Introduzione a UML

Laboratorio di Sistemi e Processi Organizzativi
Gian Piero Favini

A.A. 2006-2007

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.

- 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 è normalmente distinto dal *processo* di sviluppo:
  - il linguaggio descrive cosa deve essere ottenuto;
  - il processo descrive i passi da intraprendere per ottenerlo;
  - insieme, linguaggio e processo definiscono un *metodo* di sviluppo.

Che cos'è UML (1)

- **UML**, Unified Modeling Language, è un linguaggio semiformale e grafico (basato su diagrammi) per
  - specificare
  - visualizzare
  - realizzare
  - modificare
  - documentare

  gli artefatti di un sistema (solitamente un sistema software).
- Un *artefatto* è un qualunque prodotto tangibile del progetto: sorgenti, eseguibili, documentazione, file di configurazione, tabelle di database, benchmark, ...

Che cos'è UML (2)

- 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

- 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 metodologi 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.

- Nel 1996, Booch, Rumbaugh e Jacobson, noti come i *Three Amigos*, sono incaricati da Rational di dirigere la creazione dell'Unified Modeling Language.
- 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), seguita da 2.1 (2006).
- La versione 2.2. è attualmente in corso di sviluppo.

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.

- **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.

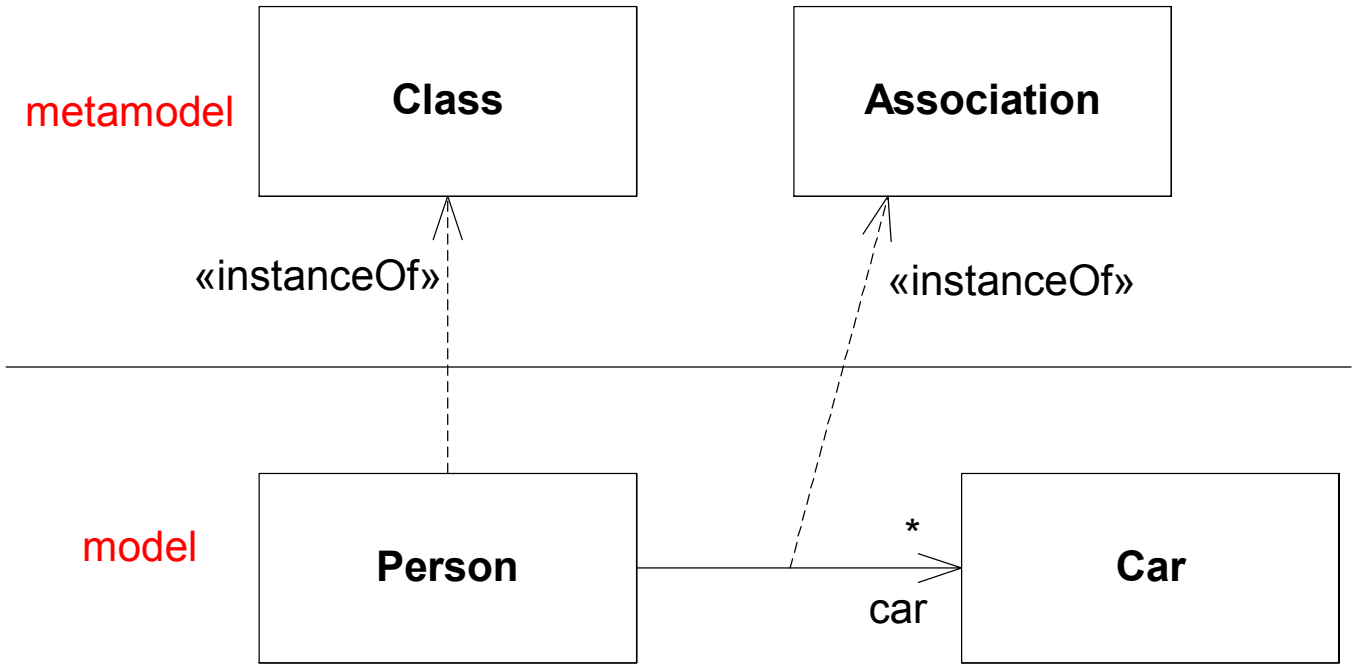
Meta-Object Facility

- 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 al tempo stesso *classe* e *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.

