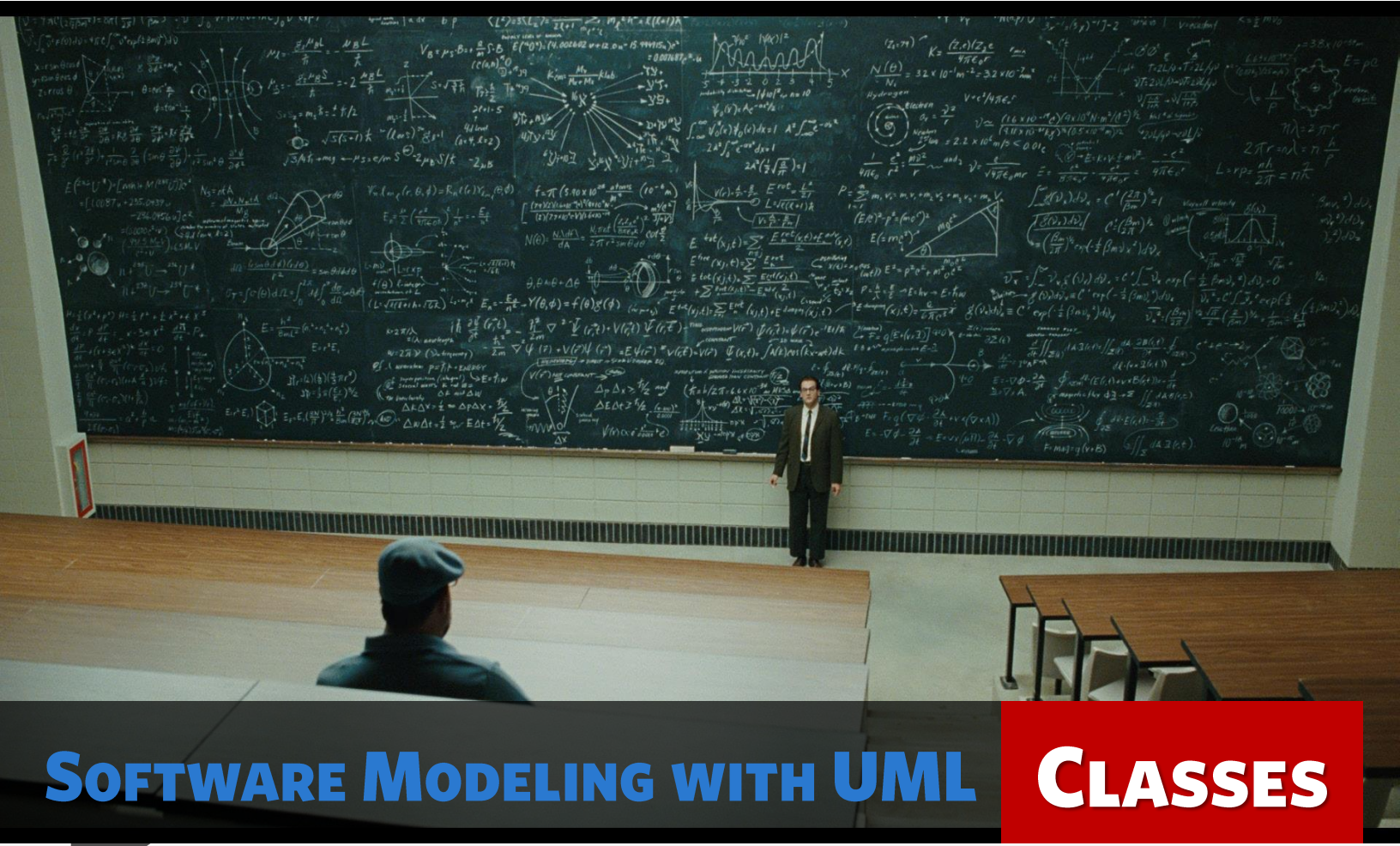


Software Modeling with UML

Classes

Shaping the static structure of a software system

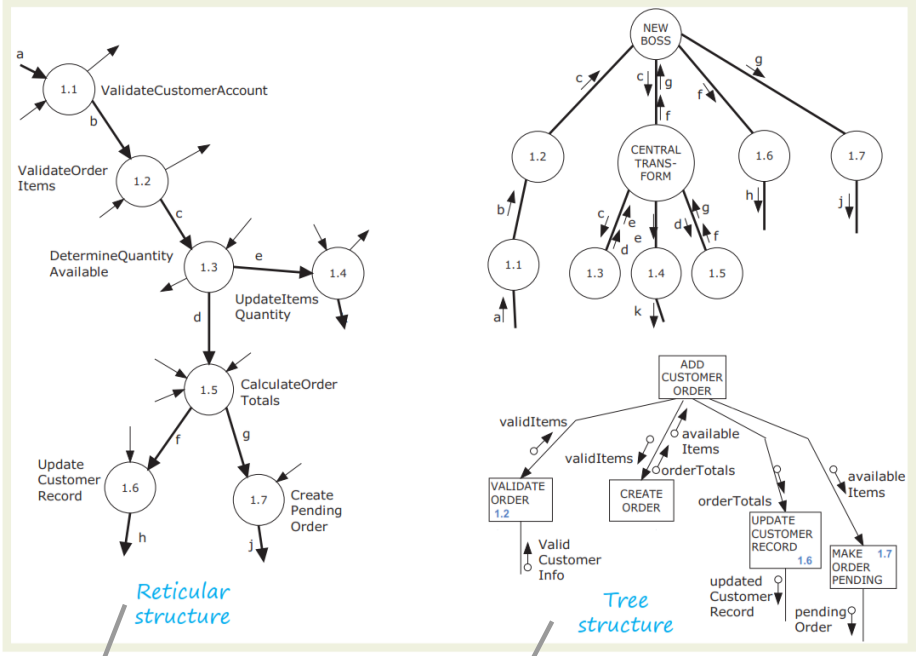
=> diagrammi delle classi : vedere cosa c'è dentro i casi d'uso

AB
Andrea Baruzzo
Publishing

The need of a unified approach from analysis to design

descrizione della struttura del sistema.

- In the Structured Method the analysis models and their counterpart of design models are based on different, incompatible notations
- This creates the impedance mismatch problem (§3.1)



Analysis model

Design model

The need of a unified approach from analysis to design

il modello delle classi descrive un insieme di concetti e relazioni tra concetti mediante un grafo : occhio alle dipendenze cicliche!

- In the Object-Oriented Method, both analysis and design models are based on the same, common structure: the class diagram

- Transitioning from analysis to design implies a refinement of models
- Design models introduce more details such as new classes, methods, and attributes, but the fundamental reticular structure remains stable

Objects vs. classes

- An object is an instance of a class (§3.2)
- An object is an abstraction of a system with an internal state, an identity, and an observable behavior
  ↳ attributi ↳ metodi
- A class is a template for the definition of all the characteristics of an object such as attributes and methods
- A class defines a set of objects that share the same common structure and the same behavior

Attributes vs. methods

- An attribute describes an information related to the internal state of an object
- Every object holds a proper value for each attribute defined in the corresponding class
- A method defines the behavior of the object for a specific event (the corresponding message sent to the object at some time)
- The behavior of the object can be either state-dependent or state independent at the moment of the message reception (Bank account, Example §3.3)

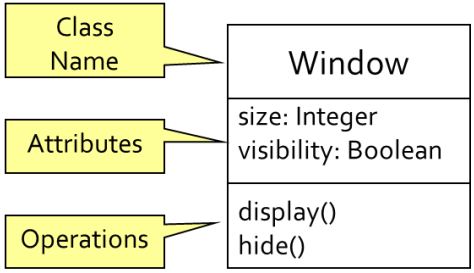
What is a class

- A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them
- A central modeling technique that is based on object-oriented principles
- The richest notation in UML

- Classes (obviously!)
- Attributes and Operations
- Relationships
  - Associations
  - Generalization
  - Dependency
  - Realization
- Constraint Rules and Notes

The class graphical notation (§3.4)

- A class is the description of a set of objects having similar attributes, operations, relationships and behavior
- A class usually describes an entity of the problem/solution space

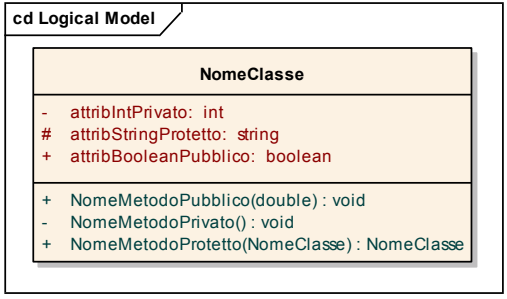


Visibility:

+ Public (default for methods)

- Private (default for attributes)

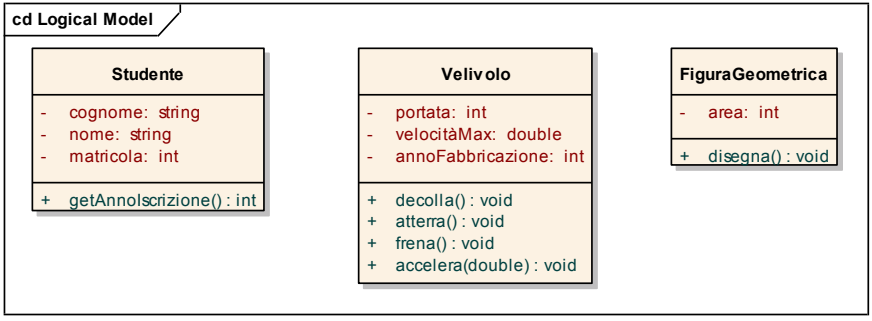
Protected (as public for derived classes, as private for all the others)



