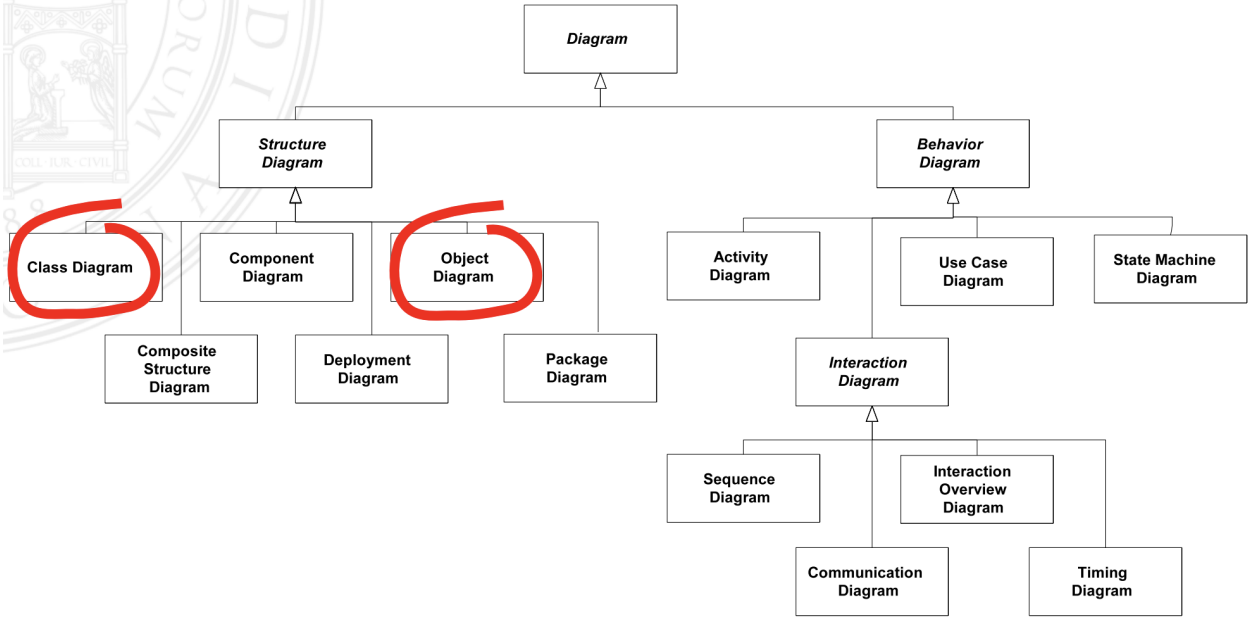
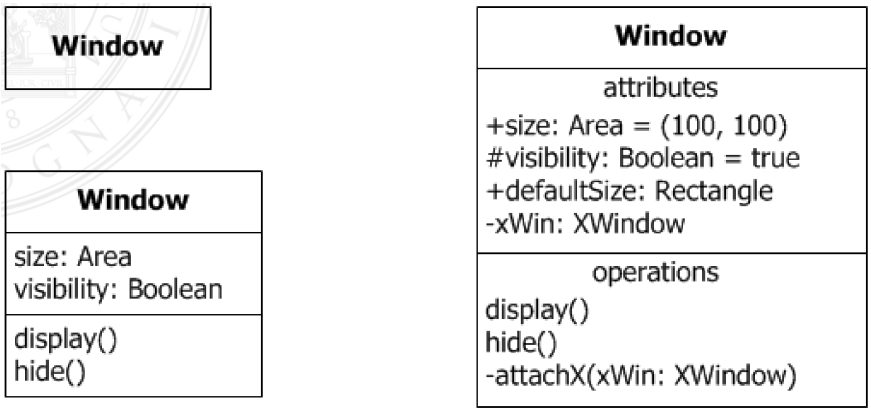
UML class diagram



Nota Bene: si considera anche *Object Diagram*, essendo molto simile al diagramma delle classi ma considerando le singole istanze.