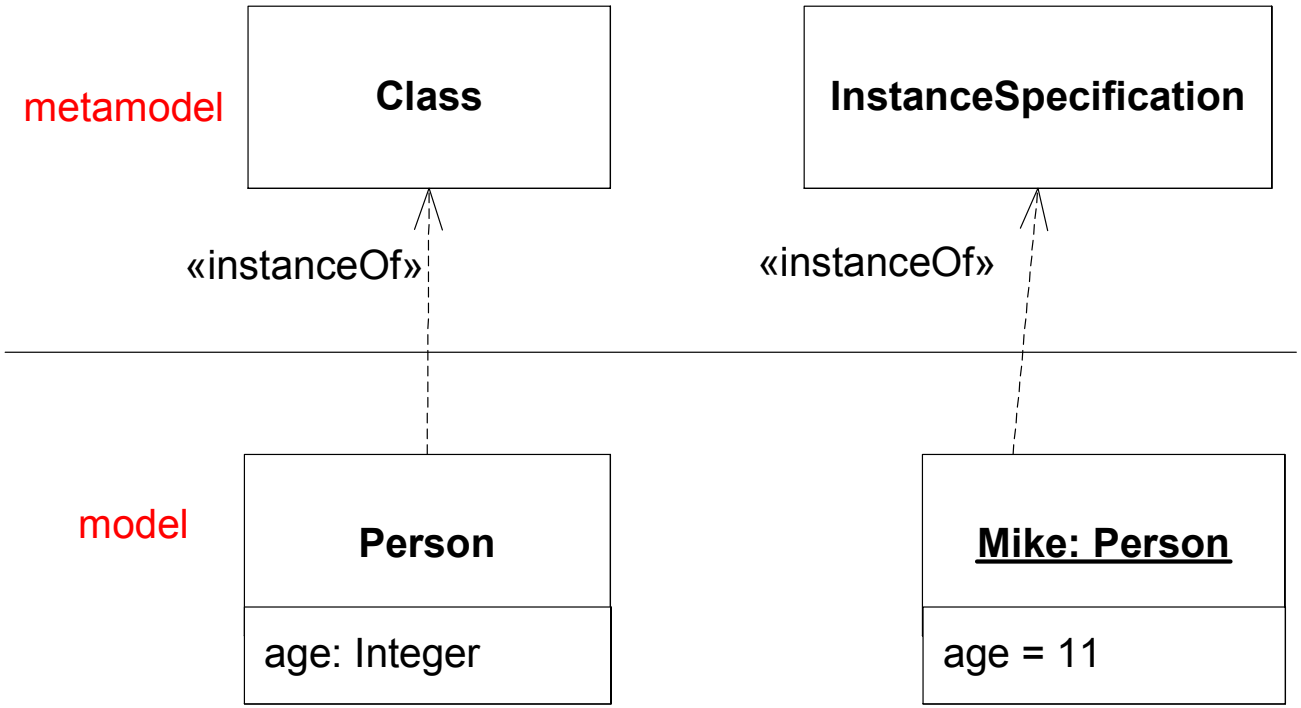
Il metamodello UML

- 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).

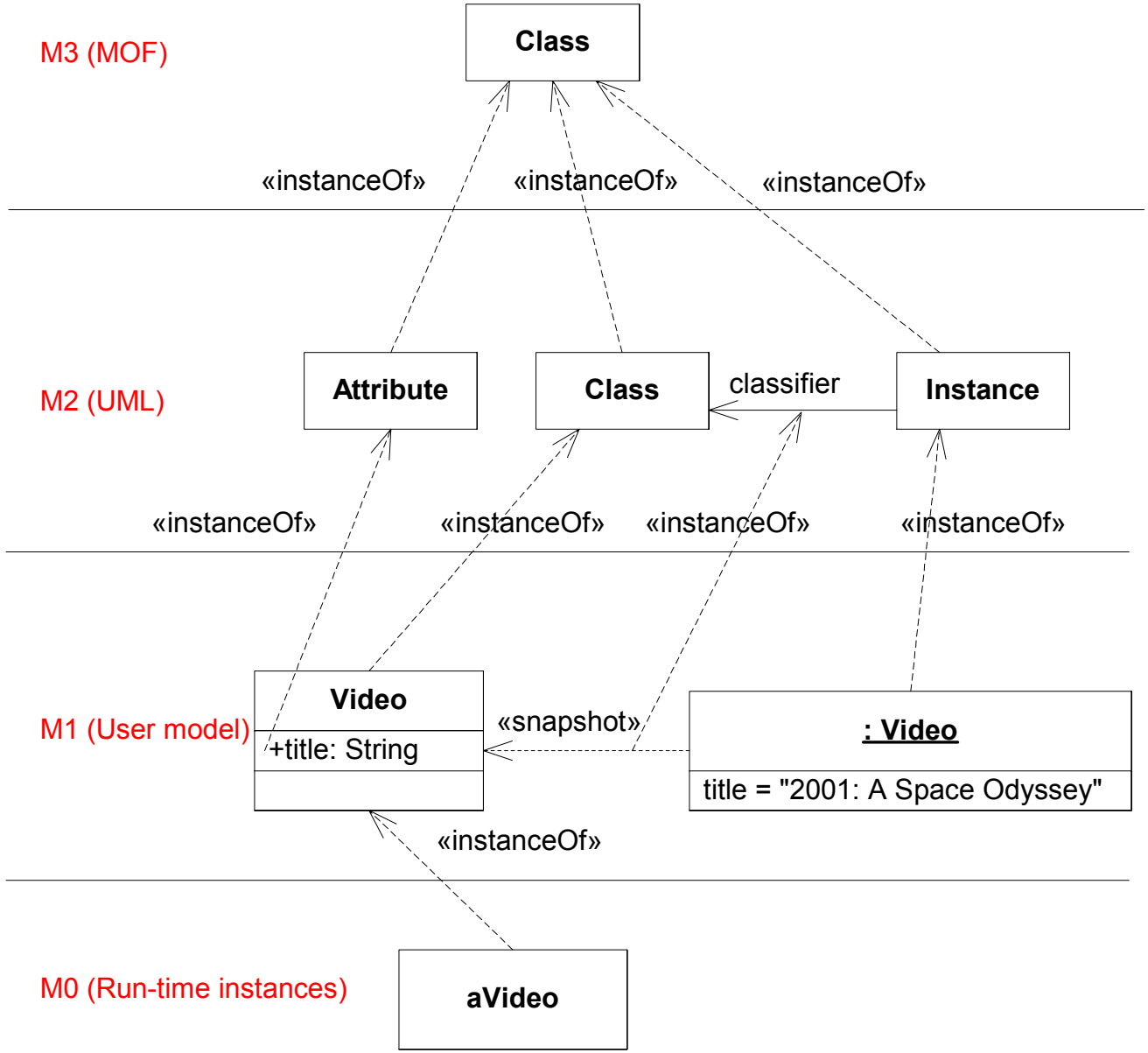
- Semplificando molto, si può dire che UML è definito tramite un modello UML (o piuttosto, usando un modello M3 che usa primitive comuni a UML).
- In qualunque momento, un oggetto al livello Mx è un'istanza di uno del livello superiore.
- Il meta-metamodello di livello M3 è progettato per essere istanza 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)


