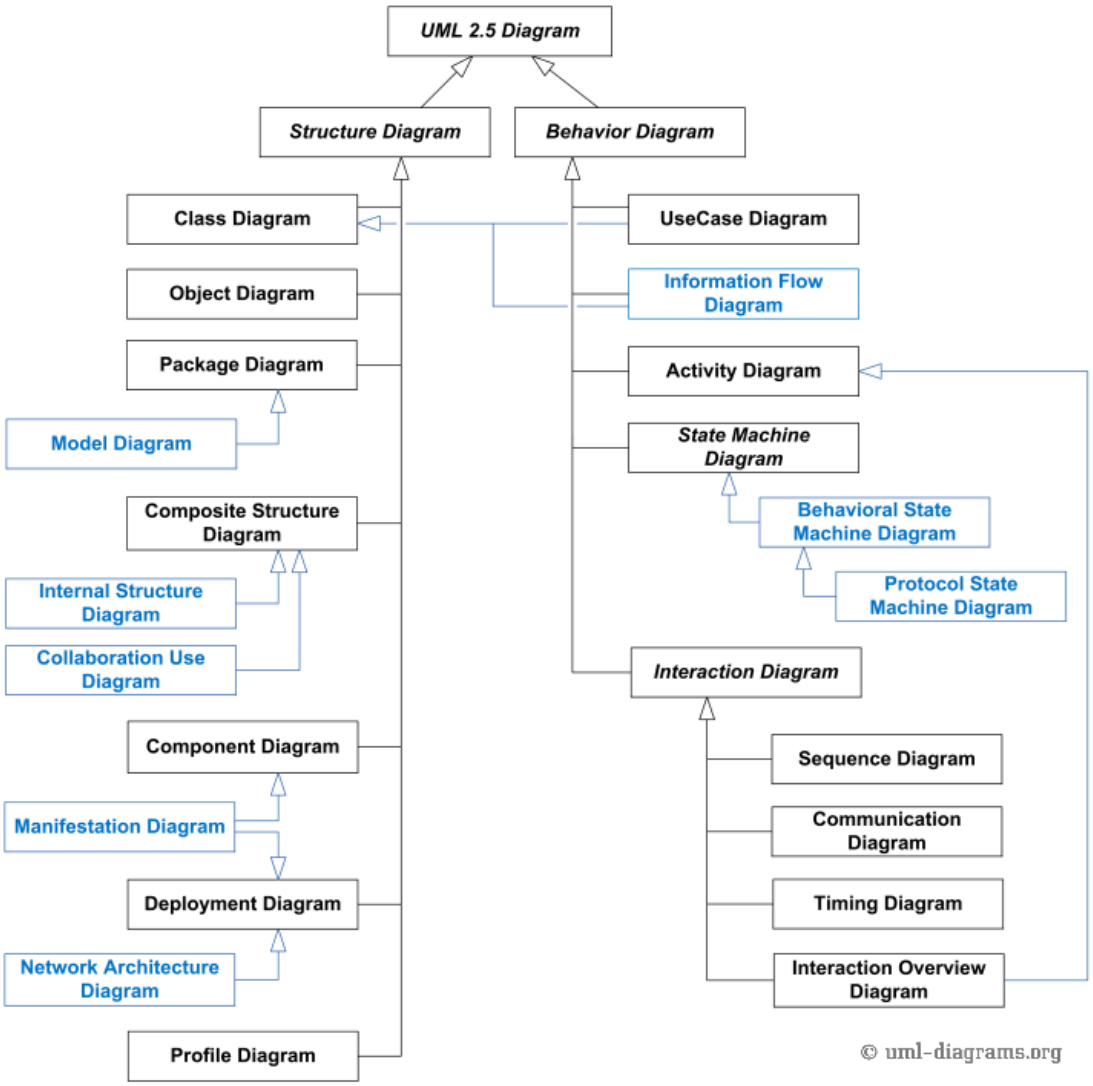
La **versione 2.5** di **UML** definisce **14 diagrammi** divisi in categorie:

- **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. Esiste un altro diagramma più dettagliato:
    - **Diagrammi di interazione**: mostrano come un certo comportamento viene realizzato dalla collaborazione delle entità in gioco.

I 14 diagrammi di UML 2:



Mentre, la **gerarchia dei diagrammi**:



Gli elementi in blu non sono parte della versione 2.5 di UML.