ELISIONE, che viene usata in UML, è la soppressione di dettagli non ritenuti utili

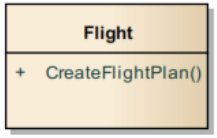
Examples of classes in UML

- Names conventions
  - Class: initial Upper-case for each single word (Camel-case)
  - Attributes: initial lower-case, initial Upper-case for each single word
  - Methods
    - (C++) initial Upper-case for each word
    - (Java) the same as for attributes

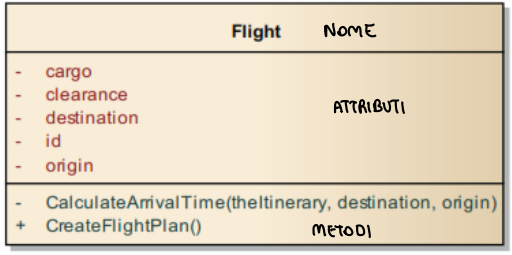


Many perspectives possible in a class model

Software modeling with UML – Classes



(a) Classe descritta dalla prospettiva concettuale



(b) Classe descritta secondo la prospettiva di specifica

(b) Classe descritta secondo la prospettiva di implementazione

- Three main perspectives (PROSPETTIVA) (§3.4.1):
  1. Conceptual (a)
  2. Specification (b)
  3. Implementation (c)

livello di dettaglio da scegliere in base allo scopo che abbiamo

e' INCAPSULAMENTO rinforza l'INTEGRITÀ dello STATO

e' INFORMATION HIDING serve per non far verificare l'effetto domino

↳ principio TEORICO su cui si basa l'incapsulamento

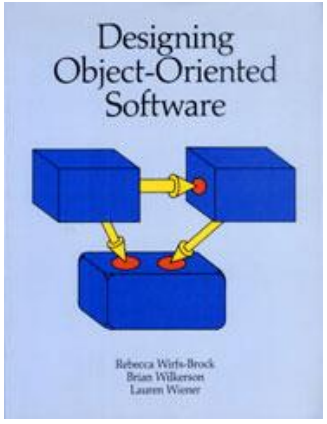
"nascondo i dettagli implementativi di una classe/oggetto all'utente"

PRINCIPIO di ELISIONE: nascondo tutto ciò che non è necessario per capire le diagramma, es. i tipi delle funzioni

Classes and responsibilities (§3.3.4)

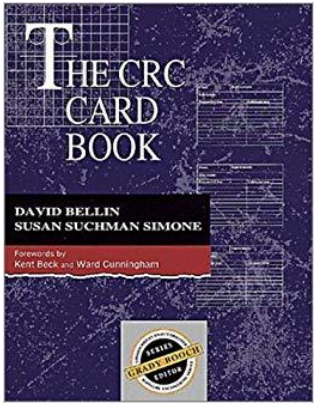
all'esame non fa domande su questo

- A class is a representation of a concept: it evokes a set of responsibilities about:
  - Information held (state)
  - Behavior (operations)
- A responsibility is an expression of what the class knows and what the class does (preserving the conceptual association with the concept)
- The name of the class (attributes, methods) is fundamental (Rebecca Wirfs-Brock)
- Avoid names that reveal implementation details



The CRC card technique (§3.5)

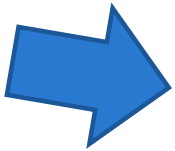
- CRC Card = Class Responsibility Collaborator card
- Invented in 1989 by Kent Back and Ward Cunningham
- A method for finding good class abstractions
  - Read specification
  - Work through requirements, highlighting nouns and noun phrases to give candidate classes
  - Work through candidates, deciding likely classes and rejecting unlikely (phantoms, out of scope, not this time, border-line, …)

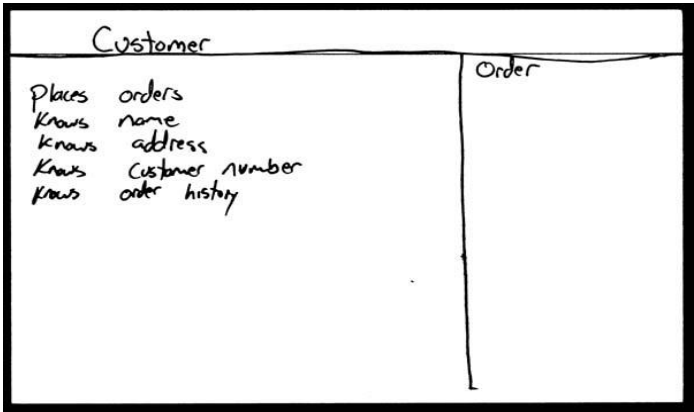


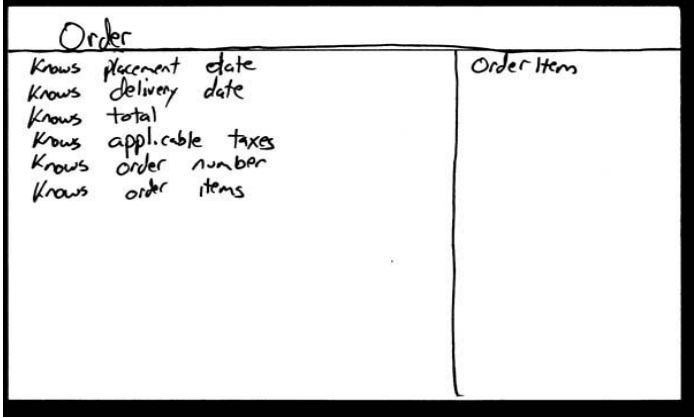
CRC card: A Index-Card template and examples

Software modeling with UML – Classes

Class Name	
Responsibilities	Collaborators



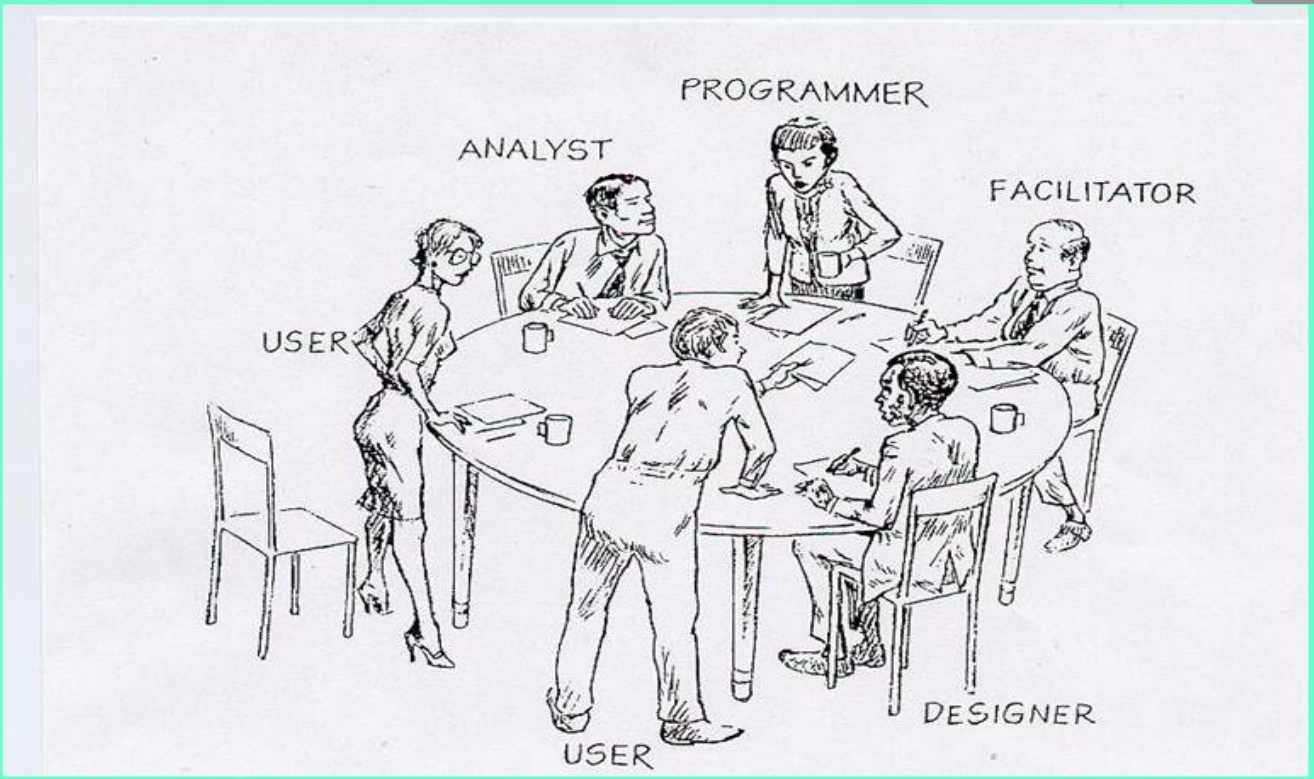




Student	
Student number Name Address Phone number Enroll in a seminar Drop a seminar Request transcripts	Seminar

