5. Diagramma delle classi

Paola Barra
a.a. 2023/2024

Diagrammi UML

- **Diagrammi dei casi d'uso**
  - Descrivono il comportamento funzionale del sistema come sono visti dagli utenti
- **Diagrammi delle classi**
  - Descrivono la struttura statica del sistema: oggetti, attributi, associazioni
- **Diagrammi delle sequenze**
  - Descrivono il comportamento dinamico tra gli oggetti del sistema
- **Diagrammi degli stati**
  - Descrivono il comportamento dinamico di un singolo oggetto
- **Diagrammi delle attività**
  - Descrivono il comportamento dinamico di un sistema, in particolare il flusso di lavoro

UML: convenzione di base

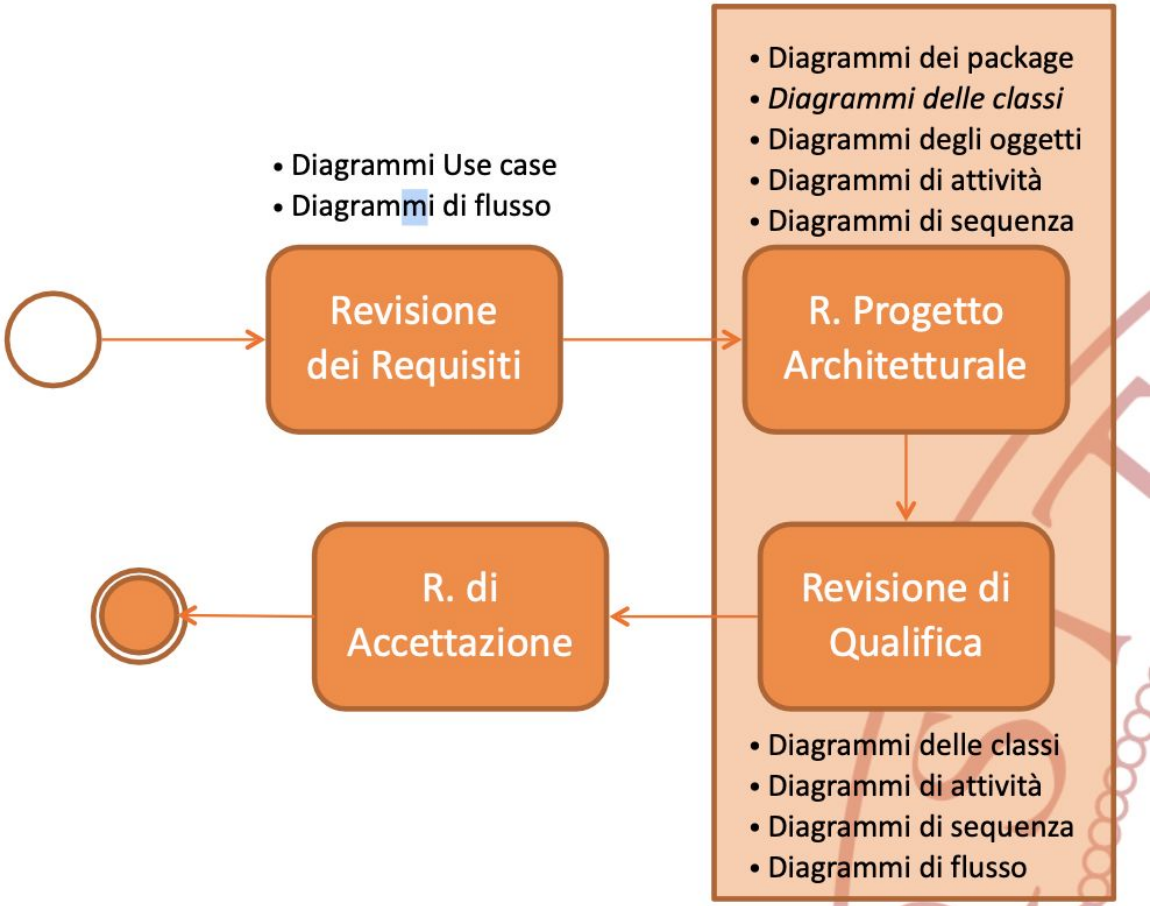
- Tutti i diagrammi UML denotano grafi di nodi e vertici
  - I nodi sono entità disegnati come rettangoli o ovali
  - I rettangoli denotano classi o istanze
  - Gli ovali denotano funzioni
- I nomi delle classi non sono sottolineati
  - SimpleWatch
  - Firefighter
- I nomi delle istanze sono sottolineati
  - myWatch:SimpleWatch
  - Joe:Fireighter
- Un arco tra due nodi indica una relazione tra le entità corrispondenti

Riepilogo della notazione UML base

- UML fornisce un'ampia varietà di notazioni per modellare molti aspetti dei sistemi software
- Ci siamo concentrati su poche notazioni
  - Modello funzionale: diagrammi dei casi d'uso
  - Modello ad oggetti: diagramma delle classi
  - Modello dinamico: diagrammi delle sequenze, degli stati

Diagramma delle classi

Specifica Tecnica, Definizione di Prodotto



Esempio

È richiesto lo sviluppo di un'applicazione che permetta la gestione di un semplice blog. In particolare devono essere disponibili almeno tutte le funzionalità base di un blog: deve essere possibile per un utente inserire un nuovo post e successivamente per gli altri utenti deve essere possibile commentarlo. Queste due operazioni devono essere disponibili unicamente agli utenti registrati all'interno del sistema. La registrazione avviene scegliendo una username e una password. La username deve essere univoca all'interno del sistema.

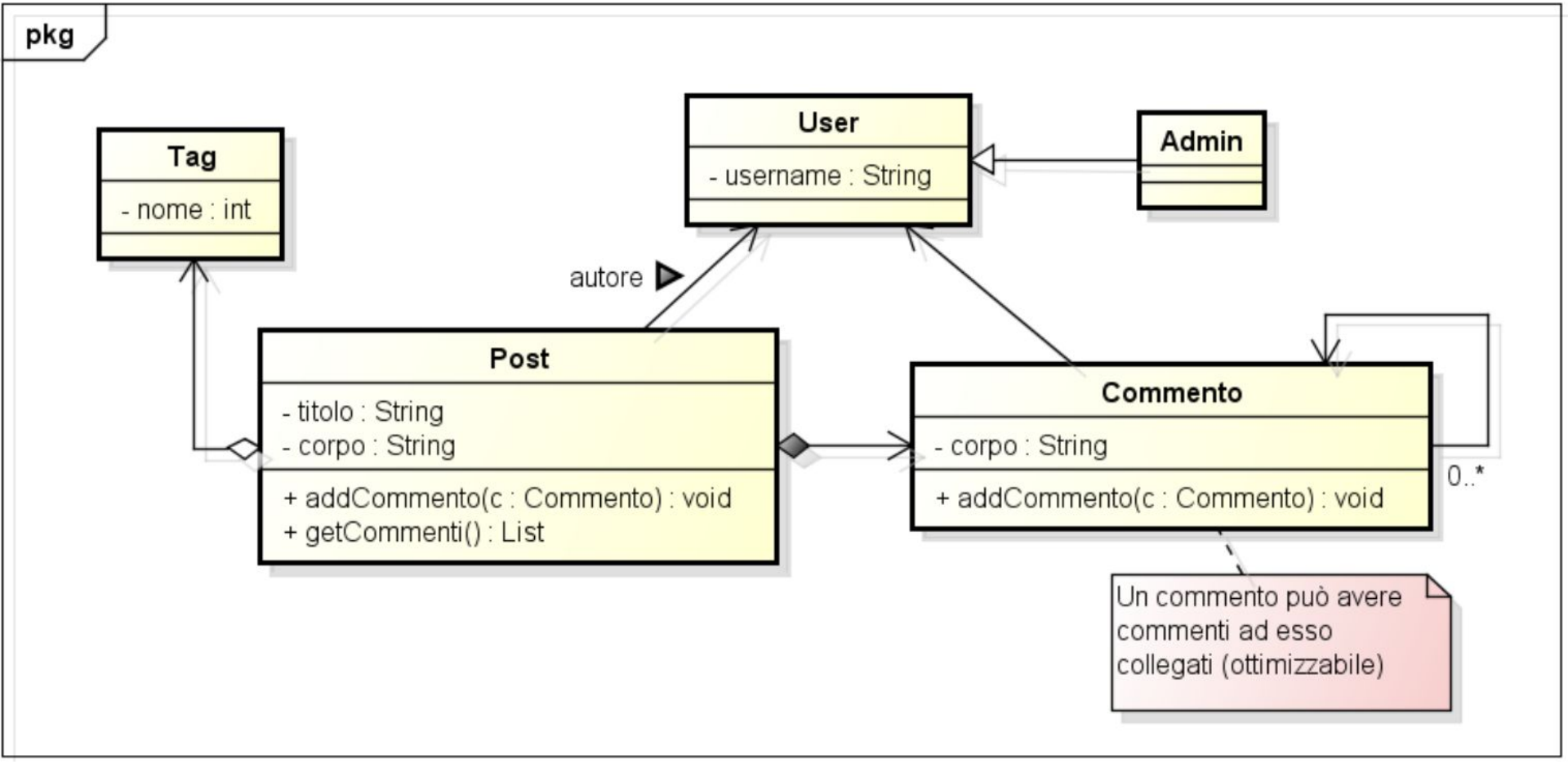
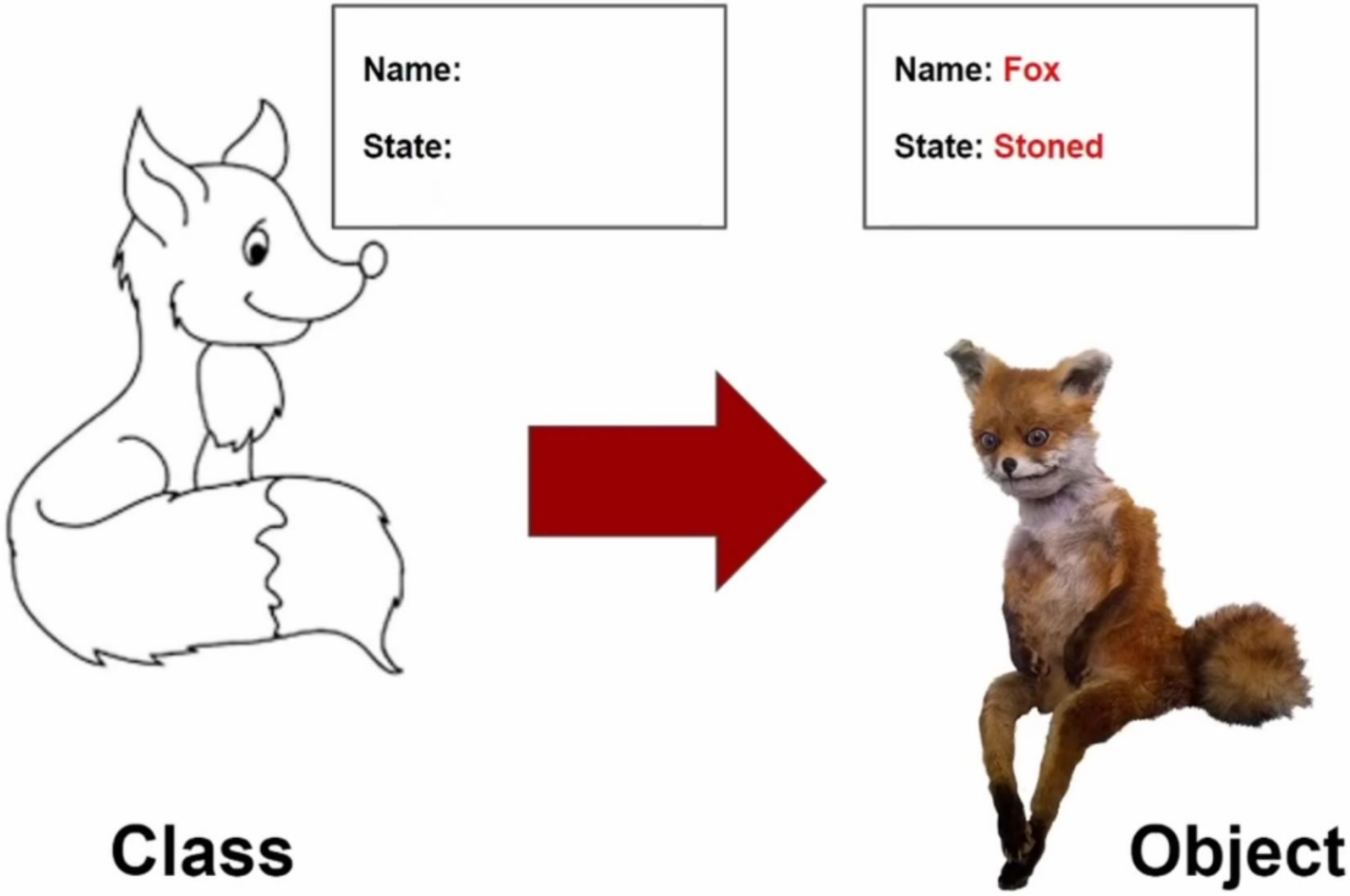
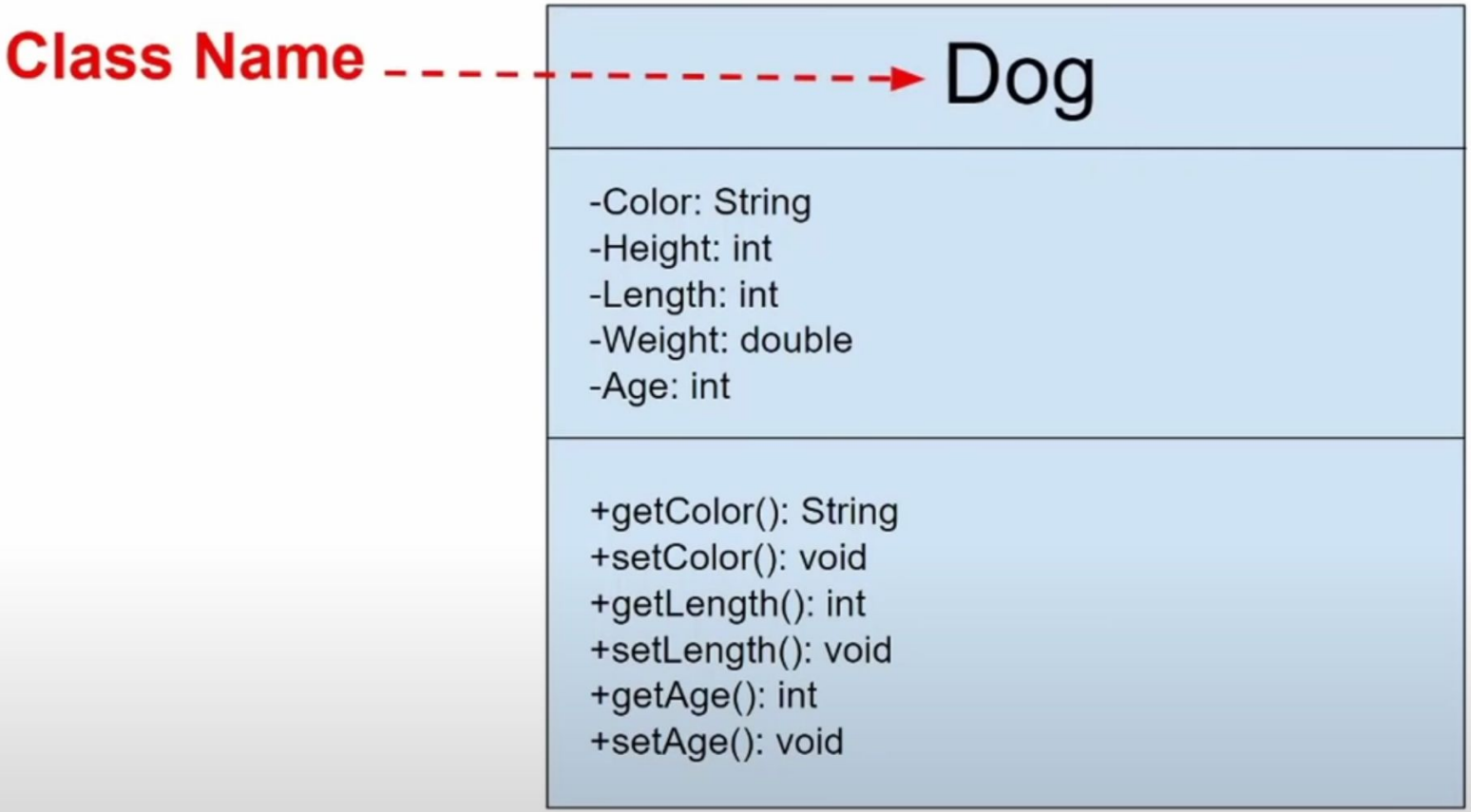