8.3 – Entità UML

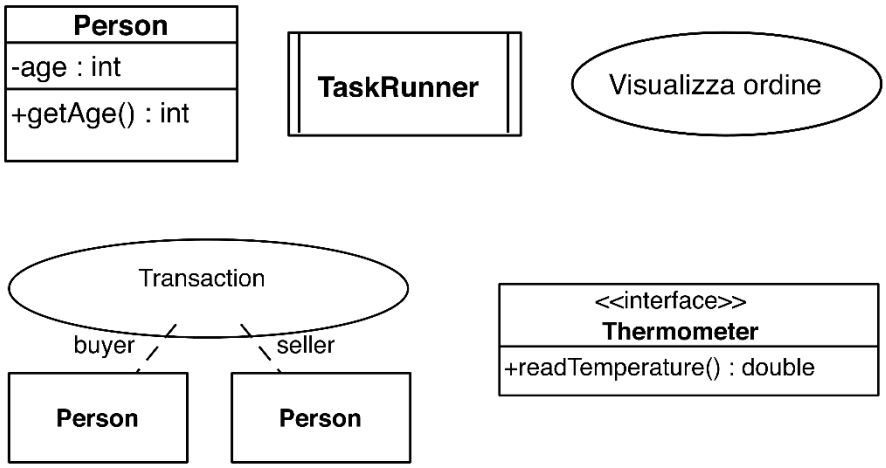
UML prevede diversi **tipi di entità** che possono essere organizzati in **quattro categorie**:

- **Strutturali**;
- **Comportamentali**;
- **Informative**;
- **Raggruppamento e contenimento**.

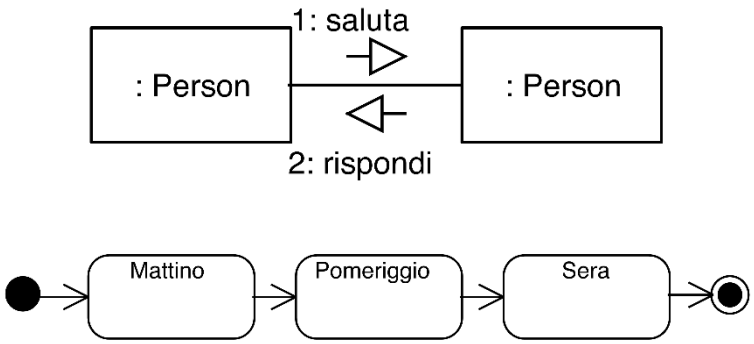
Una lista delle entità usate da ogni singolo diagramma è disponibile al link:

www.uml-diagrams.org/uml-25-diagrams.html

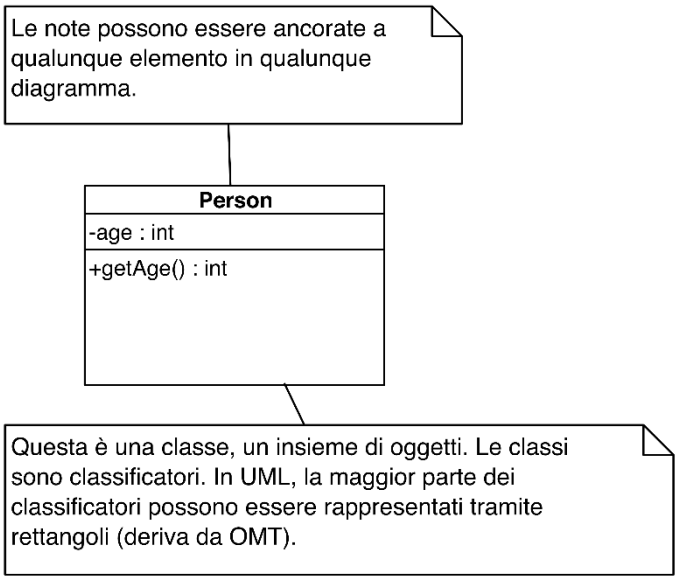
Le **entità strutturali** definiscono le "cose" del modello. Per **esempio**: le classi, classi attive, use-case, collaborazioni, interfacce.