M3 (MOF)
Class
«instanceOf»
«instanceOf»
«instanceOf»
M2 (UML)
Attribute
Class
classifier
Instance
«instanceOf»
«instanceOf»
«instanceOf»
«instanceOf»
M1 (User model)
Video
+title: String
«snapshot»
: Video
title = "2001: A Space Odyssey"
«instanceOf»
M0 (Run-time instances)
aVideo

I diagrammi e le viste

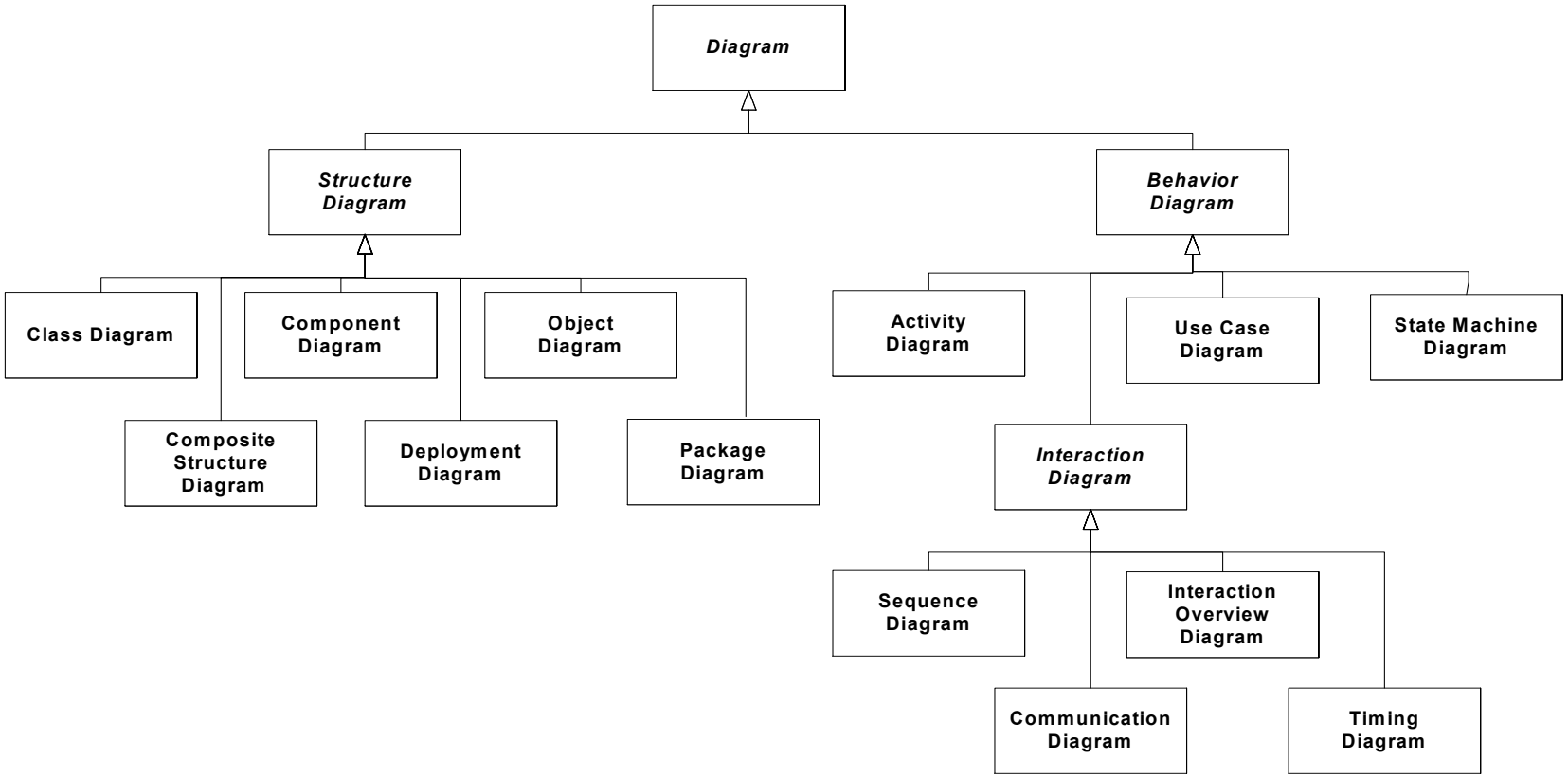
- Un diagramma è la rappresentazione grafica di una parte del modello.
- Fornisce una *vista* di un sistema o una sua parte, cioè ne mette in risalto diverse proprietà.
- 4+1 viste (Kruchten, 1995):
  - **Logical:** mette in risalto la scomposizione logica del sistema tramite classi, oggetti e loro relazioni
  - **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 definisce 13 diagrammi (contro i 9 di UML 1.x), divisi in due categorie:
  - **Structure diagrams:** come è fatto il sistema; forniscono le viste Logical, Development e Physical
  - **Behavior diagrams:** come funziona il sistema; forniscono le viste Process e Use case

Structure	Behavior
Class diagram	Use Case diagram
Object diagram	Activity diagram
Package diagram*	State Machine diagram
Composite Structure diagram*	Sequence diagram
Component diagram	Communication diagram
Deployment diagram	Interaction Overview diagram*
	Timing diagram*