pkg
Tag
- nome : int
User
- username : String
Admin
autore
Post
- titolo : String
- corpo : String
+ addCommento(c : Commento) : void
+ getCommenti() : List
Commento
- corpo : String
+ addCommento(c : Commento) : void
0..*
Un commento può avere commenti ad esso collegati (ottimizzabile)

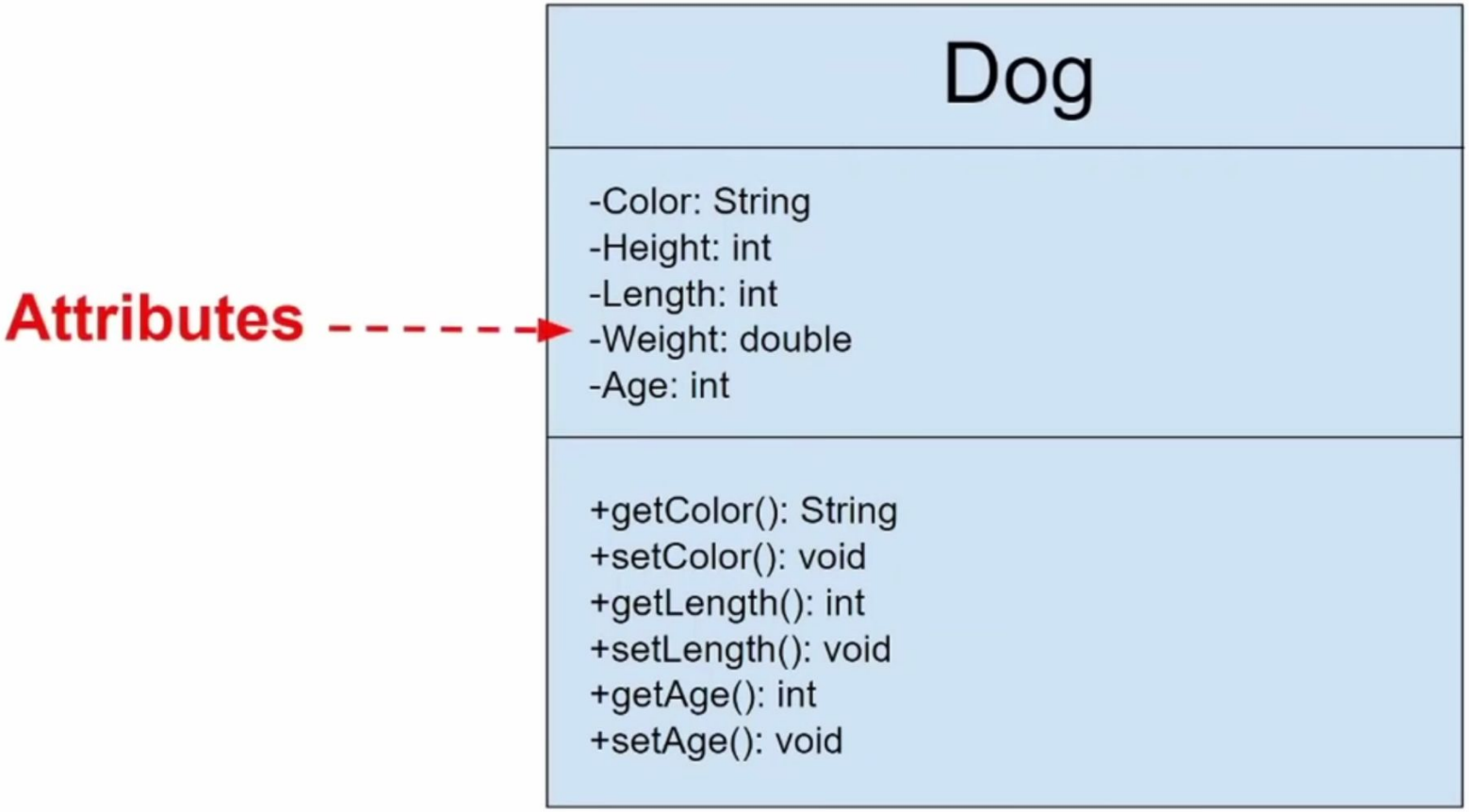
Diagramma delle classi

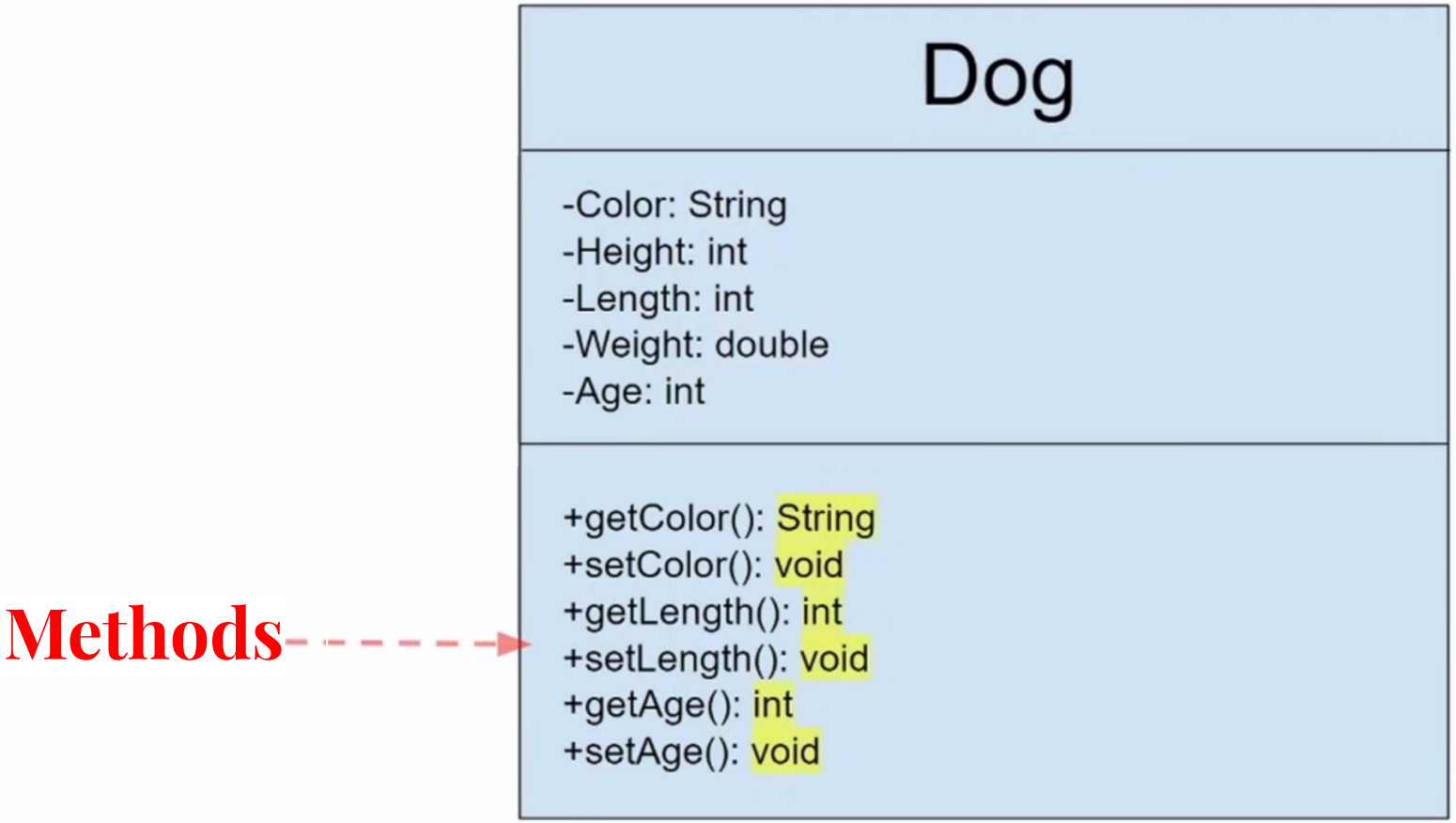
- Classi ed oggetti
  - I diagrammi delle classi descrivono la struttura del sistema in termini di classi ed oggetti
  - Le **classi** sono astrazioni che specificano gli attributi ed il comportamento di un insieme di oggetti
    - Una classe è una collezione di oggetti che condividono un insieme di attributi che contraddistinguono gli oggetti come membri della collezione
  - Gli **oggetti** sono entità che incapsulano lo stato ed il comportamento
    - Ogni oggetto ha un'identità mediante la quale ci si riferisce ad esso individualmente e che lo distingue dagli altri oggetti
  - In UML, classi ed oggetti sono rappresentati da riquadri composti da tre compartimenti
    - Parte alta: nome della classe o dell'oggetto
    - Centro: attributi
    - Parte bassa: operazioni
    - Le parti relative agli attributi e alle operazioni possono essere omesse per chiarezza