The CRC card technique is a collaborative method

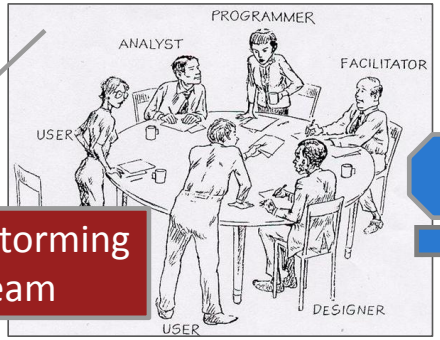
The CRC Card Team



Source: The CRC Card Book by Bellin et.al (1997)

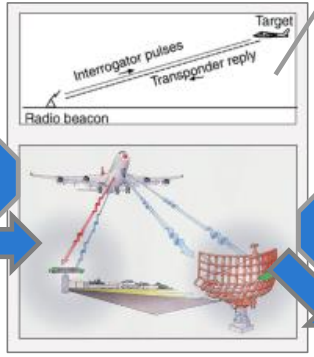
The core class selection process

Software modeling with UML – Classes



Brainstorming team

1



Problem analysis

2

Candidate classes generation

Flight (volo)
Radar (radar)
Alarm (allarme)
Aircraft (velivolo)
Runway (pista)
Trajectory (traiettoria)
Airspace (spazio aereo)
Target (obiettivo)
3DRadar (radar 3D)
Alarms (allarmi)
Telemetry (telemetria)
RadarData (dati radar)
AirborneRadar (radar aereo)
OmniAntenna (antenna Omni)
RadarMode (modalità radar)
RadarLink (collegamento)
Modem (modem)
AircraftRoute (rotta aerea)
Payload (carico)
FlightRoute (rotta volo)
Airport (aeroporto)
FlightPlan (piano di volo)
MainAntenna (antenna principale)
RadarController (controllore radar)
Transponder (transponder)
Digitalizer (convertitore digitale)
Surveillance (sorveglianza)
LookDown (modalità "Look-down")
ATCGroundStation (stazione di controllo)
ATCRadarBeaconSystem (sistema radar Beacon)
GroundClutter (disturbi d'identificazione)
PrimarySurveillanceRadar (radar primario)
SecondSurveillanceRadar (radar secondario)
TransponderCode (codice d'identificazione)
CollisionAvoidanceSystem (sistema anti-collisione)
IFFSystem (sistema d'identificazione volo amico)
MicrowaveLink (trasmissione radio a microonde)
CoaxialLink (trasmissione coassiale)
RadarDataProcessor (processore dati radar)
AltitudeEncoder (dispositivo codifica altitudine)
AirTrafficControlCenter (centro di controllo aereo)
ControlTower (torre di controllo)

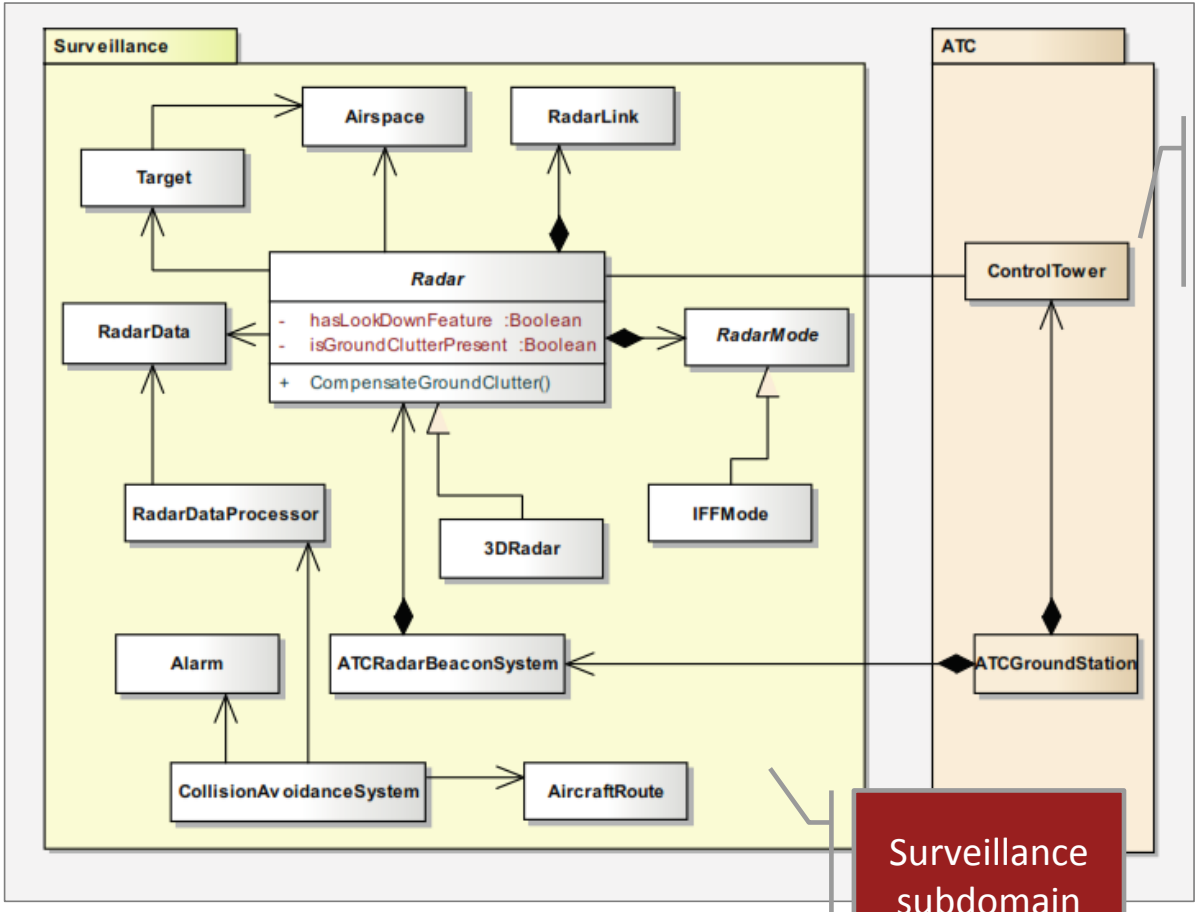
3

Classi Essenziali	Classi Incerte	Classi Irrilevanti
Radar	*MainAntenna*	~~*Aircraft*~~
Alarm	*OmniAntenna*	~~*Runway*~~
Airspace	*Flight*	~~*Trajectory*~~
3DRadar	*Digitalizer*	~~*Alarms*~~
RadarData	*ModemLink*	~~*Telemetry*~~
RadarMode	*FlightRoute*	~~*AirborneRadar*~~
RadarLink	*FlightPlan*	~~*Payload*~~
AircraftRoute	*Surveillance*	~~*RadarController*~~
ATCRadarBeaconSystem	*PrimarySurveillanceRadar*	~~*Transporter*~~
CollisionAvoidanceSystem	*SecondSurveillanceRadar*	~~*LookDown*~~
IFFMode	*MicrowaveLink*	~~*AirTrafficControlCenter*~~
RadarDataProcessor	*CoaxialLink*	~~*GroundClutter*~~
ControlTower		~~*TransponderCode*~~
ATCGroundStation		~~*AltitudeEncoder*~~
Target		~~*Airport*~~

Core classes selection

The synthesis of a conceptual domain model

Software modeling with UML – Classes



Relations: association (§3.3.3, §3.4.4)

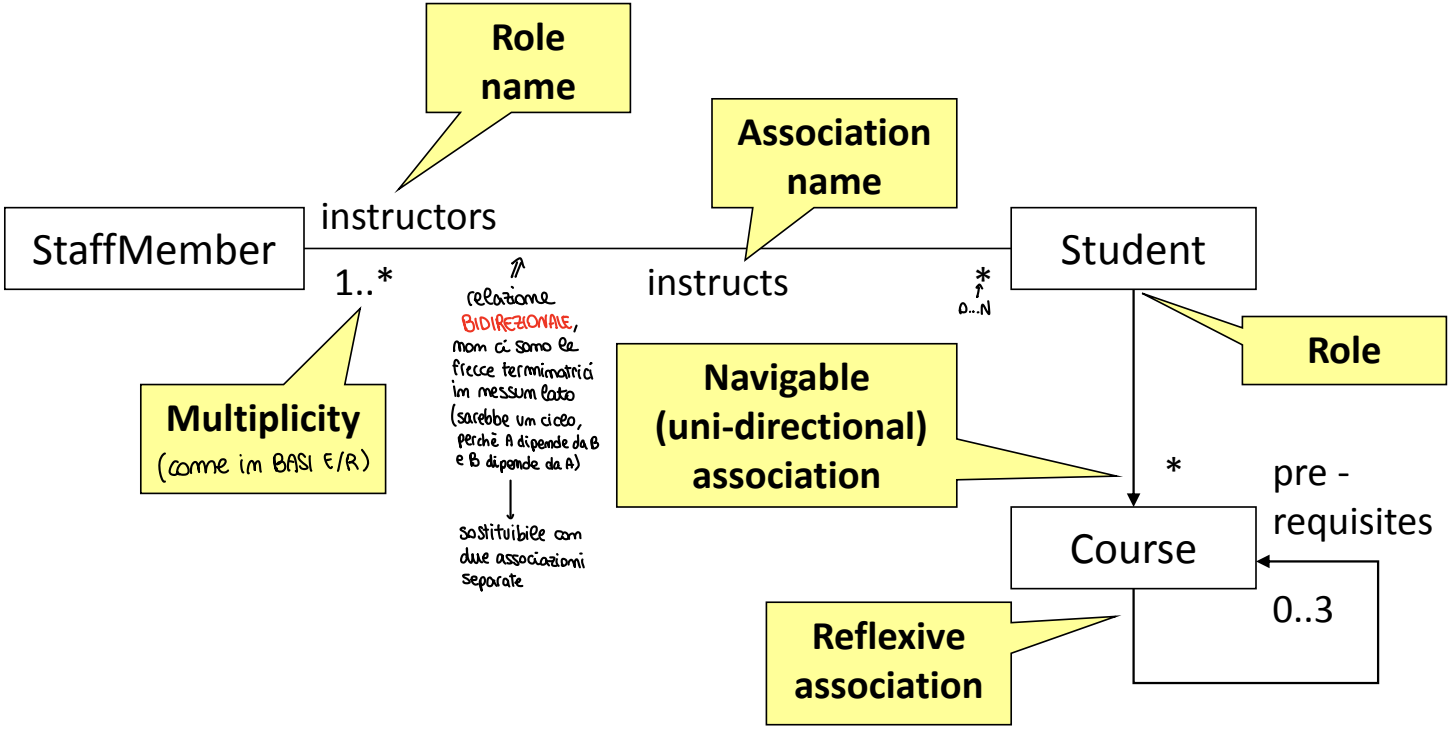
COME SONO FATTE LE RELAZIONI ?
↳ come parlano tra di loro i moduli sw ? determinato dalle relazioni