Una classe è un tipo che rappresenta oggetti che condividono caratteristiche comuni. Viene raffigurata come un rettangolo dal contorno solido con il nome della classe al suo interno ed è potenzialmente divisa in compartimenti separati da linee orizzontali sotto il nome. I compartimenti tipici sono il compartimento delle *operazioni* e il compartimento degli *attributi*. Alcuni esempi di rappresentazione grafica di una classe, con diversi livelli di complessità:



Proprietà e operazioni

Le **proprietà** sono caratteristiche strutturali di una classe. Una sintassi semplificata può essere:

[+,-,#,~][/]<name>[:<type>][<mult>]

Le **operazioni** sono caratteristiche comportamentali di una classe. Una sintassi semplificata può essere:

[+,-,#,~]<name>[<params>][:<type>]

In entrambi i casi, il nome è **obbligatorio**, mentre i valori inclusi in **[]** sono **opzionali**.

Visibilità

- +: *public*, visibile a tutti
- **-**: *private*, visibile soltanto nel suo ambiente
- #: *protected*, visibile agli elementi che possiedono una relazione di generalizzazione
- ~: *package*, visible a tutti gli elementi nel package

Moltiplicità

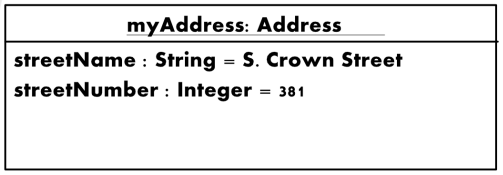
<multiplicity-range> ::= [<lower>...] <upper>

In questo caso, in presenza di *****, si considera un limite superiore infinito. Vengono spesso utilizzate per indicare la moltiplicità i seguenti valori:

- **1**: esattamente 1
- **0...1**: potrebbe non essere presente, o essere presente una sola volta
- **1...***: potrebbe esserne presente uno solo, o non si sa di preciso la quantità
- *****: non si sa di preciso la quantità

Istanze

Le **istanze** (o **oggetti**) sono, insomma, delle istanze. Obbediscono alla **struttura della classe** della quale sono una istanza. Gli oggetti di una classe devono avere un valore per ogni attributo che fa parte di quella classe, seguendo le caratteristiche dell'attributo, come il tipo e la moltiplicità.
La notazione grafica è simile a quella delle classi ma il nome compare sottolineato ed è una concatenazione del nome dell'istanza (se presente), due punti (':') e la classe della quale è una istanza.



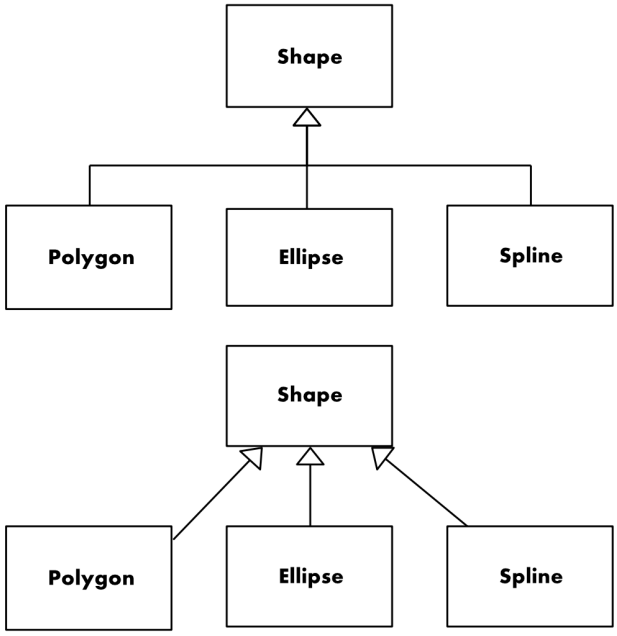
Relazioni

Molte relazioni possono essere rappresentate in diagramma di classi:

- **Generalizzazione**
- **Dipendenza**
- **Realizzazione**
- **Associazione**
- **Aggregazione**
- **Composizione**

Generalizzazione/specializzazione

Generalizzazione/specializzazione sono concetti di base nel modello *MOF* che assuomono un comportamento più dettagliato quando utilizzato tra classi. Ogni generalizzazione mette in relazione una specifica classe con una classe più generale con un meccanismo simile all'ereditarietà. Esiste una relazione "*is-a*" tra i due elementi, corrispondente all'ereditarietà nei linguaggi di programmazione.