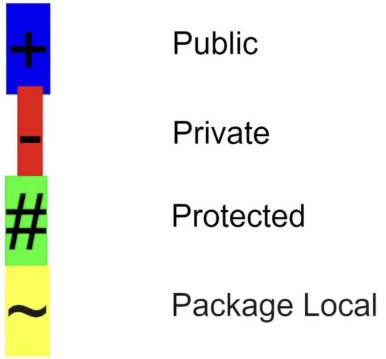
La classe è un template per creare l'oggetto in cui verranno definiti attributi e comportamento (i metodi)

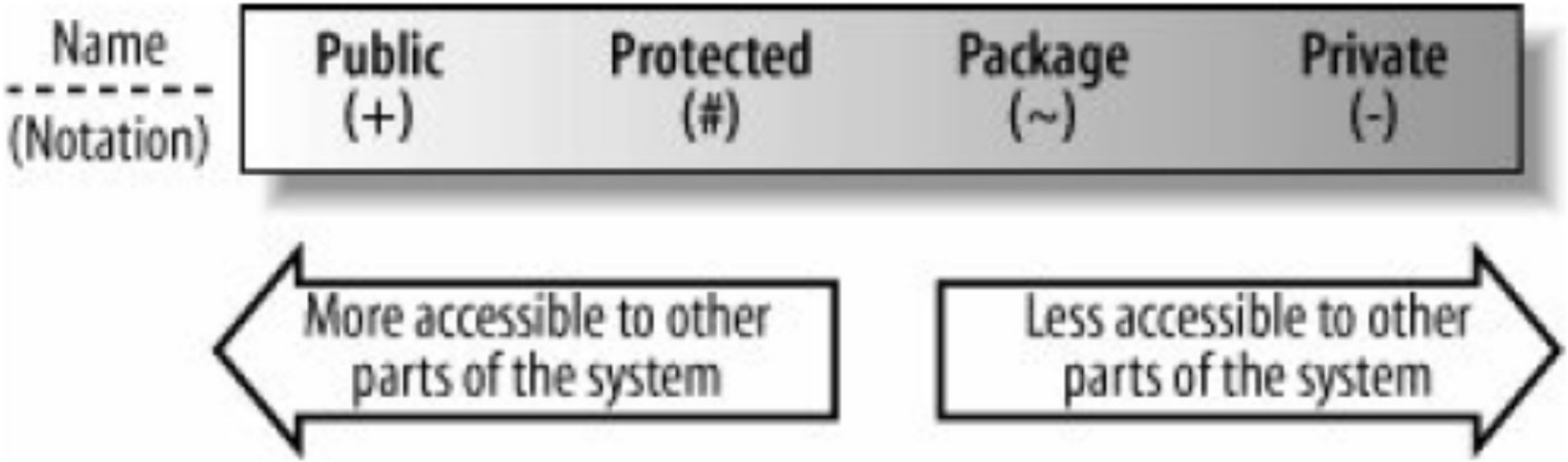

Class Name
Dog
-Color: String
-Height: int
-Length: int
-Weight: double
-Age: int
+getColor(): String
+setColor(): void
+getLength(): int
+setLength(): void
+getAge(): int
+setAge(): void


Attributes
Dog
-Color: String
-Height: int
-Length: int
-Weight: double
-Age: int
+getColor(): String
+setColor(): void
+getLength(): int
+setLength(): void
+getAge(): int
+setAge(): void


Dog
-Color: String
-Height: int
-Length: int
-Weight: double
-Age: int
+getColor(): String
+setColor(): void
+getLength(): int
+setLength(): void
+getAge(): int
+setAge(): void
Methods

- Visibilità


+
Public
-
Private
#
Protected
~
Package Local


Name
(Notation)
Public
(+)
Protected
(#)
Package
(~)
Private
(-)
More accessible to other parts of the system
Less accessible to other parts of the system

Come disegnare un diagramma di classe

Passo 1: Identificare i nomi delle classi

Il primo passo è quello di identificare gli oggetti primari del sistema.

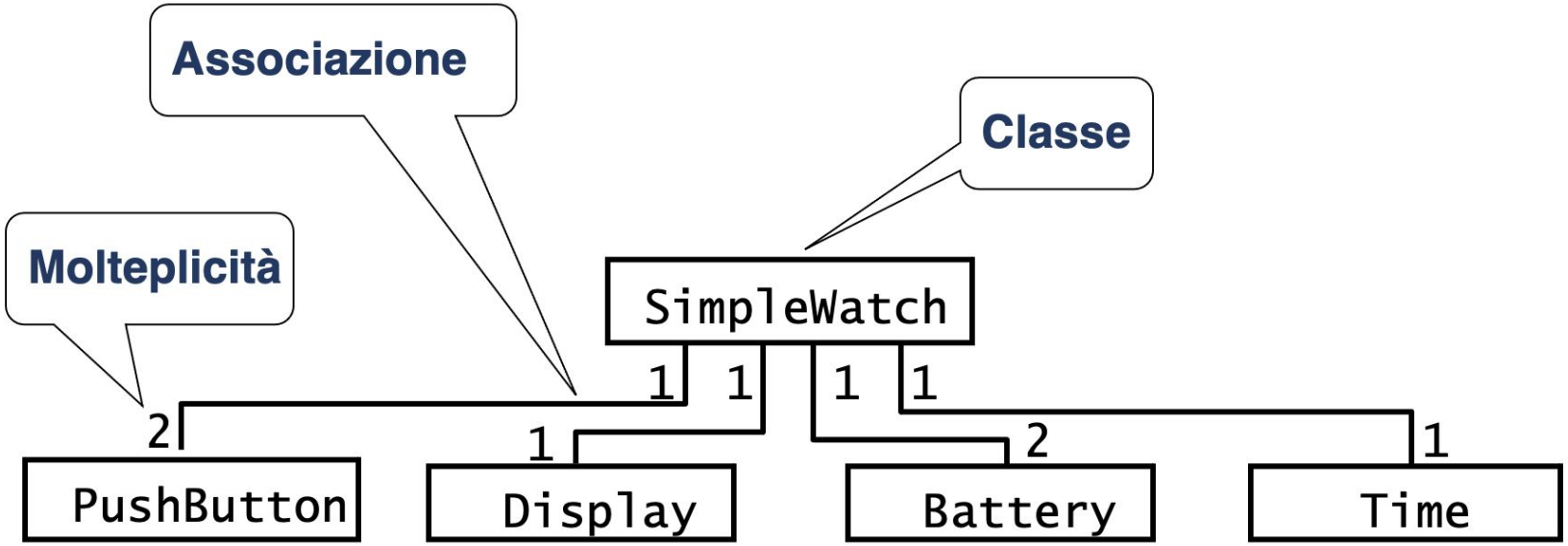
Passo 2: Distinguere le relazioni

Il passo successivo è quello di determinare come ciascuna delle classi o degli oggetti sono in relazione tra loro. Cercate i punti in comune e le astrazioni tra di loro; questo vi sarà di aiuto quando li raggrupperete quando disegnerete il diagramma di classe.

Passo 3: Creare la struttura

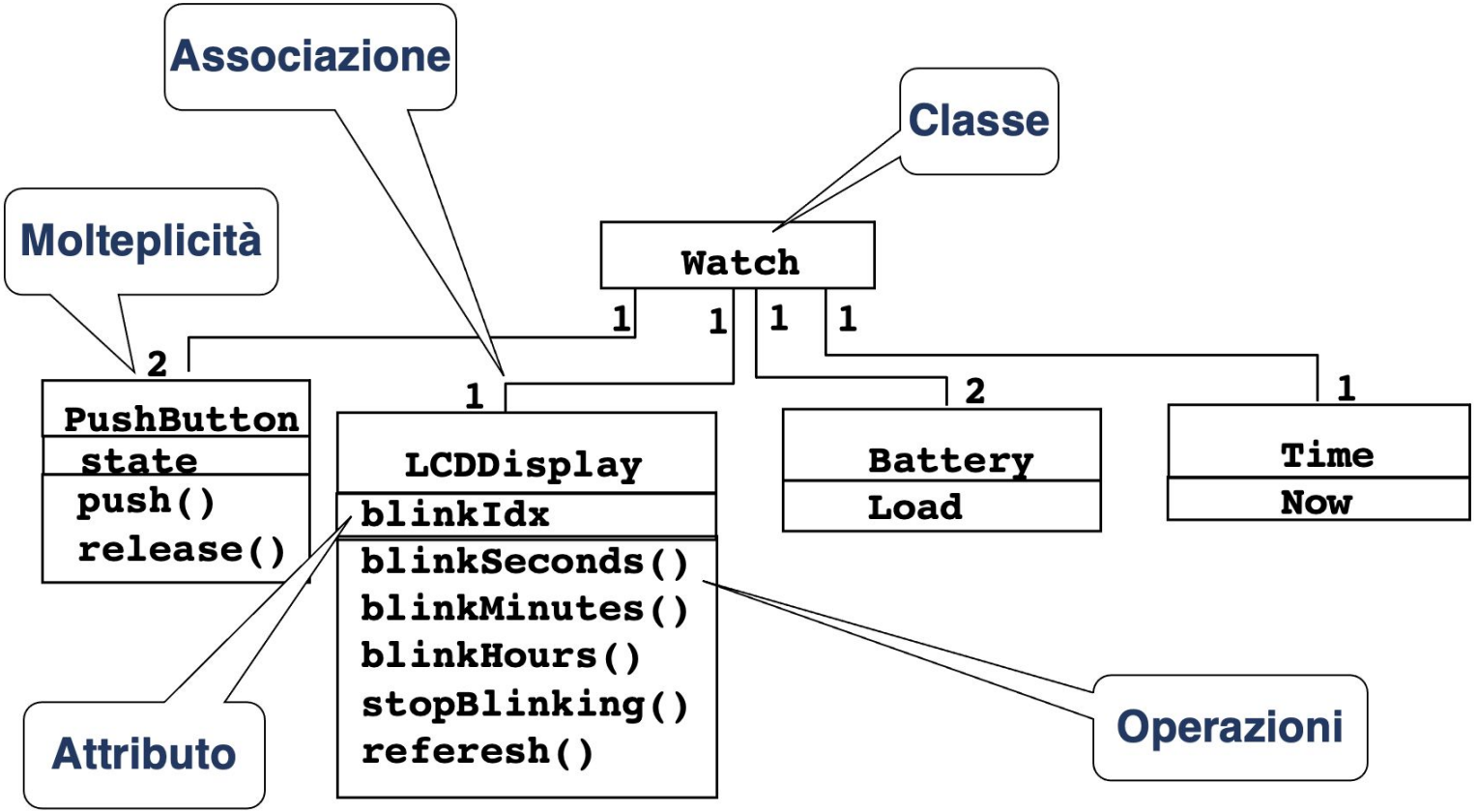
Per prima cosa, aggiungere i nomi delle classi e collegarli con i connettori appropriati. È possibile aggiungere attributi e funzioni / metodi / operazioni in un secondo momento.