* : non esiste in UML 1.x

Tassonomia dei 13 diagrammi di UML 2

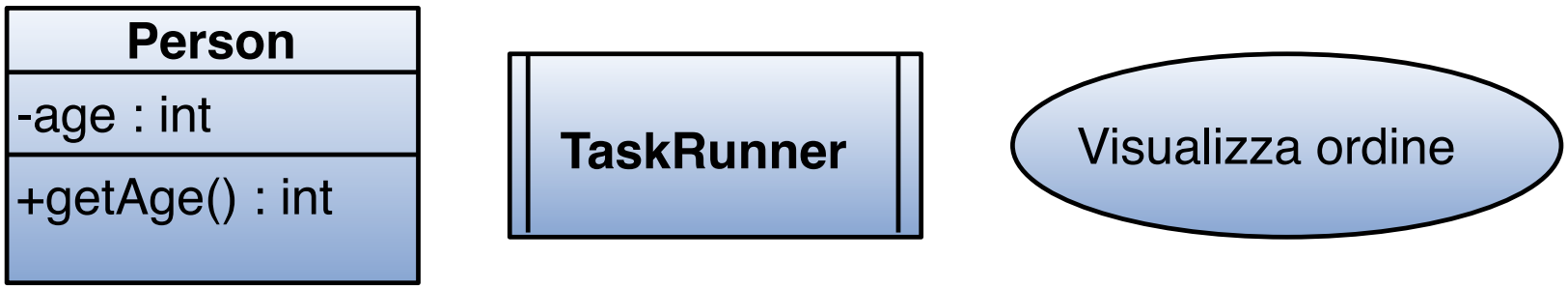


Le primitive di UML

- Un modello UML contiene tre tipi di elementi le cui sottoclassi possono apparire nei diagrammi:
  - **Classifier:** un classificatore è un insieme di cose (*things*).
  - **Event:** un evento è un insieme di possibili avvenimenti (*occurrences*).
  - **Behavior:** un comportamento è un insieme di esecuzioni (*executions*).
- Il modello non contiene le singole cose, avvenimenti ed esecuzioni, perché quelle appartengono alla realtà (M0).
- Diagrammi di tipo diverso si distinguono per gli elementi che ammettono all'interno.

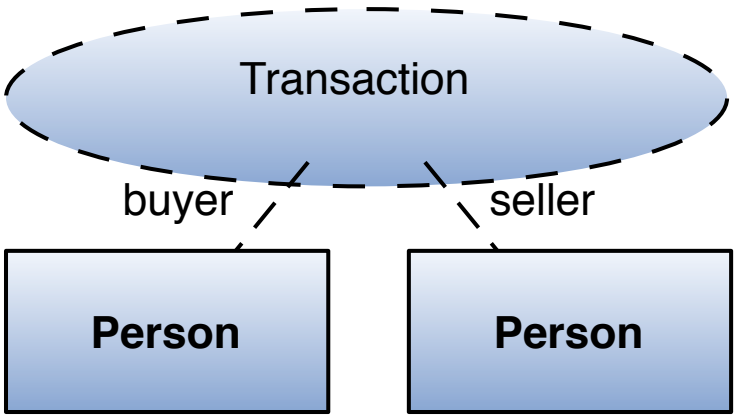
UML: entità strutturali (1)

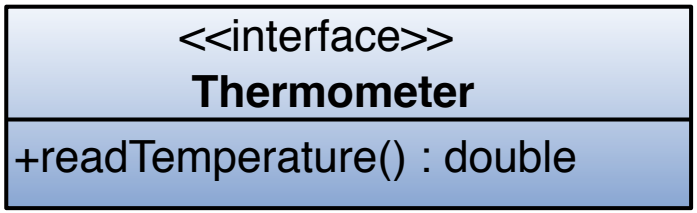
- **Class:** Un classificatore che raduna oggetti con gli stessi attributi, operazioni, relazioni e semantica.
- **Active class:** Una classe con un comportamento concorrente a quello di altri elementi (processo/thread associato).
- **Use case:** Un classificatore contenente offerte di comportamento che producono un risultato osservabile.



UML: entità strutturali (2)

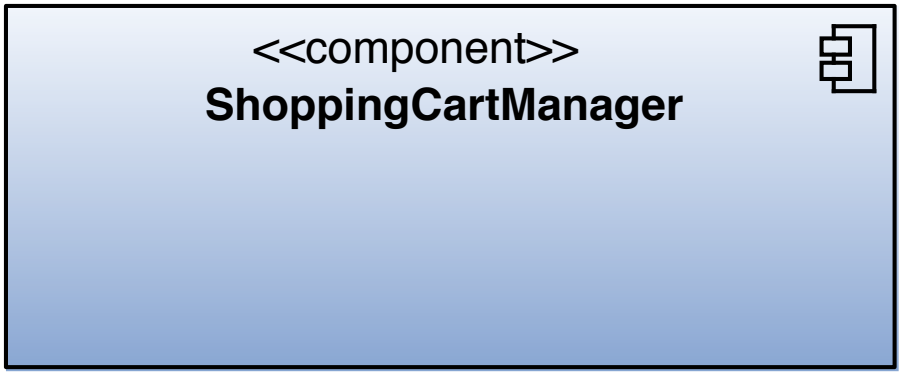
- **Collaboration:** Una collezione di ruoli che formano un comportamento cooperativo: definisce un'interazione e può realizzare un'operazione o un caso d'uso.
- **Interface:** Una collezione di operazioni che possono essere richieste oppure offerte da altri elementi.

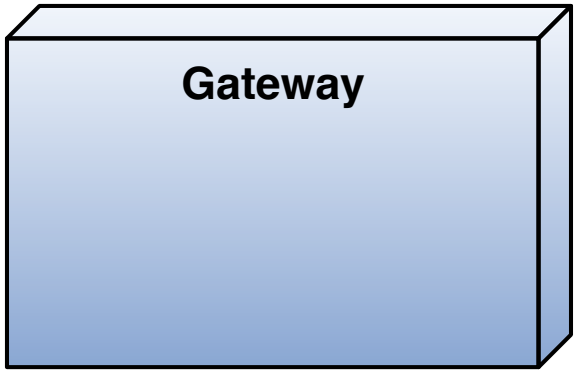




UML: entità strutturali (3)