Nota Bene: le due soluzioni hanno lo stesso significato, ma il primo è più leggibile.

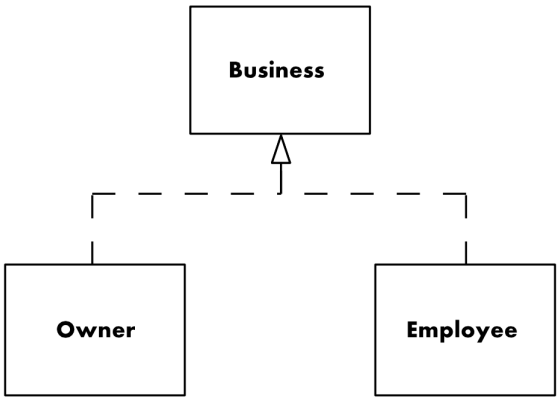
Dipendenza

Una dipendenza indica una relazione fornitore/cliente tra elementi del modello in cui **la modifica di un fornitore può influire sugli elementi del modello del cliente**. Una dipendenza implica che la semantica dei clienti non è completa senza i fornitori. Una dipendenza viene visualizzata come una freccia tratteggiata tra due classi. La freccia può essere etichettata con una parola chiave o uno stereotipo facoltativo e un nome facoltativo.

Nota Bene: se dovessimo modificare "Line Item" si potrebbe dover modificare anche "Order", data la sua dipendenza da "Line Item".

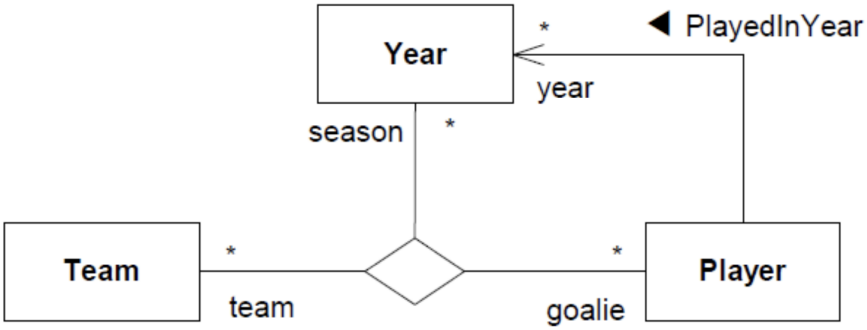
Realizzazione

Una realizzazione è una sorta di dipendenza e viene rappresentata graficamente come una linea tratteggiata con una freccia triangolare all'estremità che corrisponde all'elemento realizzato. Il realizzante deve realizzare, o implementare, il comportamento che l'altro specifica.