Diagrammi delle classi



I diagrammi delle classi rappresentano la struttura del sistema

Diagramma delle classi



Classi e oggetti

- Un oggetto è un'entità caratterizzata da
  - Un'identità
  - Uno stato
  - Un comportamento

- Una classe descrive
  - un insieme di oggetti con caratteristiche simili
  - cioè oggetti che hanno lo stesso tipo

Convenzioni UML per classi e oggetti

- I nomi degli oggetti sono sottolineati per indicare che sono istanze
- I nomi delle classi iniziano con lettere maiuscole
- Agli oggetti, nei diagrammi degli oggetti, possono essere assegnati dei nomi (seguiti dalla loro classe) per semplificarne il riferimento
  - In questo caso, i nomi iniziano con lettere minuscole

Attore vs classe vs oggetto

- Attore
  - Un’entità da modellare esterna al sistema, interagisce con il sistema (‘passenger’)
- Classe
  - Un’astrazione che modella un’entità nel dominio applicativo o del dominio della soluzione (‘User’, ‘Ticket distributor’, ‘Server’)
- Oggetto
  - Una specifica istanza di una classe (‘Joe, il passeggero che sta comprando un biglietto dal distributore di biglietti’)

Differenze tra : diagr. delle classi e diagr. degli oggetti.

Un diagramma delle classe: diagramma strutturale statico che descrive la struttura del sistema mostrando le classi, i loro attributi, i metodi e la relazione tra le classi.

Un diagramma degli oggetti: è anche un tipo di diagramma strutturale statico che mostra una vista completa o parziale della struttura di un sistema modellato in un momento specifico.

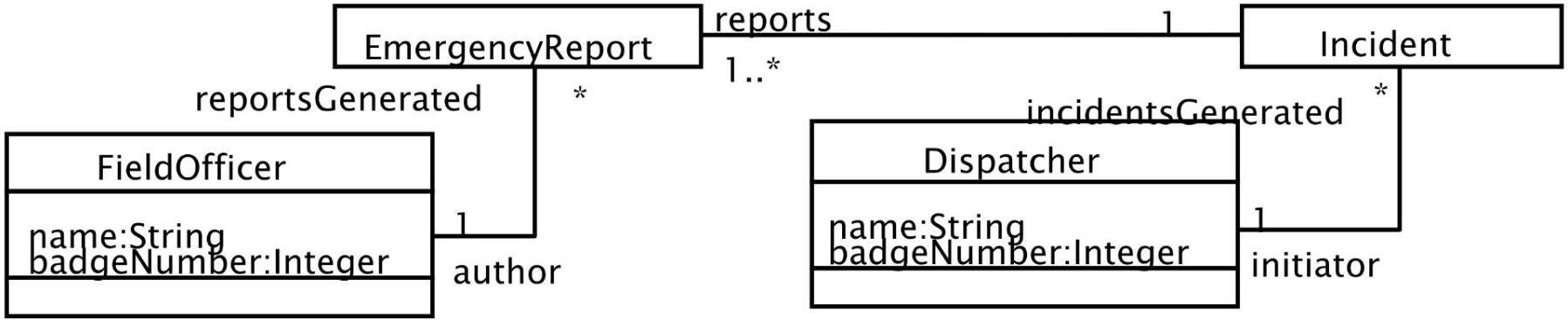
I diagrammi delle classi definiscono le classi e mostrano come si relazionano tra loro.

I diagrammi degli oggetti mostrano invece gli oggetti e le loro relazioni.

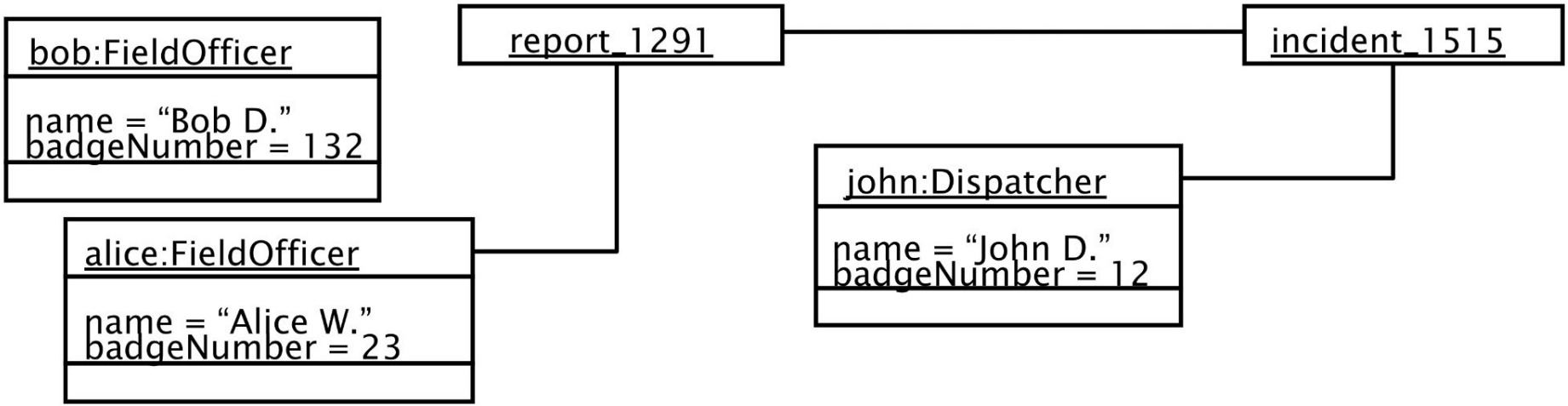
Le classi sono i modelli e **gli oggetti** sono le istanze delle classi. Questa infatti è la principale differenza tra diagramma delle classi e diagramma degli oggetti.

Ricapitolando, possiamo dire che sia i diagrammi delle classi che quelli degli oggetti rappresentano le caratteristiche statiche di un sistema. La differenza tra diagramma delle classi e diagramma degli oggetti è che il diagramma delle classi rappresenta le classi e le loro relazioni tra di loro mentre il diagramma degli oggetti rappresenta gli oggetti e le loro relazioni tra di loro in un particolare momento. Questi diagrammi infine aiutano ad ottenere una comprensione di alto livello del sistema.

Esempio di diagramm delle classi: classi partecipanti nel caso d'uso "Report Emergency"

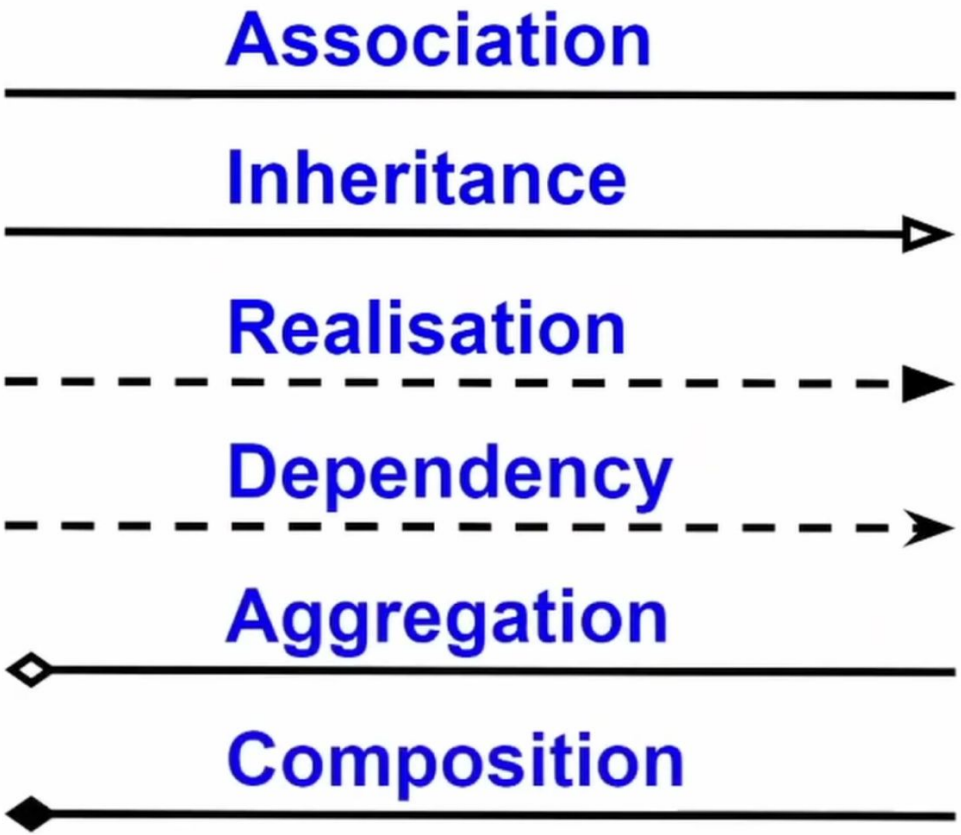


Esempio di diagramma degli oggetti: oggetti partecipanti nello scenario “warehouseOnFire”

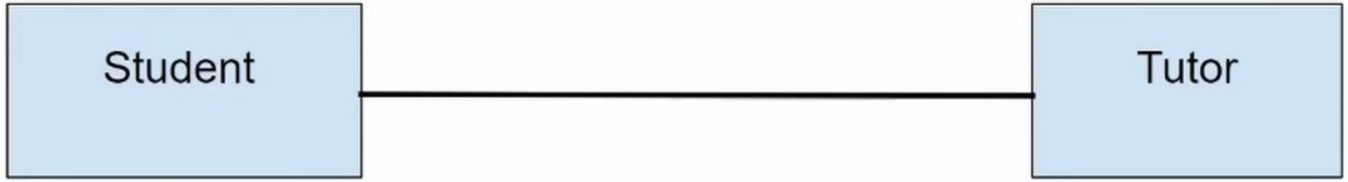


Relazioni tra classi in UML

Relationships between classes in UML

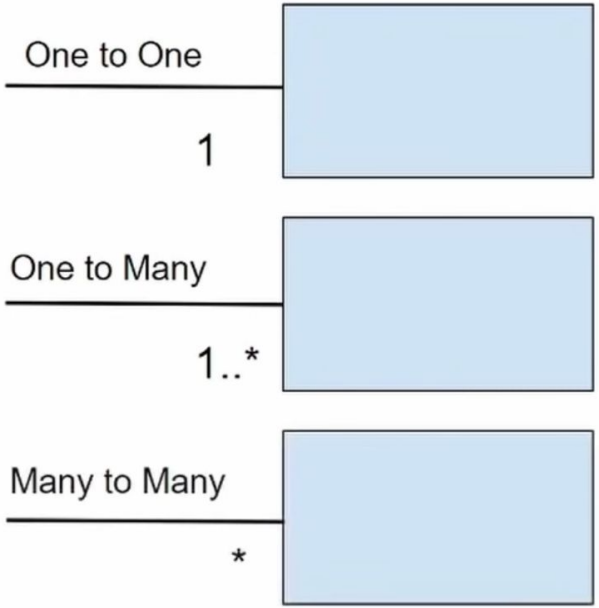


Association



L’associazione connette le classi, rappresentata da una linea continua tra le due classi.