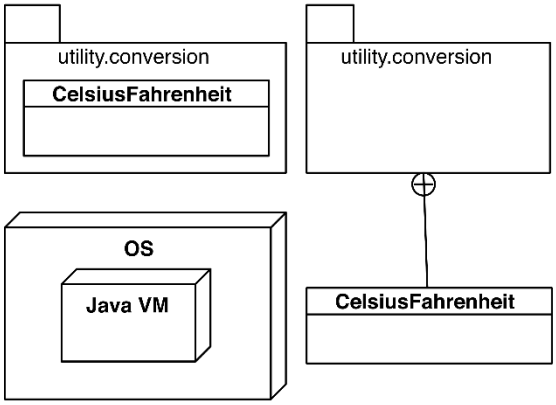
Le **entità comportamentali** descrivono il "*behavior*", ovvero le interazioni, le collaborazioni (o "*communication*"), scambio di messaggi, transizioni di stato, ecc.



Le **entità informative** non sono altro che note che hanno l'obbiettivo di migliorare la leggibilità.



I **package** raggruppano altri elementi e forniscono loro un *namespace*, il quale consente 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 (**entità di raggruppamento** e **contenimento**).

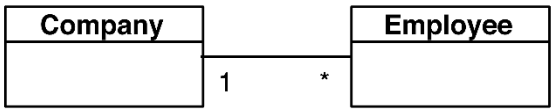


8.4 – Relazioni, frecce, stereotipi, OCL

Gli elementi del modello possono essere collegati da **relazioni**. Esse vengono rappresentate graficamente tramite linee, possono avere un nome e sono **4 sottotipi fondamentali**:

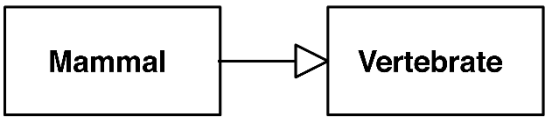
- **Association**;
- **Generalization**;
- **Dependency**;
- **Realization**.

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

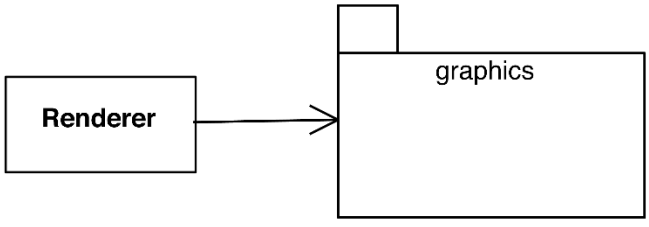


La **generalizzazione** (*generalization*) è una relazione tassonomica da un elemento specializzato verso un altro, più generale, dello stesso tipo.

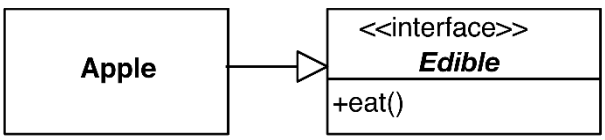
Per esempio, il figlio è sostituibile al genitore dovunque appaia, e ne condivide struttura e comportamento.



Una **dipendenza** (*dependency*) 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** (*realization*) è una relazione semantica: il *supplier* fornisce una specifica il *client* la realizza (esempio, l'implementazione di interfacce, templates, ecc.).



Un **metodo per capire il verso di tutte le frecce in UML** è il seguente.

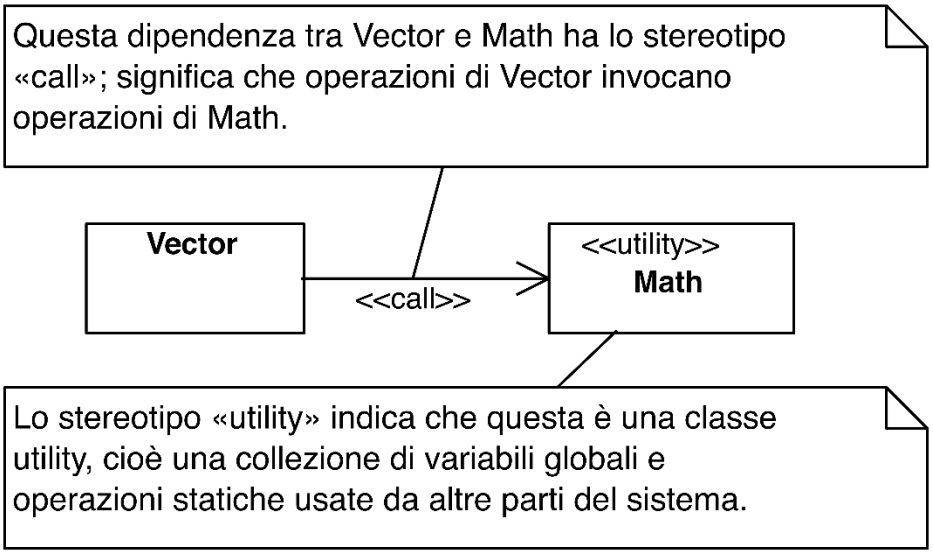
In UML, tutte le frecce vanno **da chi sa verso chi non sa** (dell'esistenza dell'altro).