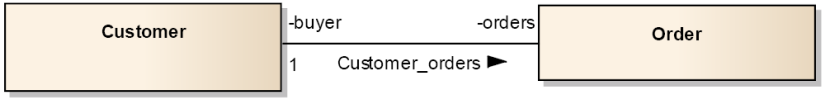
- A "semantic" relationship between two or more classes that specifies connections among their instances
- A structural relationship, specifying that objects of one class are connected to objects of a second (possibly the same) class

↳ es. ASSOCIAZIONE, io conosco i valori interni dell'oggetto guardando gli attributi

- Mono- or bi-directional relation
  - Source (origin) and target (destination) classes
  - End-points can be adorned with labels and numbers
    - Example: "An Employee works in a department of a Company"

Relations: association (cont'd)





Relations: association (cont'd)

- An association between two classes specifies that the objects of one end-point "know" the objects at the other end-point...
- ... and are capable to send them messages!
- Association name: (optional) Label placed in the middle of the relation
  - Represents a verb providing the implicit action of the relation
- Role name: (optional) Label placed at end-points
  - Specifies the role of the end-point in the whole association context
  - Usually it is a name (an attribute if we take an implementation view)
  - Mandatory only for reflexive relations

Relations: association (cont'd)

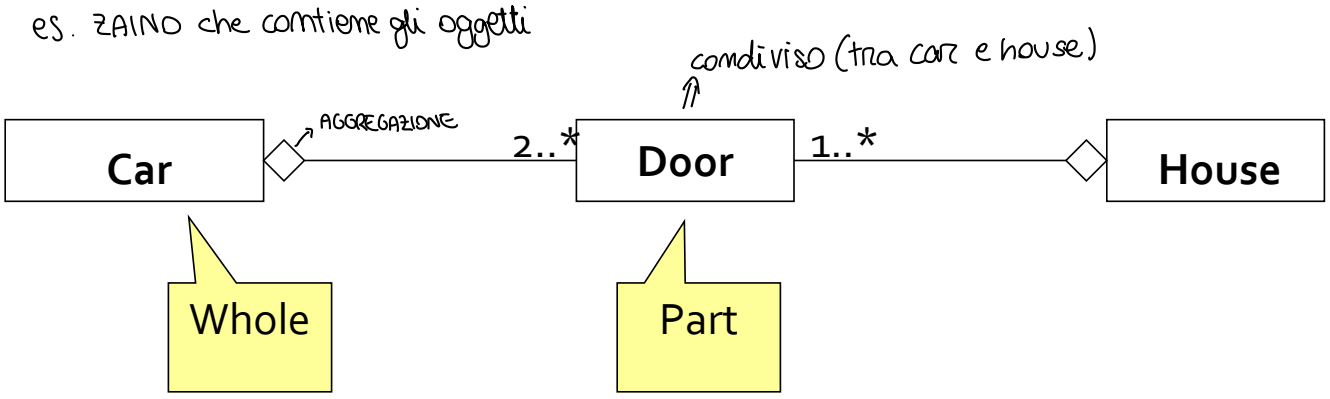
- Multiplicity
  - The number of instances of the class, next to which the multiplicity expression appears, that are referenced by a *single* instance of the class that is at the other end of the association path
  - Indicates whether or not an association is *mandatory* (OBBLIGATORIA)
  - Provides a *lower* and *upper bounds* on the number of instances

Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

Relations: aggregation (§3.4.4)

AGGREGAZIONE

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - Models a "is a part/part of" and "holds/contains" relationships
  - Examples: car-door; house-door; hangar-airplane



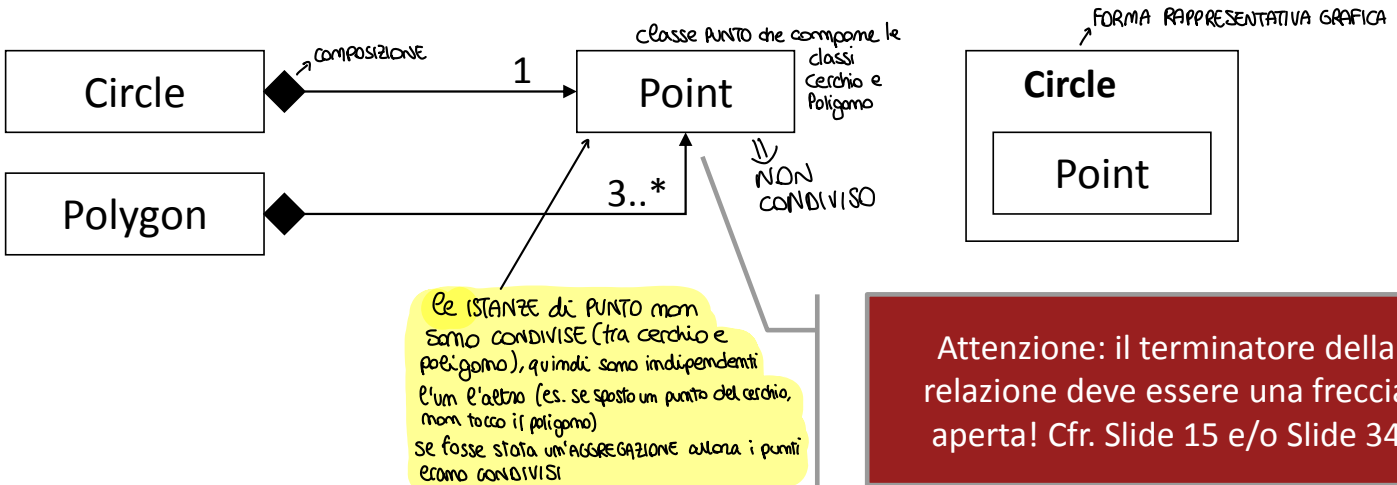
Relations: aggregation

- Not always simple to recognize (in that cases use association)
- Aggregation tests:
  - Is the phrase "part of" used to describe the relationship?
    - *A door is "part of" a car*
  - Are some operations on the whole automatically applied to its parts?
    - *Move the car, move the door*
  - Are some attribute values propagated from the whole to all or some of its parts?
    - *The car is blue, therefore the door is blue*
  - Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
    - *A door is part of a car; A car is not part of a door*

Relations: composition (§3.4.4)

COMPOSIZIONE

- A strong form of containment (stronger than aggregation)
  - The whole is the sole owner of its part
    - *The part object may belong to only one whole*
  - Multiplicity on the whole side must be zero or one
  - The life time of the part is dependent upon the whole
    - *The composite must manage the creation and destruction of its parts*



Attenzione: il terminatore della relazione deve essere una freccia aperta! Cfr. Slide 15 e/o Slide 34

Relations: aggregation vs. composition

- Quite often, "part of" ("is composed of/by") is best suited for composition, whereas aggregations are best described by "contains", "holds", "has a"
- Not always simple to discriminate → difficile molto spesso distinguere - composizione - aggregazione
- A car "is composed" of doors or it "contains" doors?

Don't spend a lot of time struggling with these details...

- Aggregation and composition describe forms of containment: recognize it and distinguish it from other types of relations!