Possiamo anche specificare la molteplicità



Attributi

- Membri di classe(privati, se possibile)
- Proprietà aggiuntive
  - Se ordered: Array o vettori
  - Se unordered: insiemi
- Convenzioni dei gruppi di programmazione
  (Esempio: Getter e setter per ogni attributo)

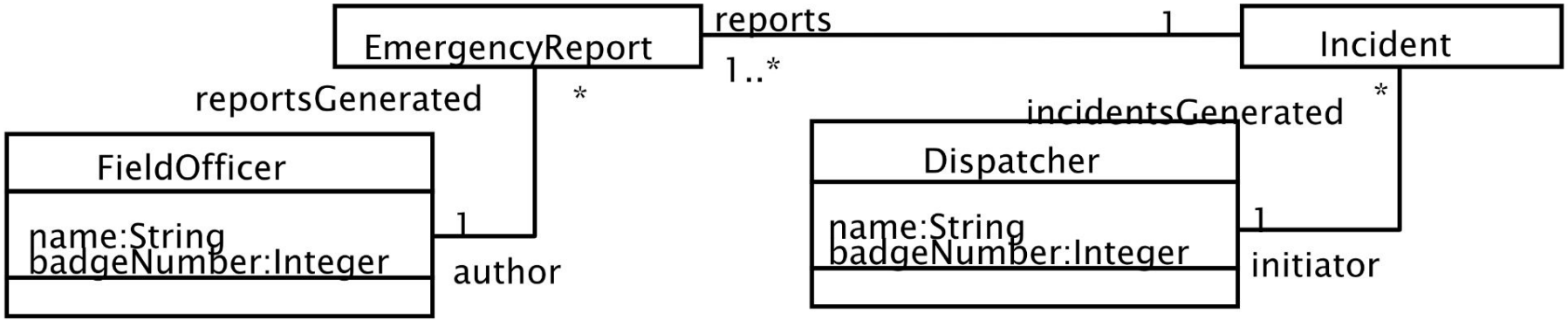
Associazioni

- Anche se etichettata con verbo, meglio renderla con un nome
- Evitare le associazioni bidirezionali

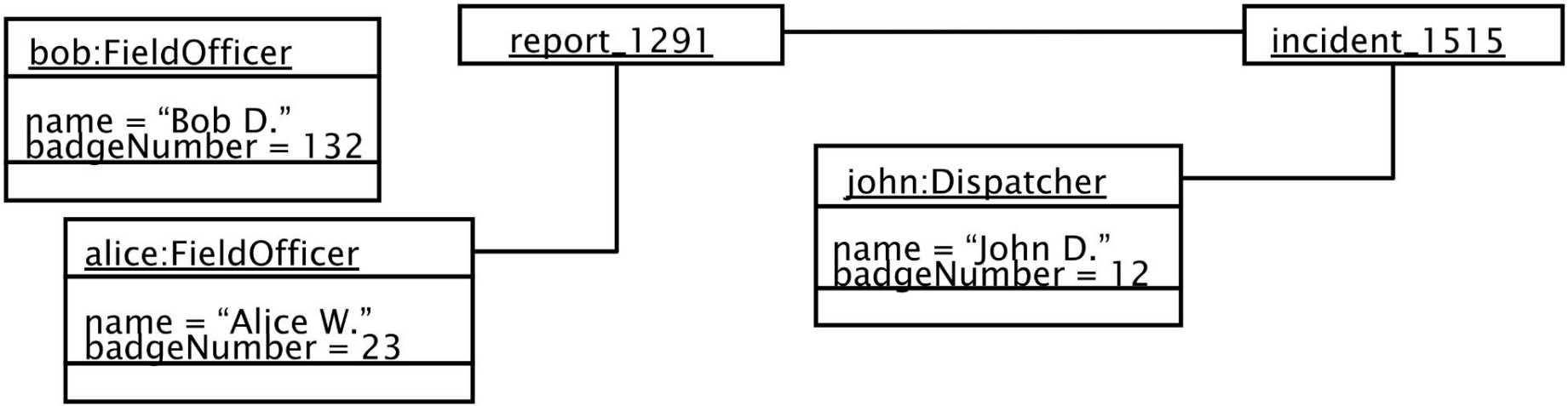
Associazioni e collegamenti (link)

- Un **collegamento** o **link** rappresenta una connessione tre due oggetti
- Le **associazioni** sono relazioni tra classi e rappresentano gruppi di link
- Nell'esempio *FRIEND*, ogni oggetto *FieldOfficer* ha anche una lista di *EmergencyReport* che sono stati scritti dal *FieldOfficer*
  - Nell'esempio del diagramma delle classi, la linea tra la classe *FieldOfficer* e la classe *EmergencyReport* è un'associazione
  - Nell'esempio del diagramma degli oggetti, la linea tra l'oggetto *alice:FieldOfficer* e l'oggetto *report_1291:EmergencyReport* è un link
    - Rappresenta uno stato del sistema per cui *alice:FieldOfficer* ha generato *report_1291:EmergencyReport*

Esempio di diagramm delle classi: classi partecipanti nel caso d'uso "Report Emergency"

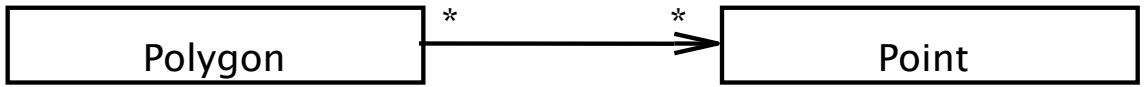


Esempio di diagramma degli oggetti: oggetti partecipanti nello scenario “warehouseOnFire”



Associazioni simmetriche e asimmetriche

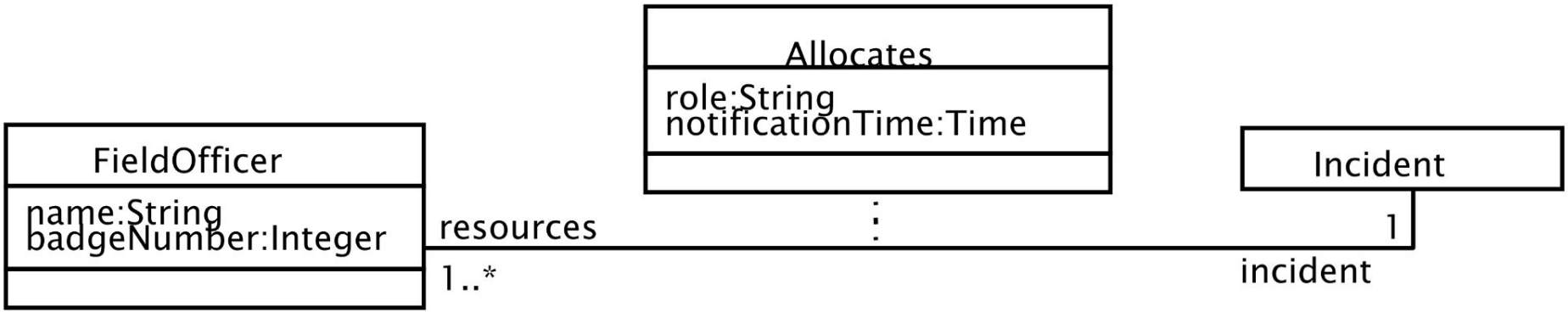
- Le associazioni possono essere bidirezionali (simmetriche) o unidirezionali (asimmetriche)
  - Nei diagrammi precedenti sono tutte simmetriche
- Una associazione asimmetrica è, ad esempio, quella tra le classi *Poligono* e *Punto*
  - La freccia di navigazione indica che il sistema supporta solo il verso da poligono a punto
    - Dato un poligono specifico, è possibile individuare tutti i punti che costituiscono il poligono. Dato un punto specifico, non è possibile individuare il poligono di cui il punto fa parte. Per convenzione, le associazioni senza frecce sono simmetriche



Classe di associazione

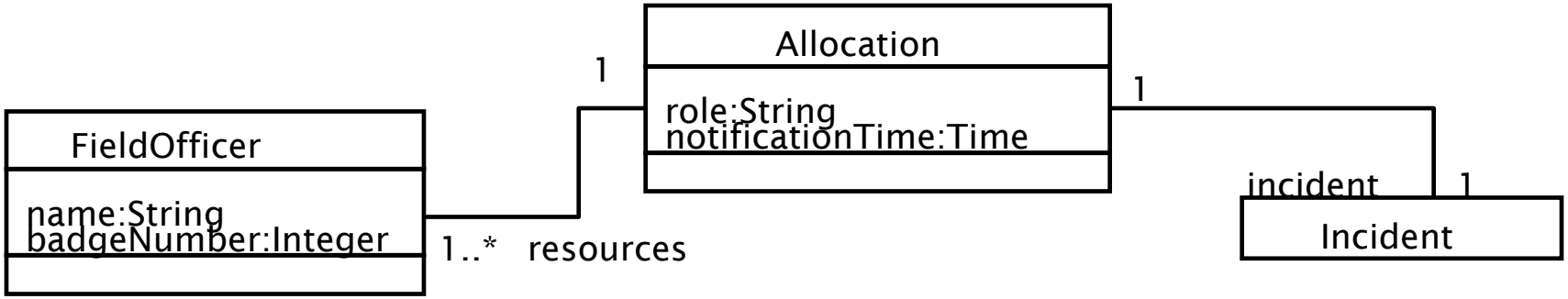
- Le associazioni sono simili alle classi poiché possono avere attributi ed operazioni
- Una tale associazione è chiamata classe di associazione
  - Disegnata con un simbolo di classe che contiene attributi e operazioni ed è connessa al simbolo di associazione con una linea tratteggiata
  - Ad esempio, l'allocazione dei *FieldOfficer* ad un Incidente è modellata come una classe di associazione con attributi ruolo (*role*) e ora di notifica (*notificationTime)*

Esempio di classe di associazione



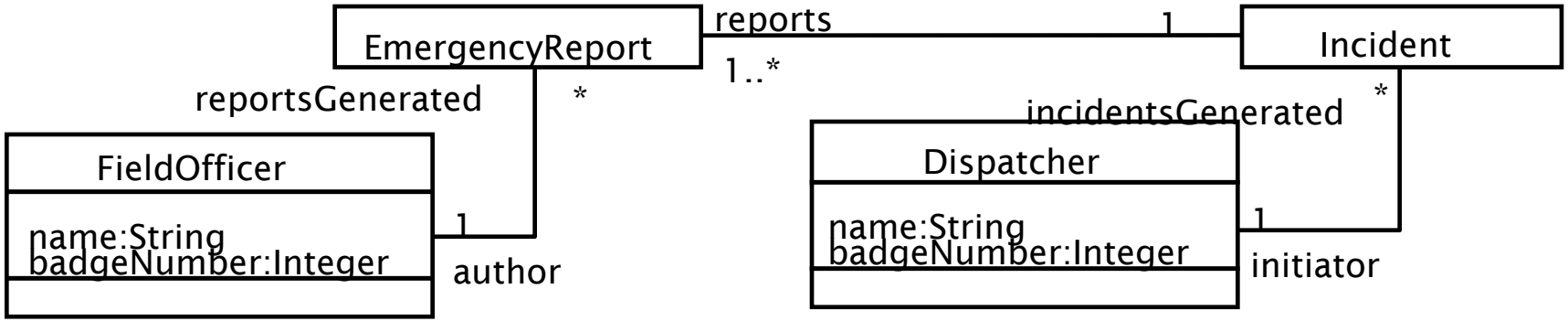
Modello alternativo di Allocation

Qualsiasi classe di associazione può essere trasformata in una classe e associazioni semplici



Ruoli

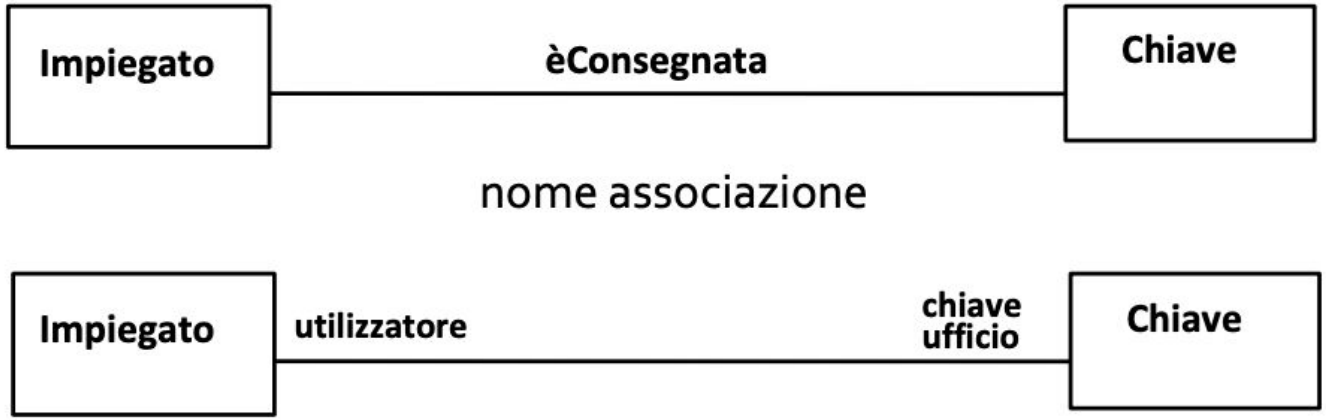
- Ciascuna estremità di un'associazione può essere etichettata con una stringa chiamata **ruolo**
- I ruoli dell'associazione tra le classi *EmergencyReport* e *FieldOfficer* sono autore (*author*) e rapporto generato (*reportGenerated*)
  - Etichettare le estremità delle associazioni con i ruoli consente di distinguere tra le multiple associazioni che si originano da una classe. Inoltre, i ruoli chiariscono lo scopo dell'associazione