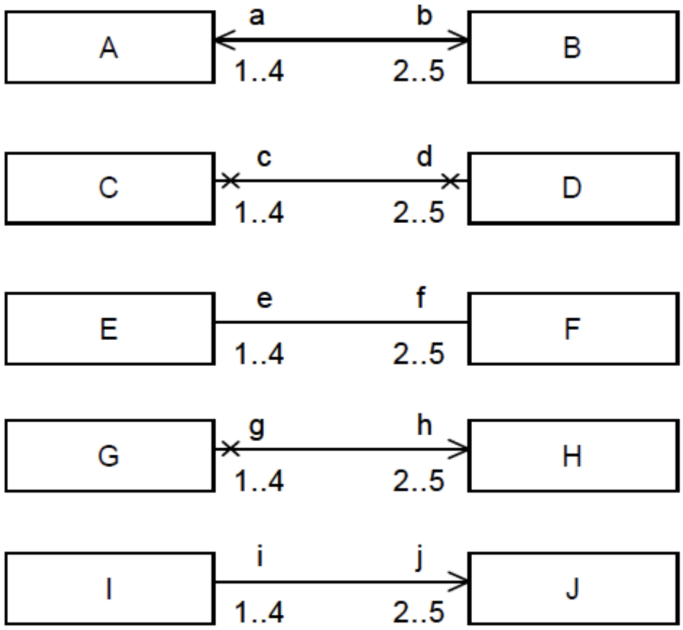
Associazione

Un'associazione afferma che possono esserci collegamenti tra istanze dei tipi associati. Un collegamento è una tupla con un valore per ciascuna estremità dell'associazione. Le associazioni hanno nomi che possono essere visualizzati ed etichette con frecce che aiutano a leggere la direzione dell'etichetta. Le estremità dell'associazione possono avere nomi, molteplicità e frecce o croci di navigazione.



Spiegazione sintetica: "PlayedInYear" è una etichetta. Tra Year e Player abbiamo un'associazione binaria, tra Year e Team e Player abbiamo un'associazione ternaria. Osserviamo ora l'associazione tra Player e Year; dato un player si può facilmente risalire agli year corrispondenti, ma non vale il contrario.

Navigazione: frecce e croci

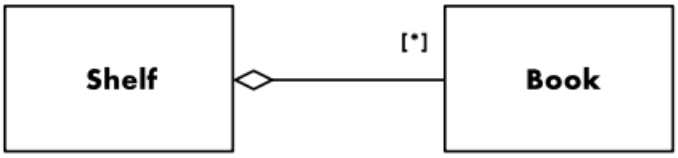


Il numero indica quanti elementi corrispondono agli elementi dell'altro insieme. Le frecce indicano che gli elementi corrispondenti sono facilmente identificabili. In caso di croce invece, non saranno facilmente identificabili. Se non troviamo croci o frecce, non si sa la difficoltà nel trovare gli elementi corrispondenti. Esempio:

La croce da Person è presente perchè è difficile da un cane risalire al suo padrone, al contrario è più semplice. '*' indica che ogni Person può avere zero o più cani.

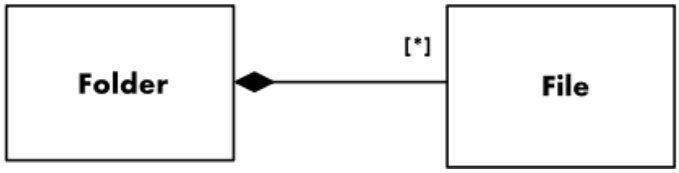
Aggregazione

Un'aggregazione (aggregazione condivisa) ci dice che una parte dell'istanza è indipendente dal composito.



Composizione

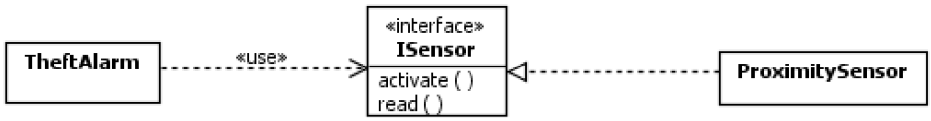
La composizione (aggregazione composita) è un'associazione binaria in parte o per intero. L'oggetto composito è responsabile dell'esistenza e della memorizzazione degli oggetti composti.