In una **generalizzazione**, il figlio sa di estendere il genitore, ma non viceversa (il genitore non sa di essere esteso).

In una **dipendenza**, chi dipende sa da chi dipende, ma non viceversa.

In una **realizzazione**, chi implementa conosce la specifica, ma non il contrario.

In ogni diagramma UML esistono gli **stereotipi**. Hanno l'**obbiettivo** di rendere 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. Esempio $\ll import \gg, \ll utility \gg, \ll interface \gg$.



Gli stereotipi forniscono significato aggiuntivo ai costrutti UML e solitamente vengono utilizzati per adattare le UML a particolari ambiti e piattaforme di sviluppo.

Infine, gli stereotipi, i vincoli e le regole aggiuntive vengono raccolti in **profili**, che costituiscono uno dei principali meccanismi di estensione di UML.

L'**Object Constraint Language** (OCL) è un linguaggio, approvato e standardizzato, per la specifica di vincoli. Si usa assieme ad UML e a tutti i linguaggi dell'architettura MOF.

Non è obbligatorio, tuttavia aggiunge rigore formale al modello. Specifica condizioni che devono essere soddisfatte dalle istanze di una classe: i vincoli sono espressioni booleane considerate $true$.

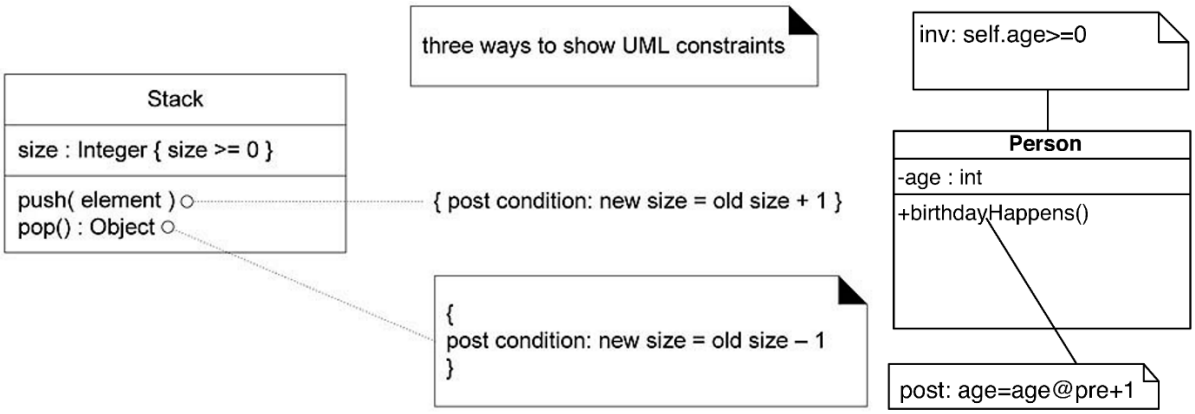
Un tool UML può usare OCL per validare il modello o per generare il codice.

Un **vincolo OCL** opera in un determinato **contesto**, specificando proprietà soddisfatte da tutte le istanze di quel contesto. Il contento può essere una classe, un suo attributo o una sua operazione. I vincoli hanno un tipo che descrive l'ambito della loro **validità**.

Per **esempio**, $context\ Car\ inv: fuel \geq 0$. Questo vincolo si applica alla classe Car ed è un invariante (sempre valido).

Esistono 6 **tipi di vincoli**:

1. ***inv***: invariante, sempre valido nel contesto.
2. ***pre***: nel contesto di un'operazione, una precondizione per la sua esecuzione.
3. ***post***: nel contesto di un'operazione, una postcondizione vera dopo l'esecuzione.
4. ***body***: definisce una query nel contesto.
5. ***init***: definisce il valore iniziale nel contesto.
6. ***derive***: definisce un attributo derivato dal contesto.



La dicitura "$@pre$" permette di accedere al valore precedente di un attributo.

9 – Conclusioni

La **modellazione** è indispensabile in qualunque progetto non banale.

UML è un linguaggio general-purpose per la modellazione.

Non è perfetto, tuttavia è potente e costituisce uno standard diffuso ed accettato.

Costruire un buon modello è difficile.