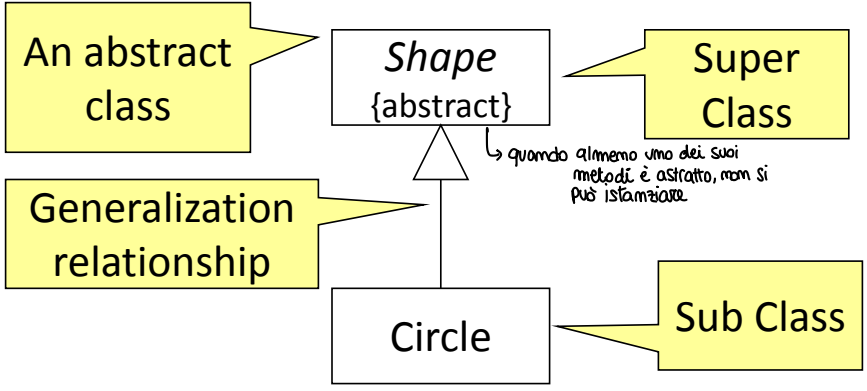
Relations: generalization (§3.3)

GENERALIZZAZIONE (EREDITARIETÀ) equivalente del costrutto EXTEND → un elemento è specializzazione di un altro componente generale. Potrebbe essere sostituito

- Indicates that objects of the specialized class (subclass) are substitutable for objects of the generalized class (super-class)
  - "is kind of" relationship

figlio eredita dal padre → un figlio può avere solo 1 padre

{abstract} is a tagged value that indicates that the class is abstract. The name of an abstract class should be italicized



Relations: generalization (cont'd)

- A sub-class inherits (EREDITA) from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship may not be used to model interface implementation

Relations: realization (§3.4.2)

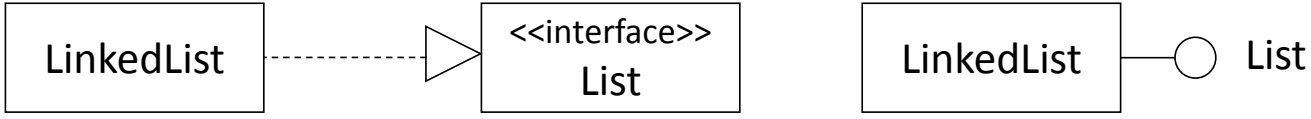
REALIZZAZIONE (IMPLEMENTAZIONE) → in una relaz. di REALIZZAZIONE, un'entità indica una responsabilità che NON È IMPLEMENTATA da sola e l'altra entità che la implementa.

Oltre al costrutto "EXTENDS", c'è anche il costrutto "IMPLEMENTS" → interfaccia (classe base che non ha stato ed ha tutti i metodi astratti)

- A realization relationship indicates that one class implements a behavior specified by some interface
- An interface can be realized by many classes
- A class may realize many interfaces

  una classe può implementare più interfacce, un figlio può estendere solo un padre

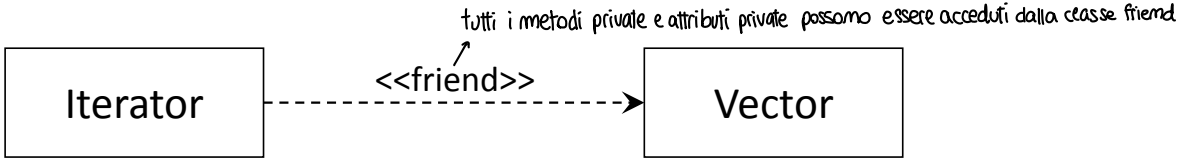
  – Interface refers the concept of "set of visible behaviors"; not confuse this concept with the interface construct (Java)



Relations: dependency

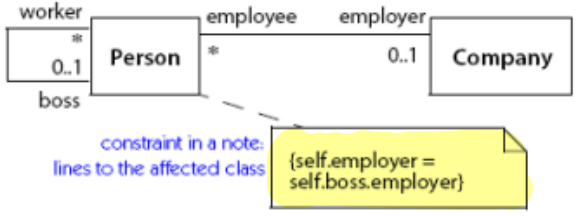
DIPENDENZA

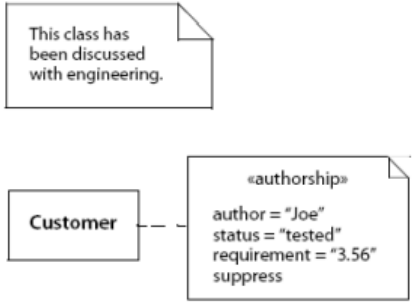
- A dependency indicates a relation between two classes although there is no explicit association between them
- A stereotype may be used to denote the type of the dependency
- It is the weakest form of coupling
  - Arguments or return type of a method;
  - arguments of a constructor;
  - new instantiation inside a method body



Constraints rules and notes (§3.4.5)

- VINCOLI
  Constraints and notes annotate among other things associations, attributes, operations and classes.
- Constraints are semantic restrictions noted as Boolean expressions





Constraints are used for...

I VINCOLI SONO USATI PER:

- Document assumptions (concerning analysis, design or implementation aspects)
- Describe invariants
- Design by contract :
  - Invariant : is always true for an object
  - Pre-condition : is true when method is called
  - Post-condition : is true after method is called

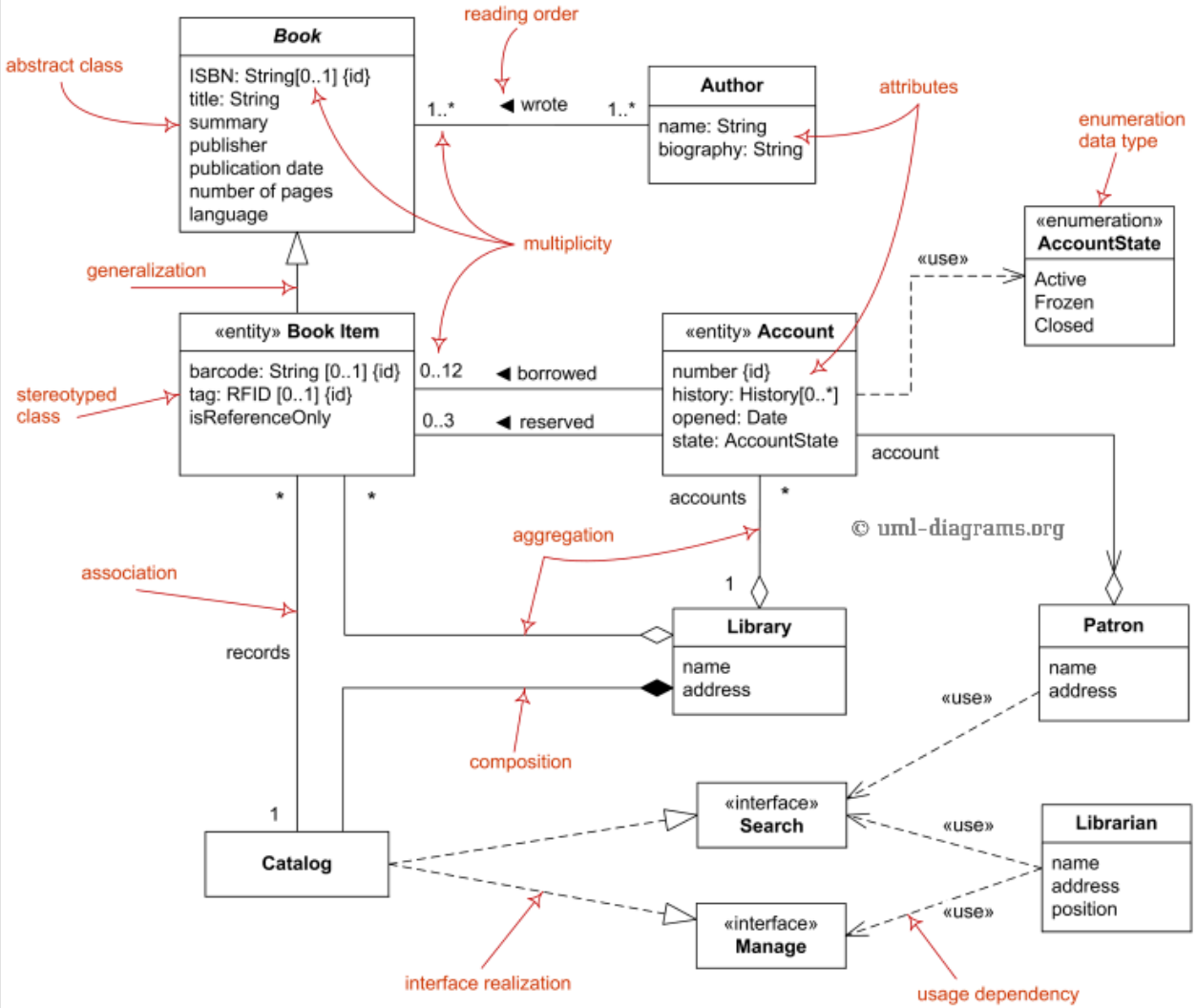
Stereotypes

STEREOTIPI

- The simplest extension mechanism in UML
- Used to provide a specialized (or custom) semantics to an existing symbol present in the notation (and to a new symbol also)
  - Extend the UML using the << ... >> notation
  - e.g. Classes can be
  - <<Interface objects>>
  - <<Control objects>>
  - <<Entity objects>>
  - Patterns
    e.g. <<singleton>>

A REVIEW OF THE CLASS NOTATION

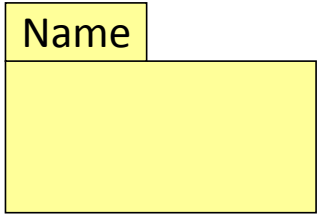
Software modeling with UML – Classes



AB
Andrea Baruzzo Publishing

Packages (§3.4.3)