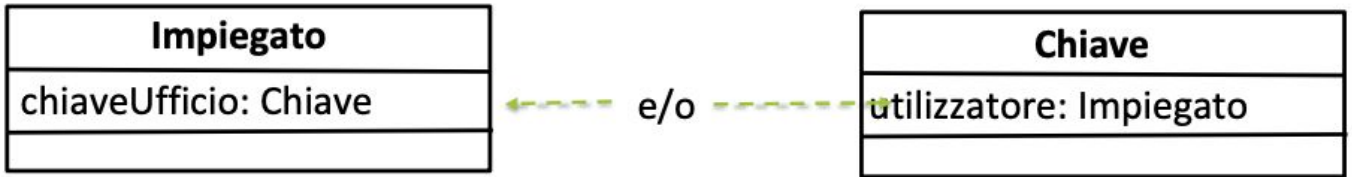
Associazione (binaria): nome e ruoli

- nome e ruoli: lowercase
- nome associazione: normalmente un verbo
  eventualmente seguito da una freccia per capire da che parte si legge
- ruolo: normalmente un sostantivo
- Formalmente opzionali,
  - è utile ci sia o il nome dell'associazione o l'indicazione dei ruoli
  - inutile entrambi

Associazione (binaria): esempio



- Si esplicitano i ruoli degli oggetti nella relazione
  - c è la chiave dell'ufficio di i
  - i ne è l'utilizzatore
- Quando si trasforma il modello in codice:



Associazione (binari): sintassi

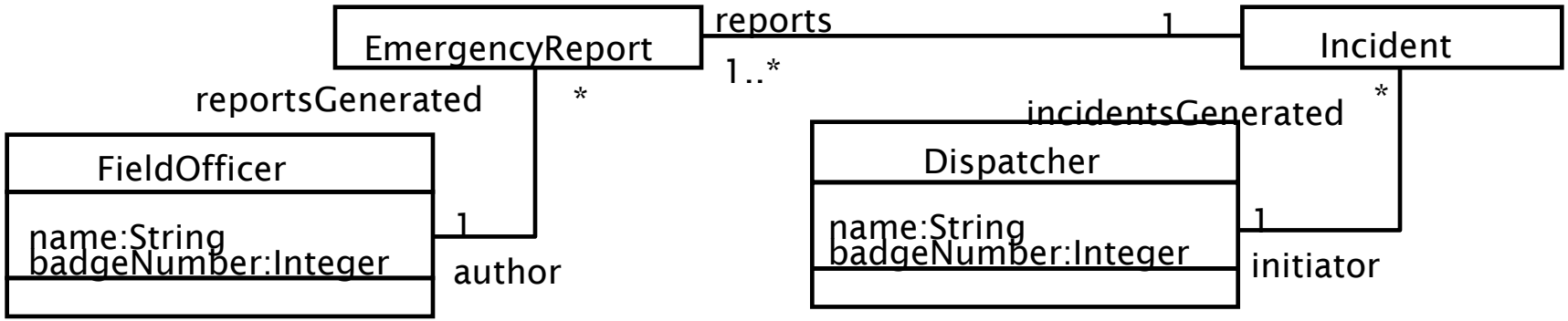
- Una linea retta tra le classi
  - A volte, e solo documentanto il codice (non il dominio) una freccia ⟶ (eventualmente doppia) per specificare la navigabilità
- Almeno uno tra nome o ruoli, raramente entrambi.
  - Servono a caratterizzare la relazione

Molteplicità

- Ogni estremità di un’associazione può essere etichettata con un insieme di interi che indicano il numero di link che si originano da un’istanza della classe connessa all’estremità dell’associazione
- Questo insieme di interi è chiamato **molteplicità** dell’estremità dell’associazione

Esempio di molteplicità

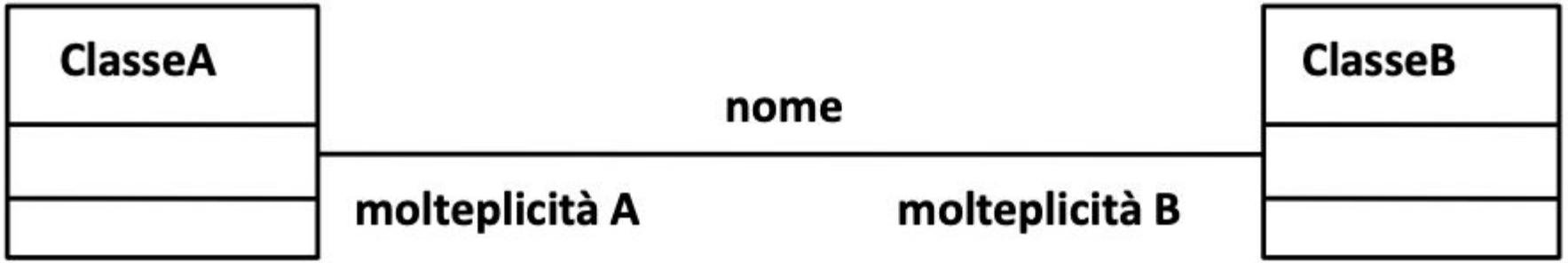
- L'estremità dell'associazione *author* ha una molteplicità pari ad 1
  - Significa che tutti gli *EmergencyReport* sono scritti esattamente da un *FieldOfficer* vale a dire, ogni oggetto *EmergencyReport* ha esattamente un link ad un oggetto della classe *FieldOfficer*
  - La molteplicità dell'estremità dell'associazione *reportsGenerated* è "molti" ed indicata con un asterisco (*) che indica 0..n



Tipi di molteplicità

- Le associazioni solitamente usate nei diagrammi sono di tre tipi:
  - **Associazione uno-a-uno**
    - Ha molteplicità 1 su entrambe le estremità: esiste esattamente un link tra le istanze di ogni classe
  - **Associazione uno-a-molti**
    - Ha molteplicità 1 su di una estremità e 0..n o 1..n dall'altra estremità: denota il rapporto di composizione tre due classi
  - **Associazione molti-a-molti**
    - Ha molteplicità 0..n o 1..n su ambo i lati: indica che un numero arbitrario di link posso esserci tra le istanze di due classi

Associazioni: vincoli di molteplicità



Numero di oggetti coinvolti nell'associazione

in un dato istante

Molteplicità, un esempio

- **Un oggetto Società può essere in relazione con molti oggetti Lavoratore**
- **Un oggetto Lavoratore può essere in relazione con un solo oggetto Società**

in un dato istante

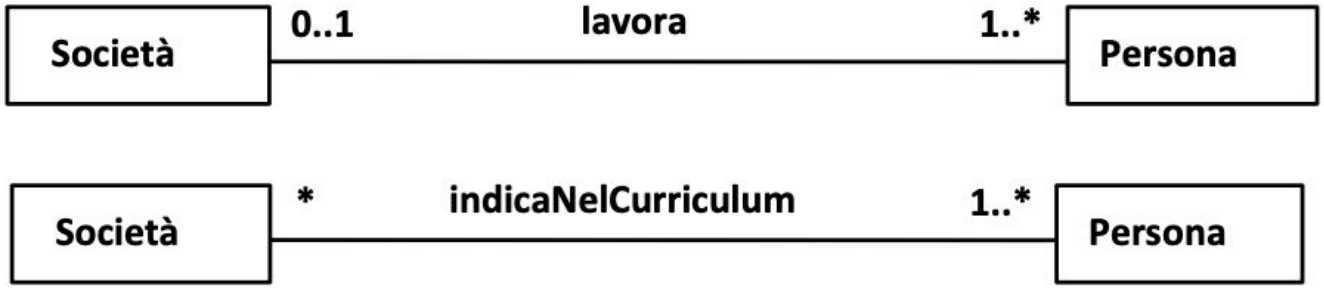
Molteplicità delle relazioni

Le molteplicità si possono definire indicando gli estremi inferiore e superiore di un intervallo

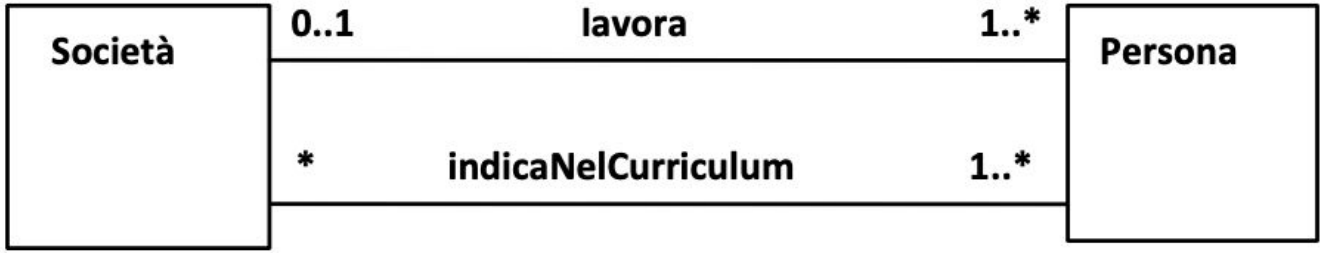
- Esempio 2..4 per le ruote di un veicolo

- l'estremo inferiore può essere zero o un numero positivo
- quello superiore un numero positivo o * (indefinito)
- n..n ≡ n
- 0..* ≡ *
- 1 è il default e si può omettere

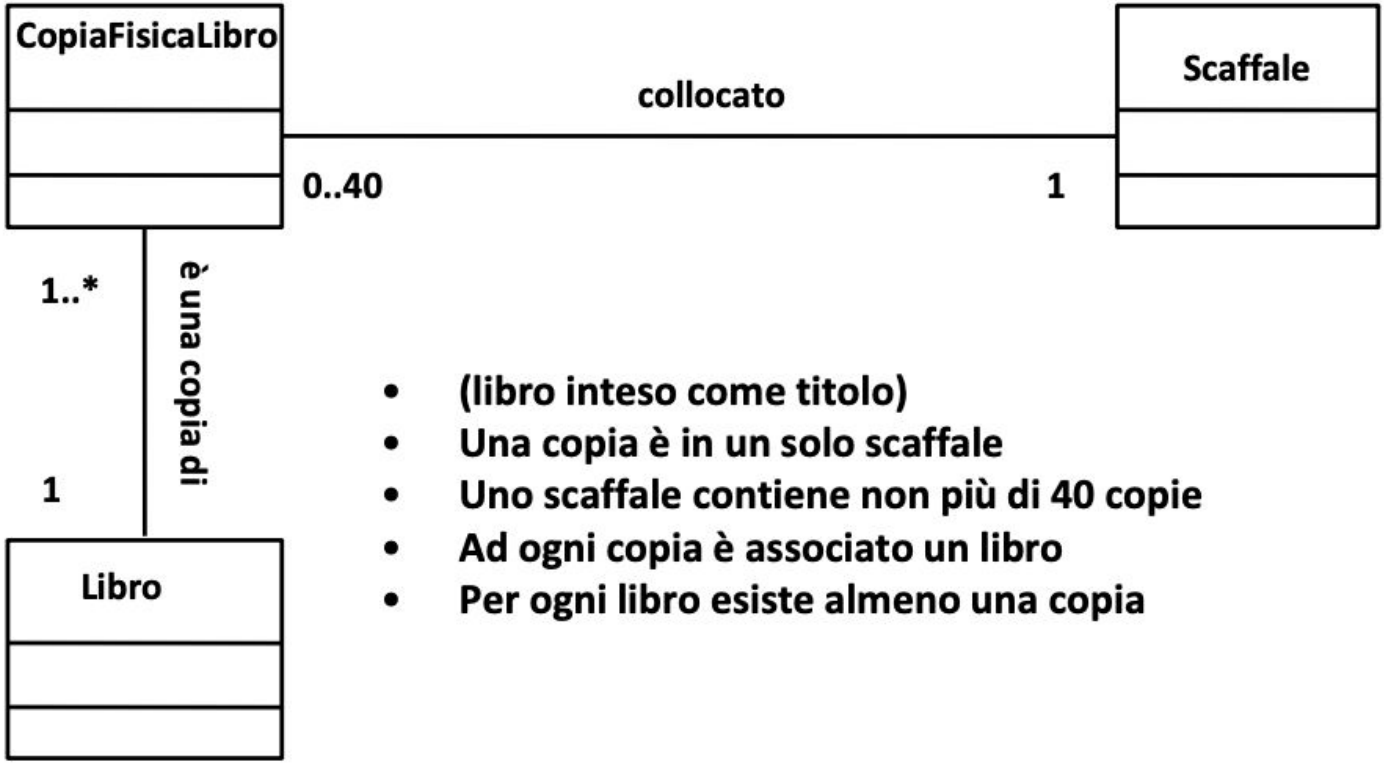
La molteplicità è legata al nome dell'associazione