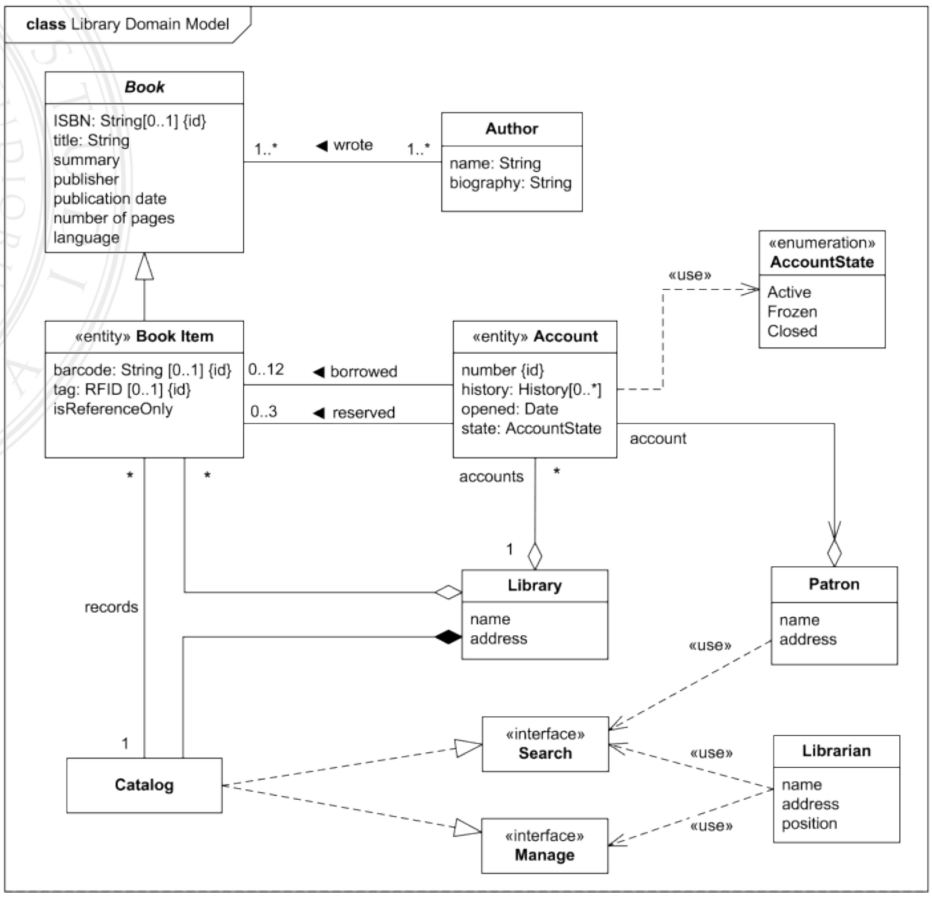
Classi astratte e interfacce

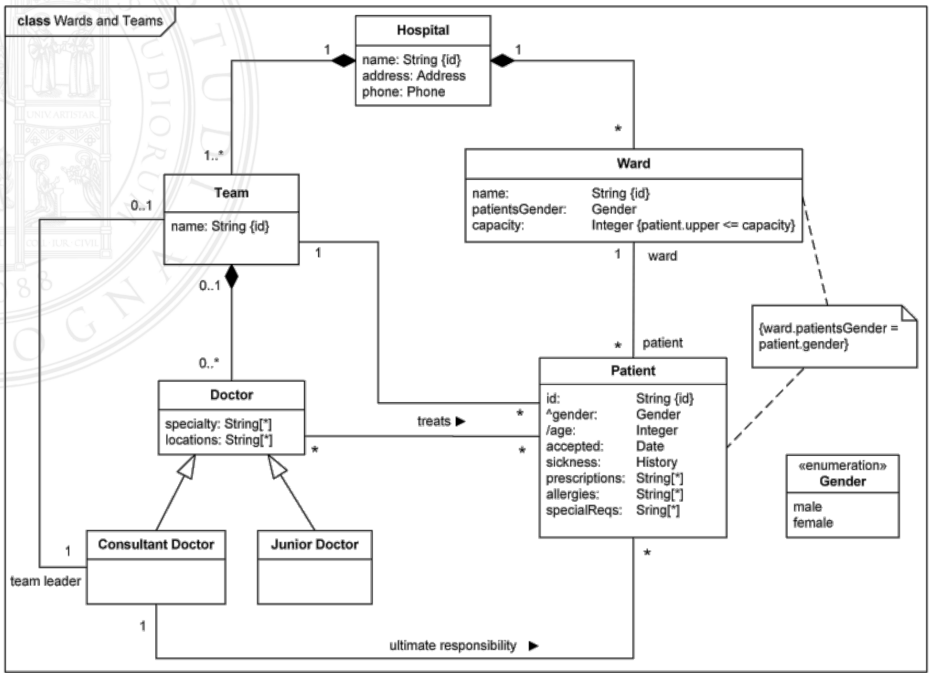
Una classe astratta non ha istanze dirette: le sue istanze sono istanze di una delle sue specializzazioni. Il nome di una classe astratta è mostrato in *corsivo* o/e utilizzando l'annotazione testuale abstract.