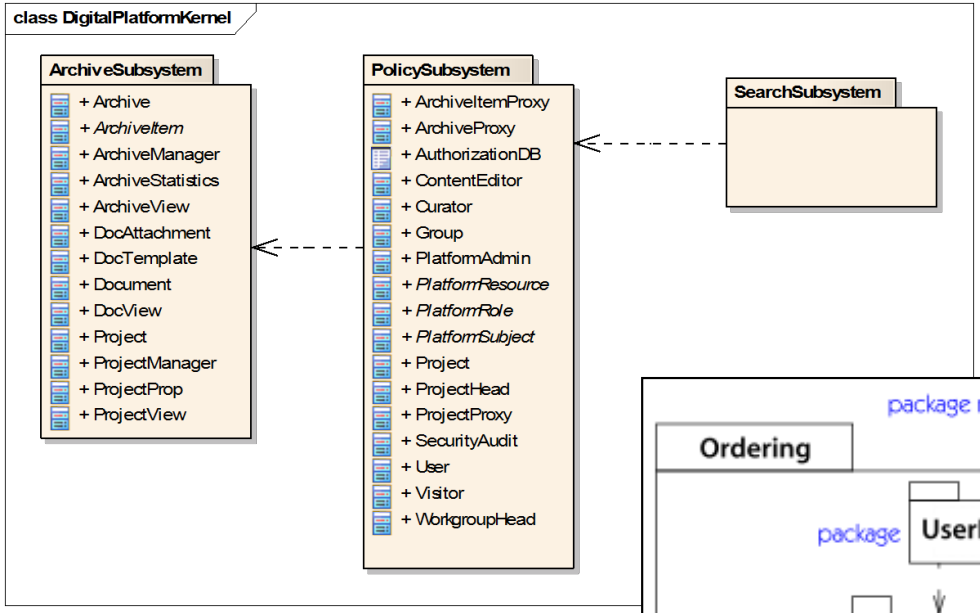
- A package is a general purpose grouping mechanism

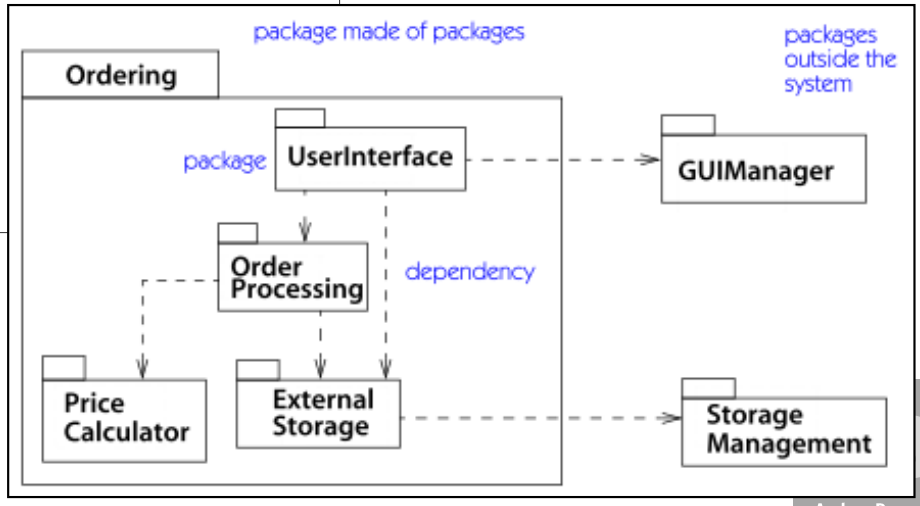


- Commonly used for specifying the logical architecture of the system
- A package does not necessarily translate into a physical sub-system