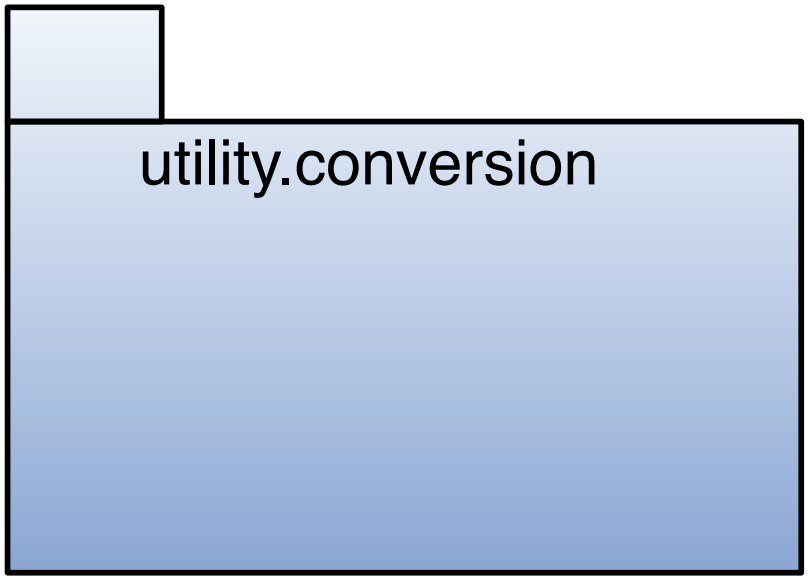
- **Component:** Una parte modulare di un sistema che fornisce e richiede interfacce ed è sostituibile nel suo ambiente.
- **Node:** Una risorsa computazionale sulla quale viene effettuato il deployment degli artefatti del sistema.





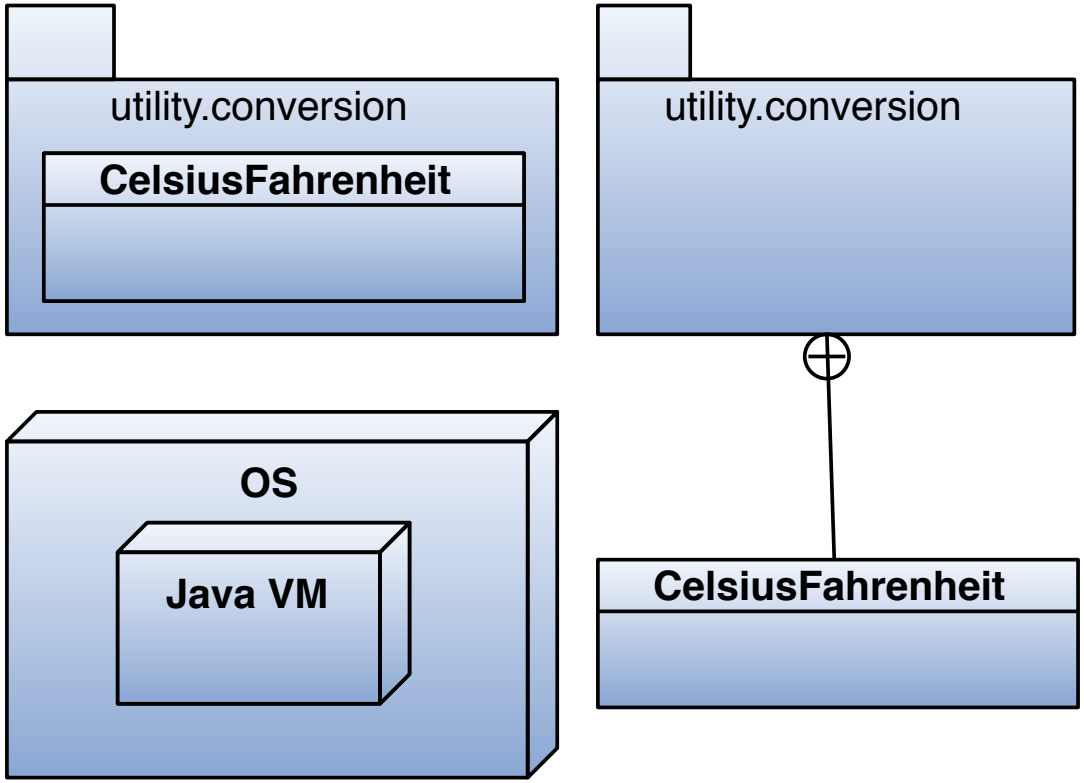
UML: entità di raggruppamento

- Rappresentate dai *package*.
- Un package raggruppa altri elementi e fornisce loro un *namespace* (un'entità che permette di identificare ogni elemento con il suo nome).