Le interfacce dichiarano servizi coerenti implementati dalle classi che li implementano. Un'interfaccia specifica un contratto; qualsiasi istanza di una classe che realizza l'interfaccia dovrà adempiere a tale contratto.

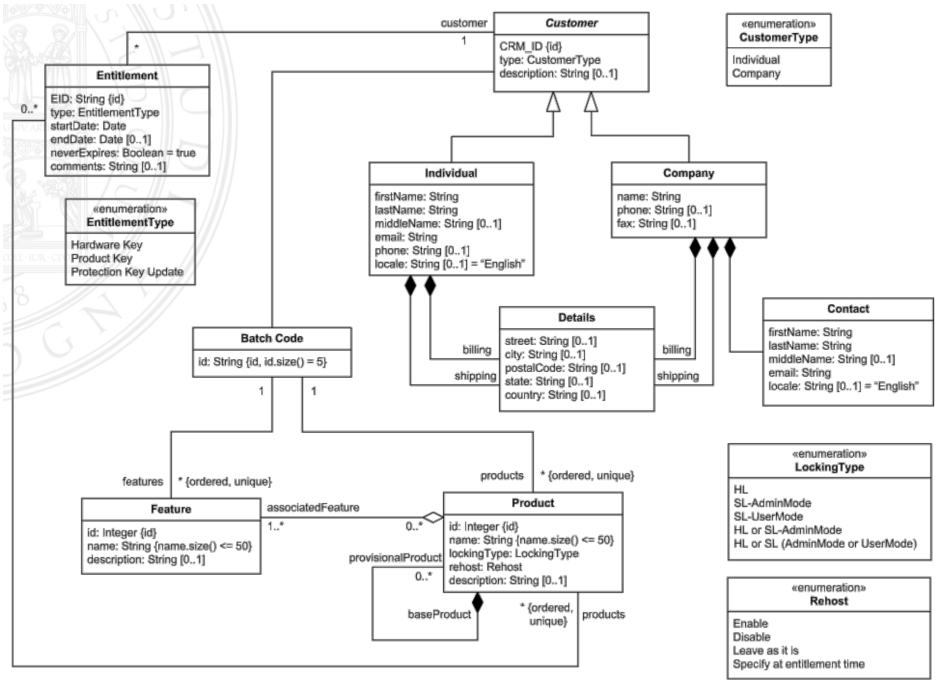


Esempi di modelli di dominio (modello di analisi)





Esempi di modelli di design



Modello di analisi/dominio

Nella programmazione Object Oriented il modello di dominio (oggetto) rappresenta il **concetto** nel dominio del problema, le loro **caratteristiche** e come si correlano tra loro. È un **dizionario visivo** per il dominio problematico.

L'artefatto risultante è (di solito) un diagramma di classi UML. L'approccio più elementare per identificare le **classi** inizia con l'analisi di tutti i **nomi** e le **frasi nominative** che fanno parte della descrizione del problema. I **verbi** vengono analizzati successivamente per identificare **responsabilità e collaborazioni**. Questi passaggi vengono iterati per rifinire il modello (dividere classi, aggregarle, transformarle in proprietà,...).

Esercizio

ciao