Package diagrams

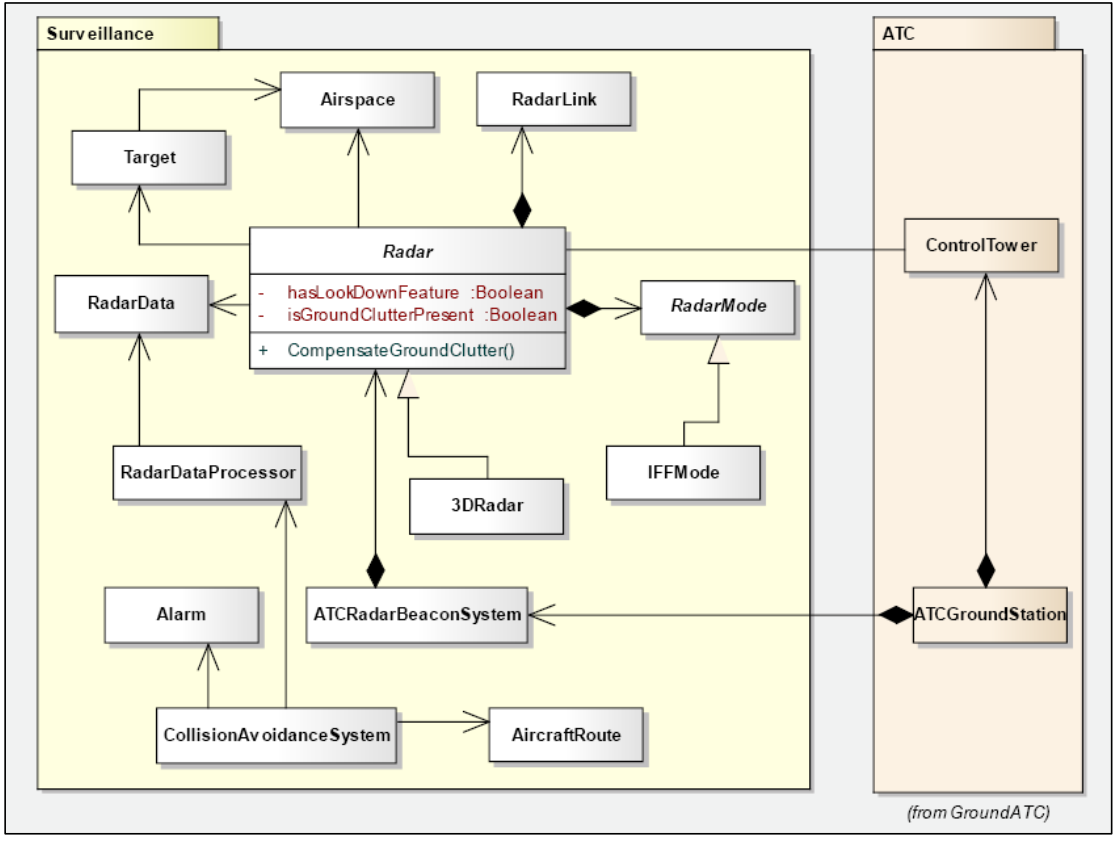
Software modeling with UML – Classes





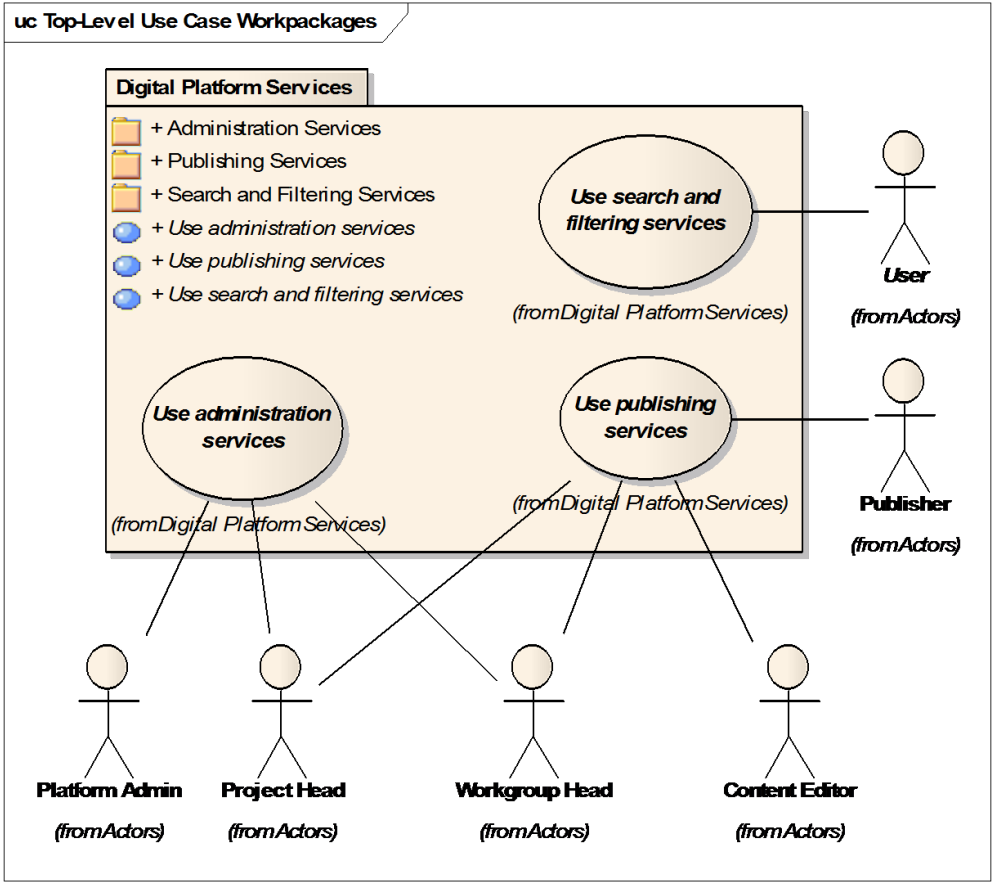
Internal structure of a package

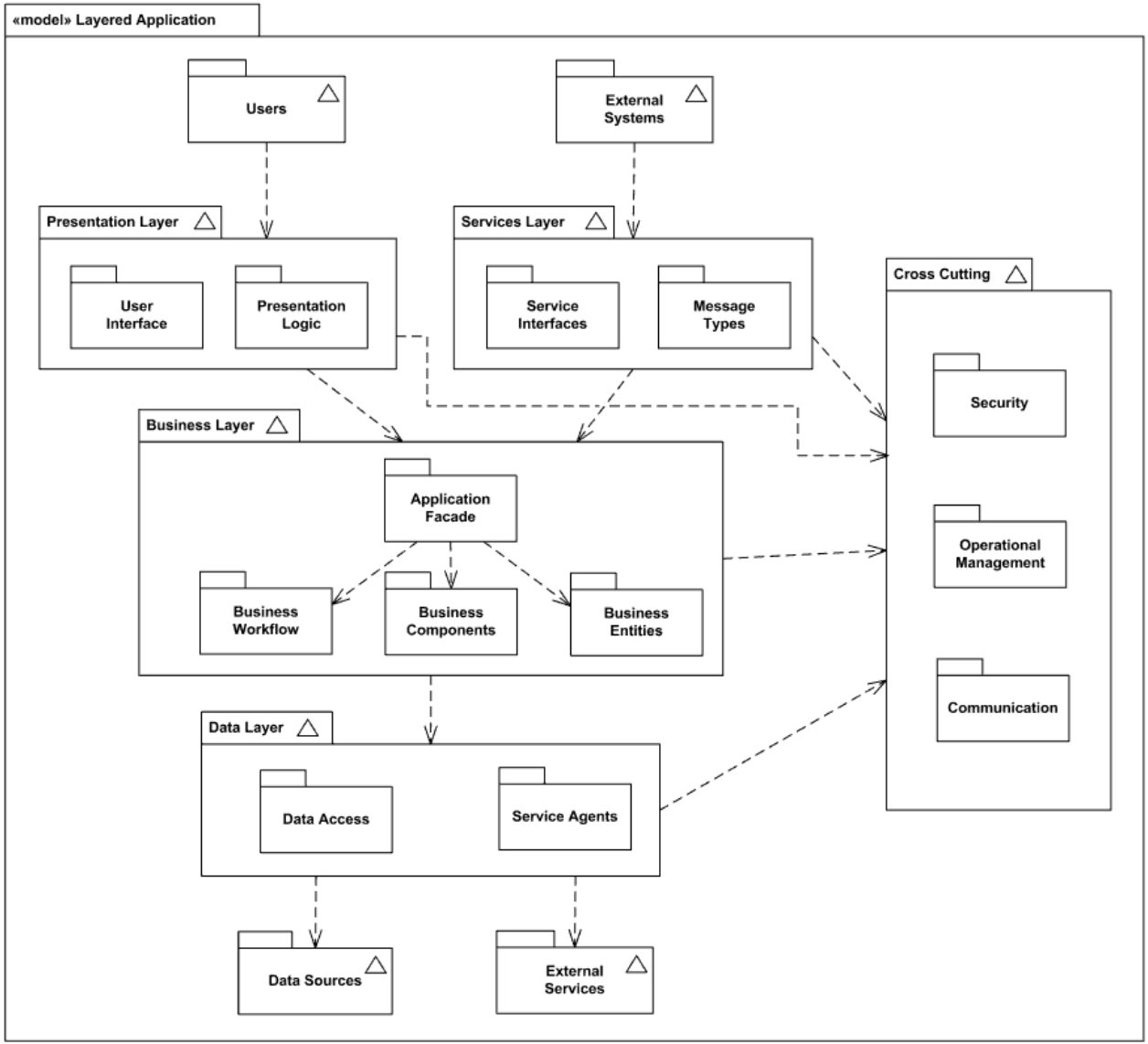
Software modeling with UML – Classes



Package in use case diagrams

Software modeling with UML – Classes

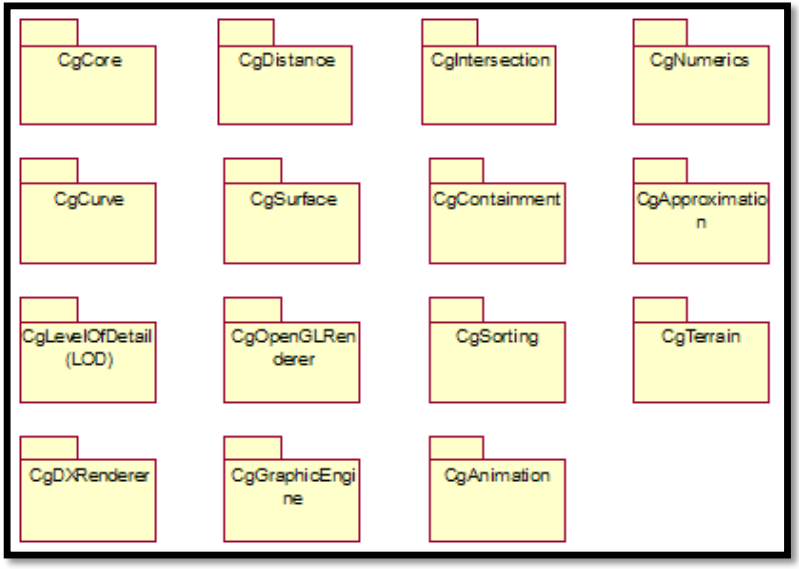




A non-informative package diagram

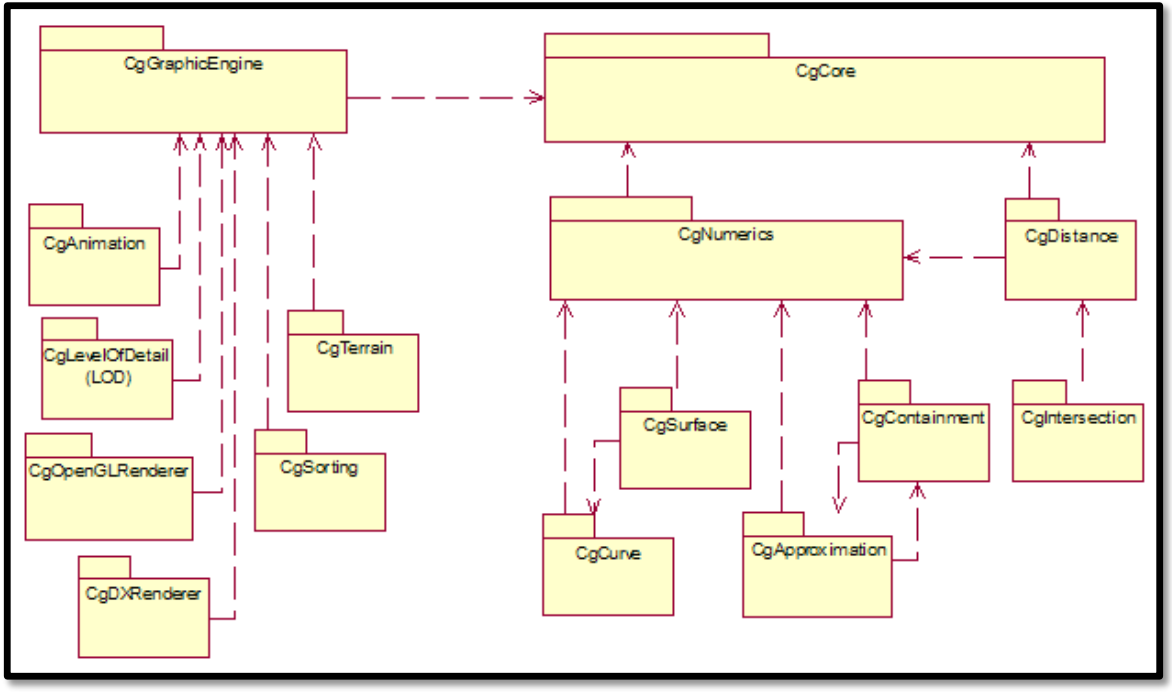
Tombstone packages

- No relations
- No abstraction levels
- No details about the inside

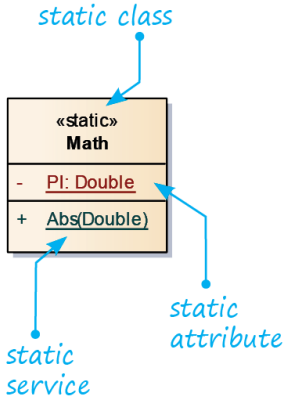


A MORE INFORMATIVE PACKAGE DIAGRAM

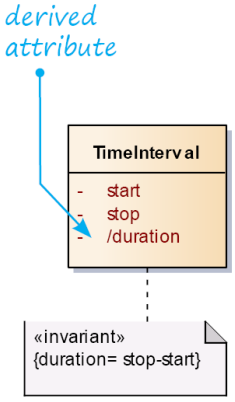
Software modeling with UML – Classes



Static, derived, read-only, and frozen (§5.1)




```
1  double constantValue= Math.PI;
2  double x= -3.14;
3  double positiveValue= Math.Abs(x);
```

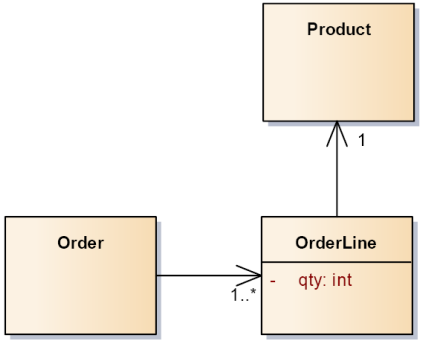


Vincoli:

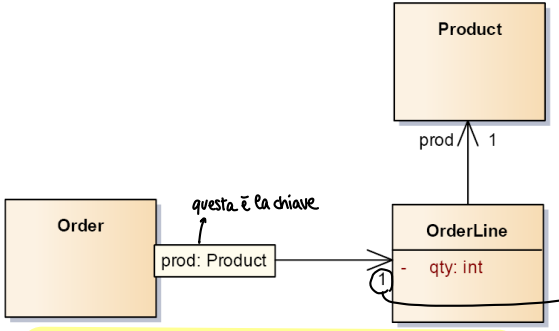
- `{readOnly}` è un vincolo: precisa che il valore dell'attributo a cui è riferito non può essere modificato dall'esterno
- `{frozen}` è un vincolo: precisa che il valore dell'attributo a cui è riferito non può cambiare per tutta la vita di un oggetto

Qualified associations, associations class (§5.1.3)

ASSOCIAZIONE QUALIFICATA: meccanismo di recupero CHIAVE-VALORE

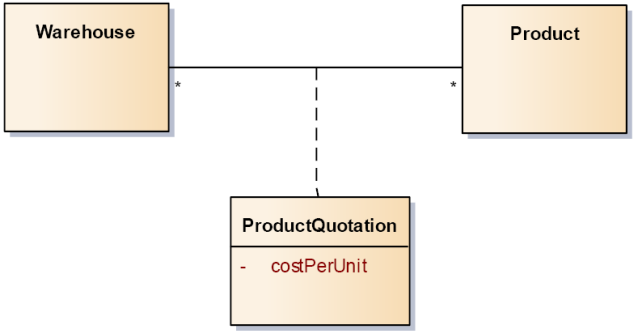


(a) *In un modello con una catena di associazioni semplici non è possibile garantire che per ogni istanza di* `Product` *ci sia una sola istanza di* `OrderLine` *associata*

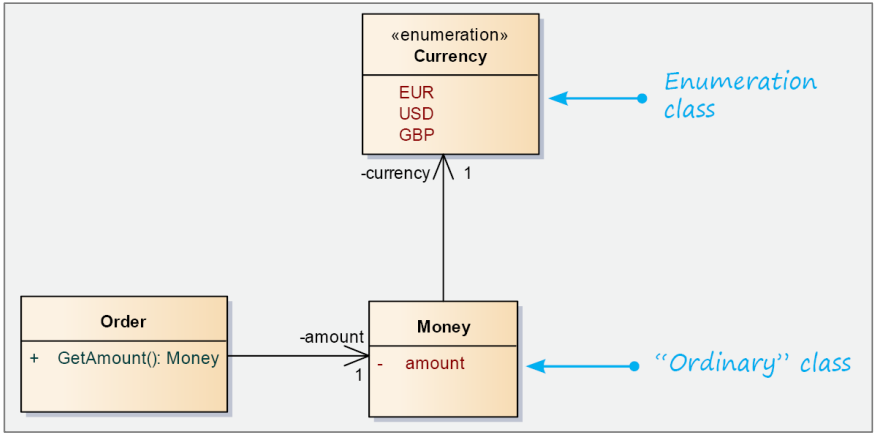


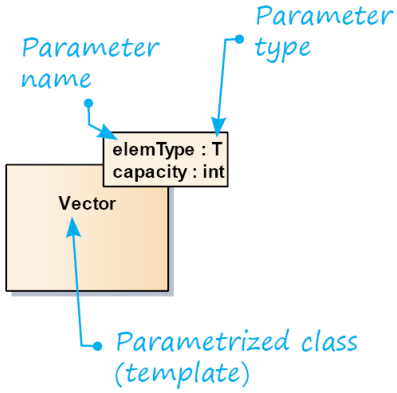
(b) *L'uso di un'associazione qualificata permette di esprimere un vincolo sulle istanze di* `OrderLine` *messe in corrispondenza di un oggetto* `Product` *all'interno di ciascun oggetto* `Order`

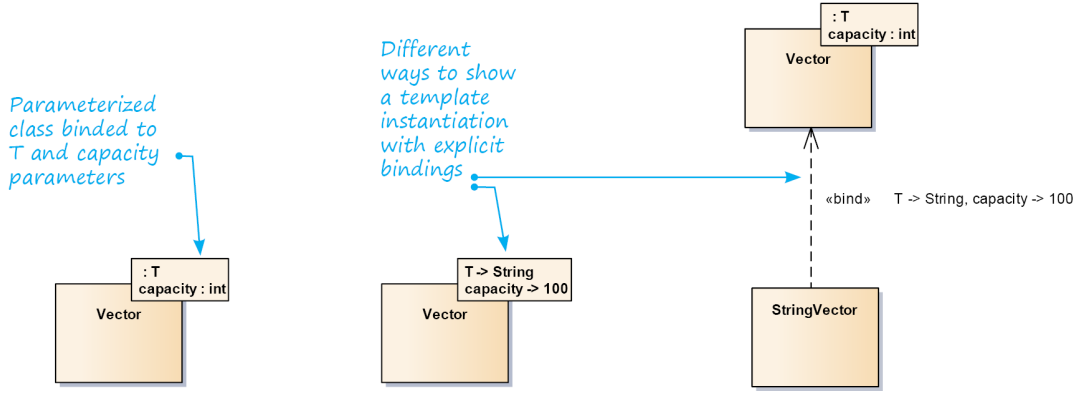
questa ora diventa la "molteplicità di OrderLine fissata la chiave di order"
↓
prima era 1...*



Enumerations and templates (§5.1.5, §5.1.6)







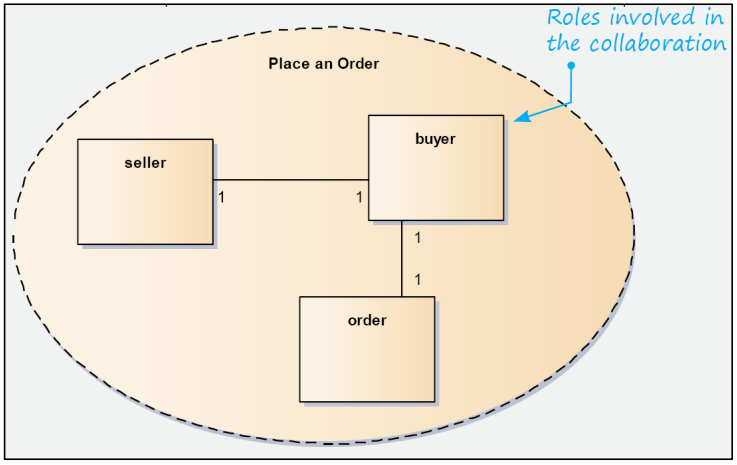
(a) Template generico Vector<T, capacity> con parametri T (tipo generico) e capacity (di tipo int)

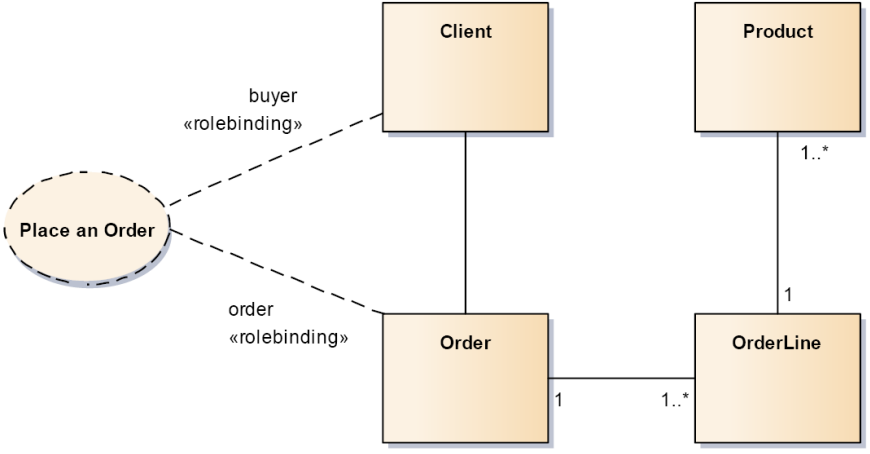
(b) Template generico Vector<T,capacity> istanziato nel caso in cui T sia legato al tipo String e capacity inizializzato col valore 100

(c) Template generico Vector<T,capacity> che esplicita la relazione di bind per i legami di T col tipo String e capacity col valore 100

Collaborations (§5.1.8)

Software modeling with UML – Classes





Bibliografia

Software modeling with UML – Classes

- P. Stevens, R. Pooley- *Usare UML: Ingegneria del Software con Oggetti e Componenti*. Addison-Wesley. 2008
- G. Booch, J. Rumbaugh, I. Jacobson- *The Unified Modeling Language User Guide 2/E*. Addison-Wesley. 2005
- M. Fowler – *UML Distilled: A Brief Guide to the Standard Object Modeling Language 3/E*. Addison Wesley. 2003
- L.V. Tagliati – *UML e Ingegneria del Software: dalla Teoria alla Pratica*. Tecniche Nuove. 2003
- [Baruzzo, 2017] Andrea Baruzzo. *Analisi e Progettazione di Sistemi Software Industriali*. CreateSpace. 2017