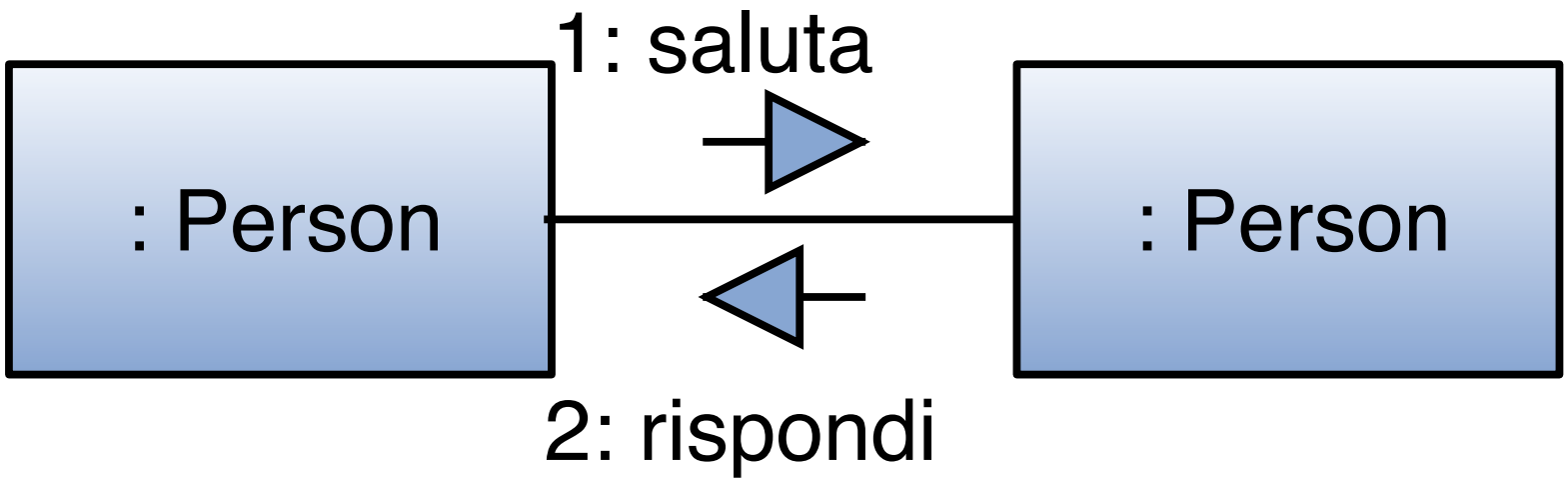
UML: contenimento

- In UML, moltissimi elementi possono contenere altri elementi al loro interno, formando una struttura gerarchica.
- Questo può essere rappresentato graficamente.



UML: entità comportamentali (1)

- **Interaction:** Un'unità di comportamento basata sullo scambio di messaggi osservabili tra elementi del modello.
- Non esiste un'unica primitiva grafica per rappresentare un'interazione: la seguente è rappresentata tramite un diagramma di comunicazione (collaborazione in UML 1.x).

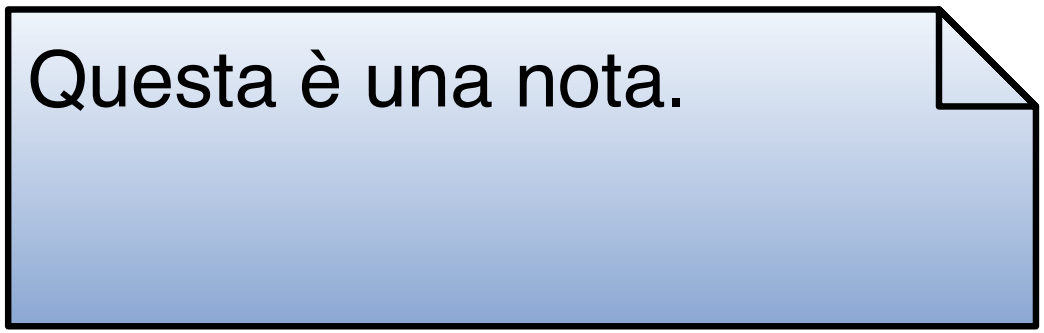


UML: entità comportamentali (2)

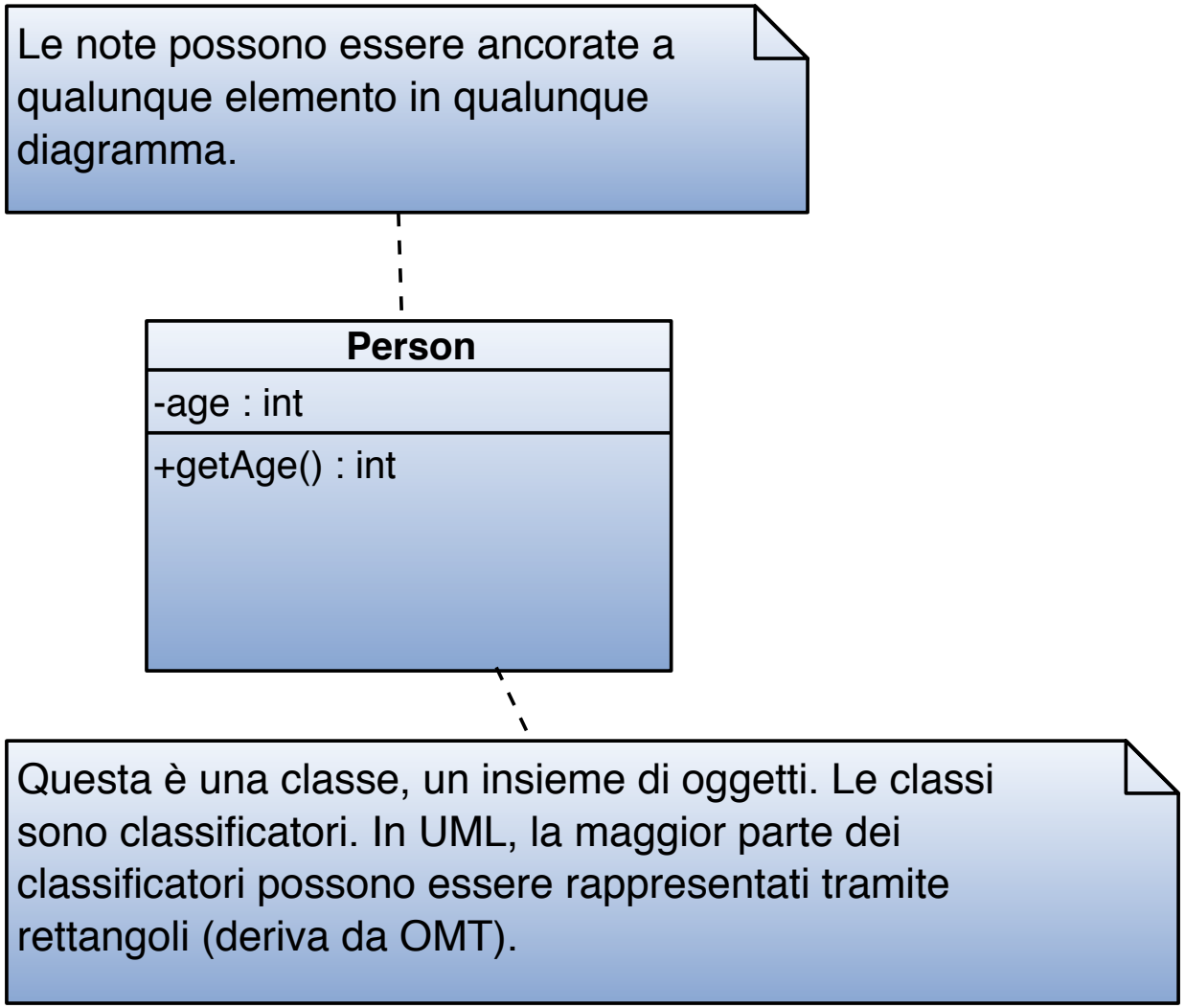
- **State machine:** Descrive il comportamento di un elemento del modello tramite transizioni discrete in un insieme finito di stati.
- Rappresentata tramite State Machine diagram (Statechart diagram in UML 1.x).