Sintatticamente si possono anche avere più associazioni tra due classi



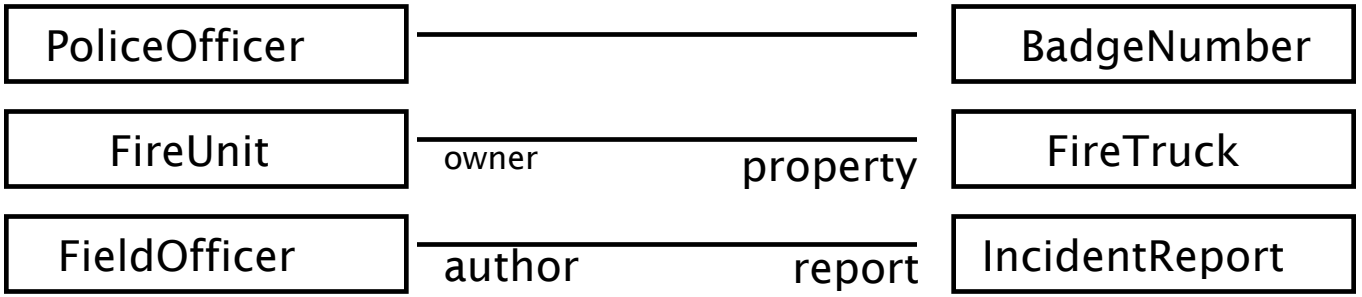
Molteplicità: esempio



- (libro inteso come titolo)
- Una copia è in un solo scaffale
- Uno scaffale contiene non più di 40 copie
- Ad ogni copia è associato un libro
- Per ogni libro esiste almeno una copia

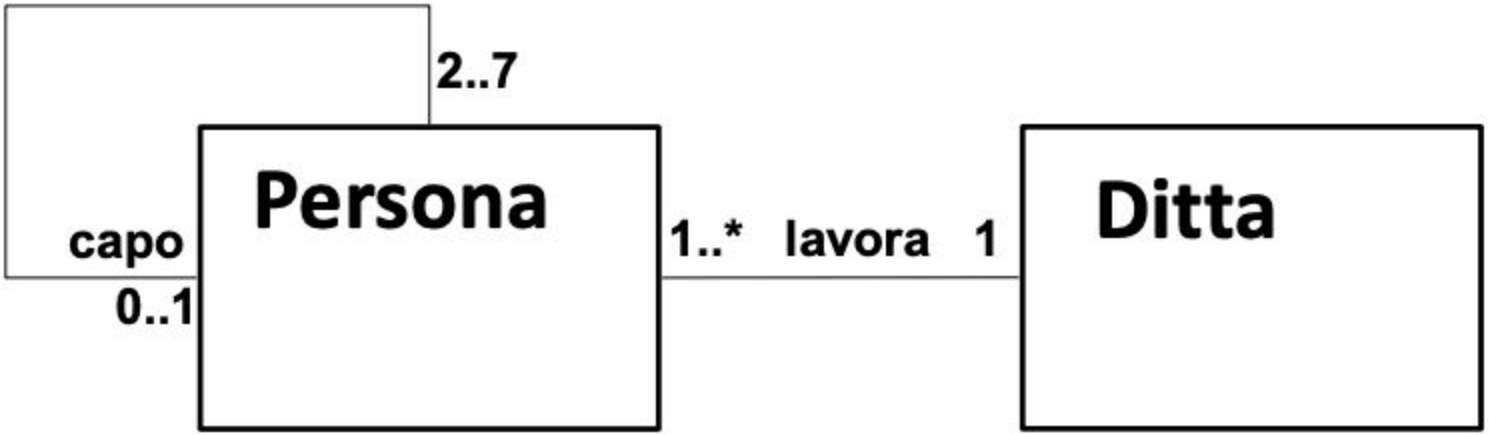
Esempi di molteplicità

che molteplicità associamo a queste associazioni?



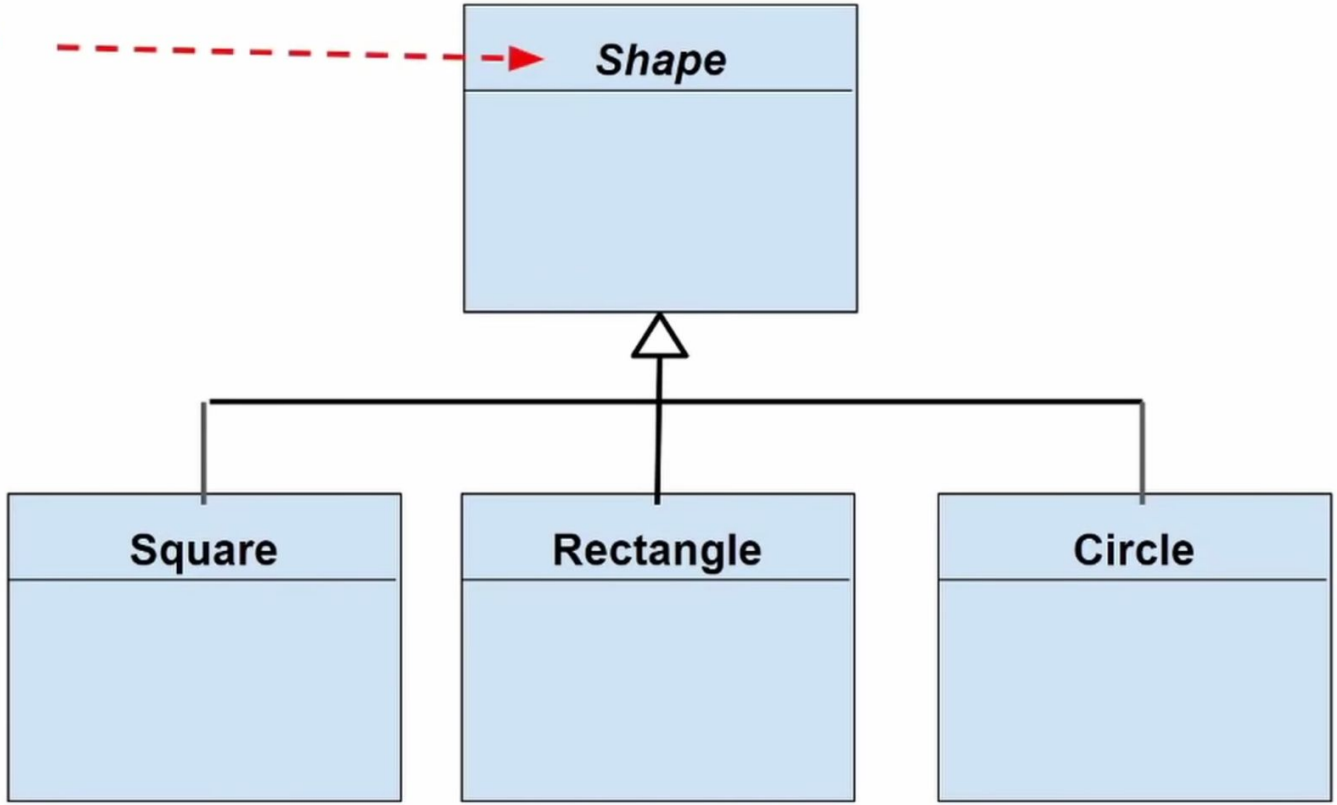
Associazioni riflessive

In questo caso è fondamentale indicare il ruolo



Inheritance

Abstract Class

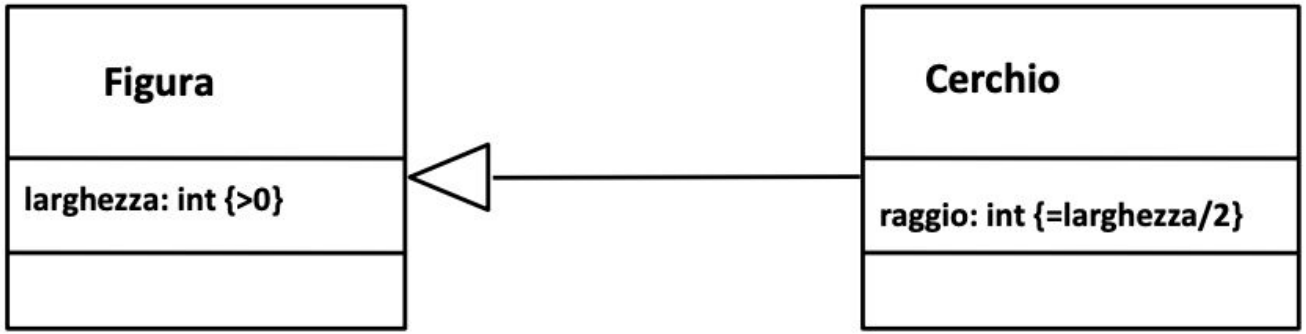


Ereditarietà

- L'**ereditarietà** è la relazione tra una classe generale ed una o più classi specializzate
- L'ereditarietà consente di descrivere tutti gli attributi e le operazioni che sono comuni ad un insieme di classi
  - Esempio: *FieldOfficer* e *Dispatcher* hanno entrambi gli attributi *name* e *badgeNumber*. Tuttavia, *FieldOfficer* ha un'associazione con *EmergencyReport*, mentre *Dispatcher* ha un'associazione con *Incident*. Gli attributi comuni di *FieldOfficer* e *Dispatcher* possono essere modellati introducendo una classe *PoliceOfficer* che è specializzata da *FieldOfficer* e *Dispatcher*
  - *PoliceOfficer* è chiamata la **superclasse**; *FieldOfficer* e *Dispatcher* sono chiamate **sottoclassi**
- Le sottoclassi ereditano gli attributi e le operazioni della loro superclasse