Entità informative

- Uno degli scopi principali della modellazione è la leggibilità: se un diagramma non è leggibile e informativo, sconfigge il suo stesso scopo.
- UML fornisce una entità informativa che non ha effetti sul modello ma migliora la leggibilità: le *note*.



Usare le note



Relazioni

- Le relazioni collegano due o più elementi del modello.
- Rappresentate graficamente tramite linee.
- Una relazione può essere un estremo di un'altra relazione.
- Possono avere un nome.
- Quattro sottotipi fondamentali:
  - **Association**
  - **Generalization**
  - **Dependency**
  - **Realization**

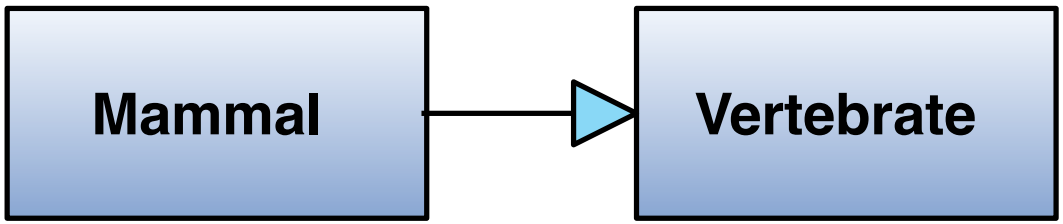
Association

- Descrive l'esistenza di un nesso tra le istanze di classificatori.
- Formalmente, un'associazione definisce un insieme di tuple i cui valori si riferiscono ad istanze.

- Varie caratteristiche opzionali (nell'esempio, *molteplicità*).
- Una compagnia può dare lavoro a qualunque numero di impiegati, ma ciascun impiegato può lavorare per una sola compagnia.

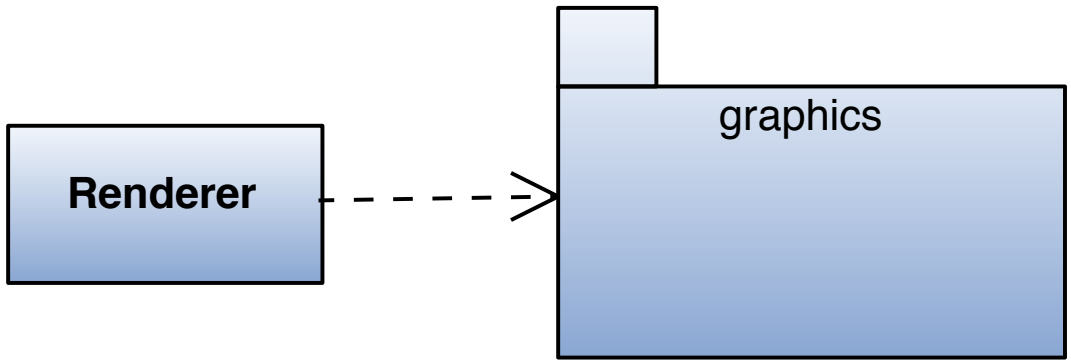
Generalization

- 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.
- Tra i due elementi esiste una relazione "is-a" che corrisponde all'ereditarietà nella programmazione object-oriented.



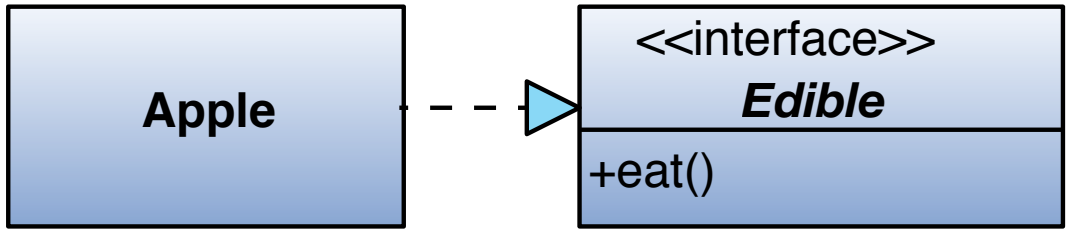
Dependency

- Relazione semantica; la freccia parte da un elemento (o più), detto *client*, e si dirige verso un altro (o più), detto *supplier*.
- Significa che il client dipende, semanticamente o strutturalmente, dal supplier, cioè che variazioni alla specifica del supplier possono cambiare quella del client.



Realization

- Relazione semantica; il supplier fornisce una specifica, il client la realizza.
- Può rappresentare implementazione di interfacce, raffinamento di concetti, ottimizzazioni, trasformazioni, templates, ...



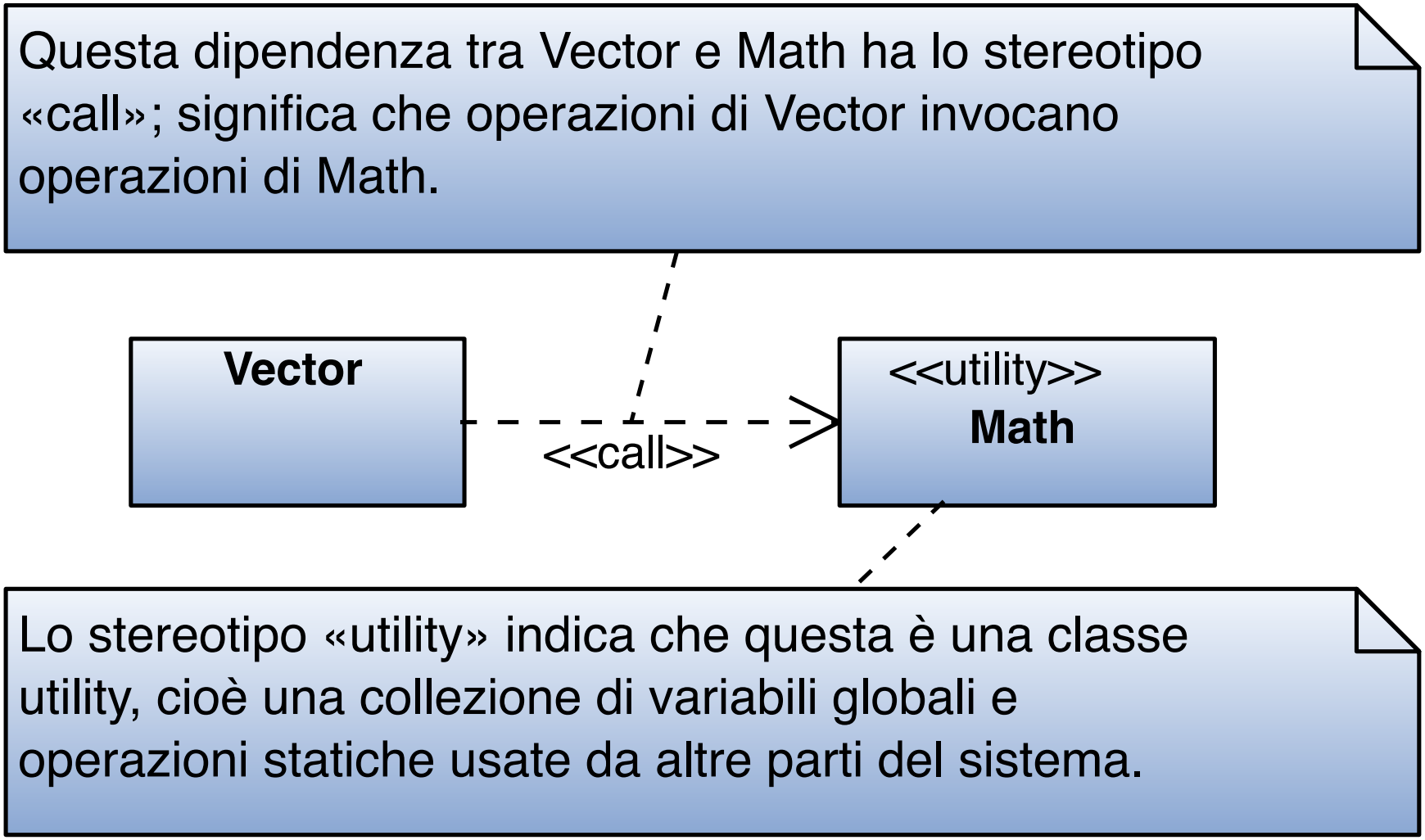
Le frecce in UML

- 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



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.
- Alcuni profili disponibili:
  - CORBA
  - J2EE
  - SysML

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*.
- OCL può anche eseguire query sul modello (interrogazioni senza effetti collaterali).
- Un tool UML può usare OCL
  - per validare il modello
  - per generare codice

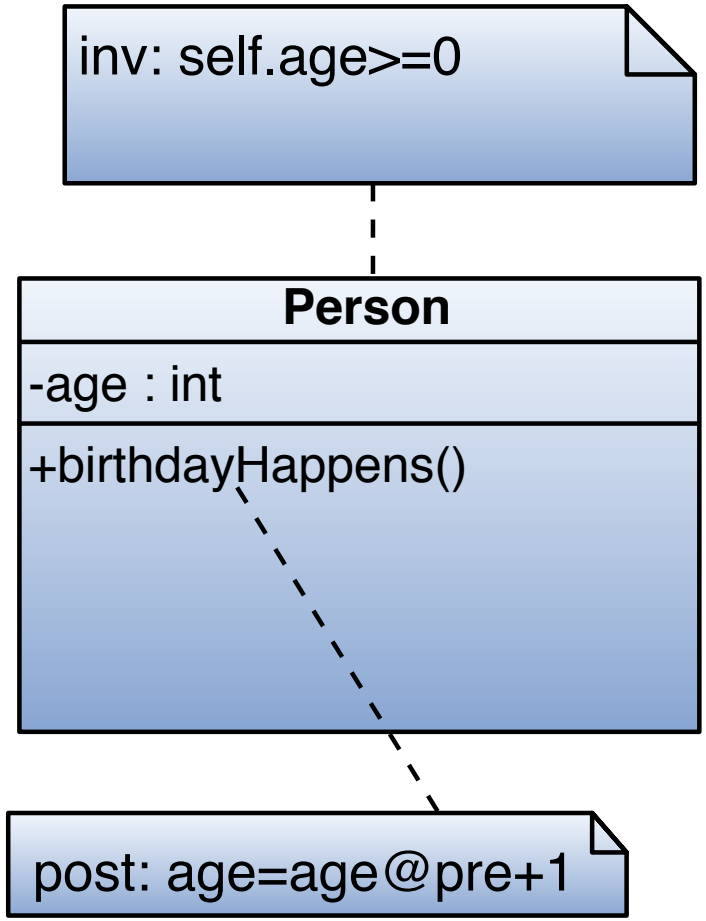
OCL (2)

- 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 precondizione 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



(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.
  - Che cos'è un *buon* modello?