Ereditarietà di classe

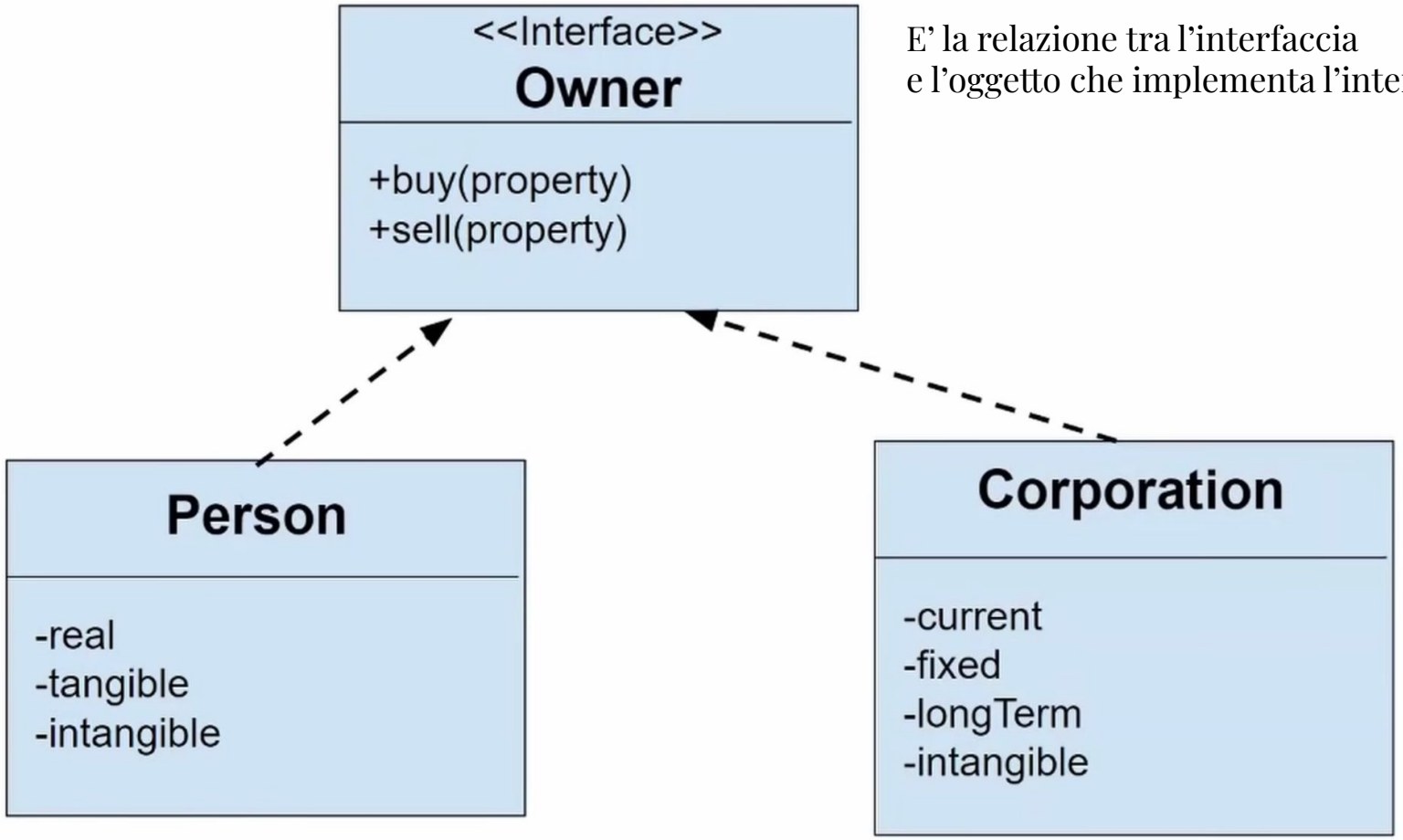
- Le sottoclassi ereditano tutte le caratteristiche della superasse:
  - attributi, operazioni , relazioni e vincoli
- Le sottoclassi possono aggiungere caratteristiche e ridefinire le operazioni



Classi astratte

- La classe *PoliceOfficer* è una classe astratta. Per distinguerla dalle classi concrete si scrive il nome in corsivo
  - Le classi astratte sono utilizzate nella modellazione orientata agli oggetti per classificare concetti collegati riducendo, quindi, la complessità totale del modello
    - Eliminano la ridondanza

Realization



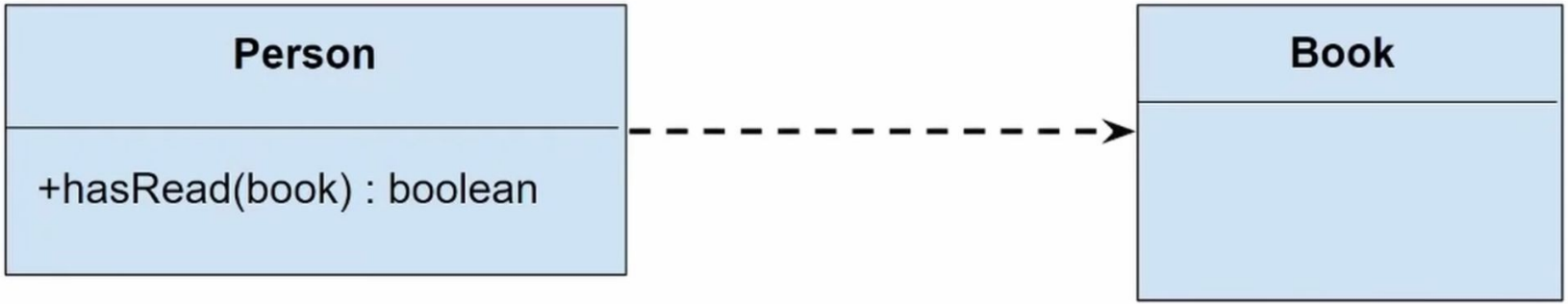
E' la relazione tra l'interfaccia
e l'oggetto che implementa l'interfaccia.

Realization

L'oggetto sorgente implementa o realizza la destinazione. Realize viene utilizzato per esprimere tracciabilità e completezza nel modello: un processo o un requisito aziendale è realizzato da uno o più casi d'uso che sono a loro volta realizzati da alcune classi, che a loro volta sono realizzate da un componente, ecc. Mappatura di requisiti, classi, ecc. attraverso la progettazione del tuo sistema, fino ai livelli di astrazione della modellazione, garantisce che il quadro generale del tuo sistema ricordi e rifletta tutte le piccole immagini e i dettagli che lo vincolano e lo definiscono. Una realizzazione è mostrata come una linea tratteggiata con una punta di freccia solida e lo stereotipo «realizzare».

Dependency

E’ un caso speciale di associazione tra due classi.



Dipendenze

Una relazione in cui le classi hanno ruolo di cliente e fornitore

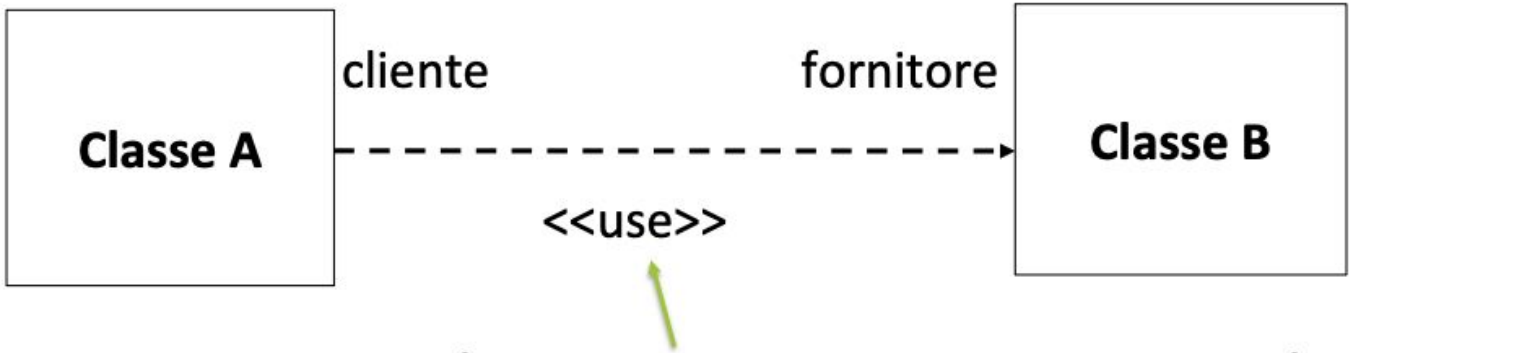
- Il cliente dipende dal fornitore

- una modifica nel fornitore può influenzare il cliente

Dipendenze d'uso: esempi

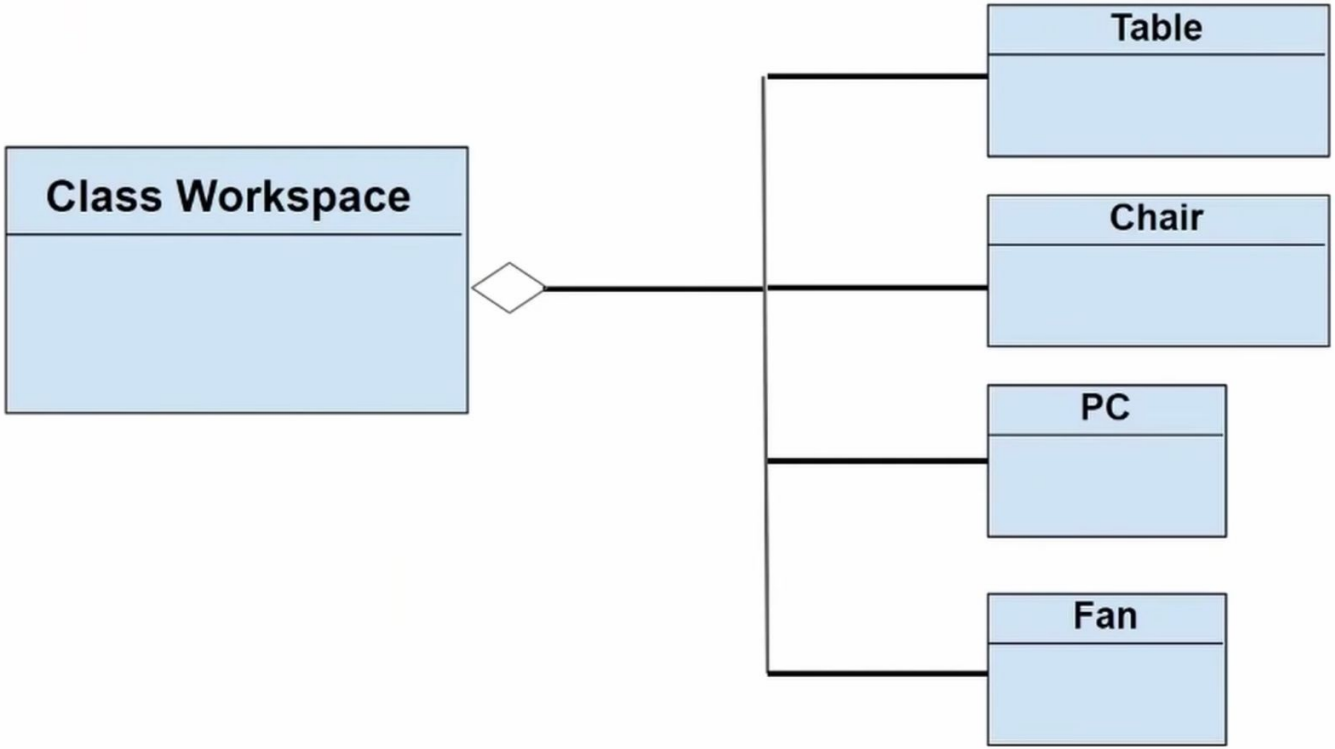
Le dipendenze più comuni

- Un parametro di un'operazione di A è di tipo B
- Un'operazione di A restituisce un oggetto di tipo B
- Un'operazione di A crea dinamicamente un oggetto di tipo B



Aggregation

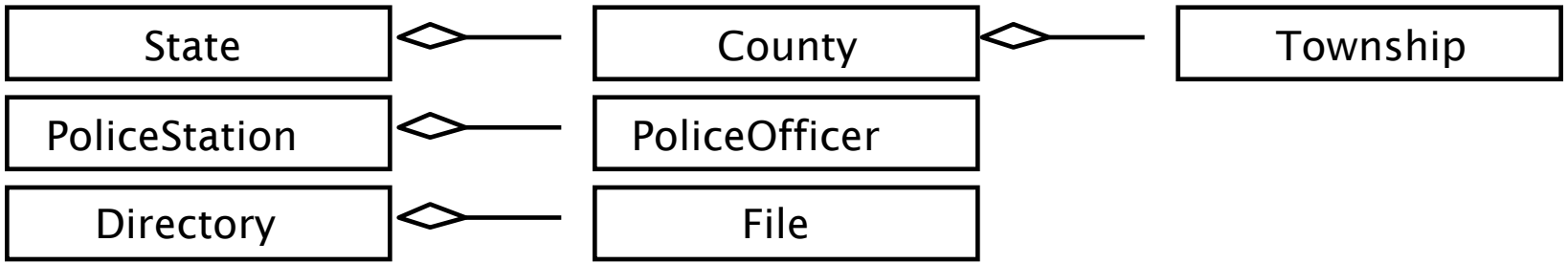
Quando una classe è composta da parti di altre classi.



Aggregazione

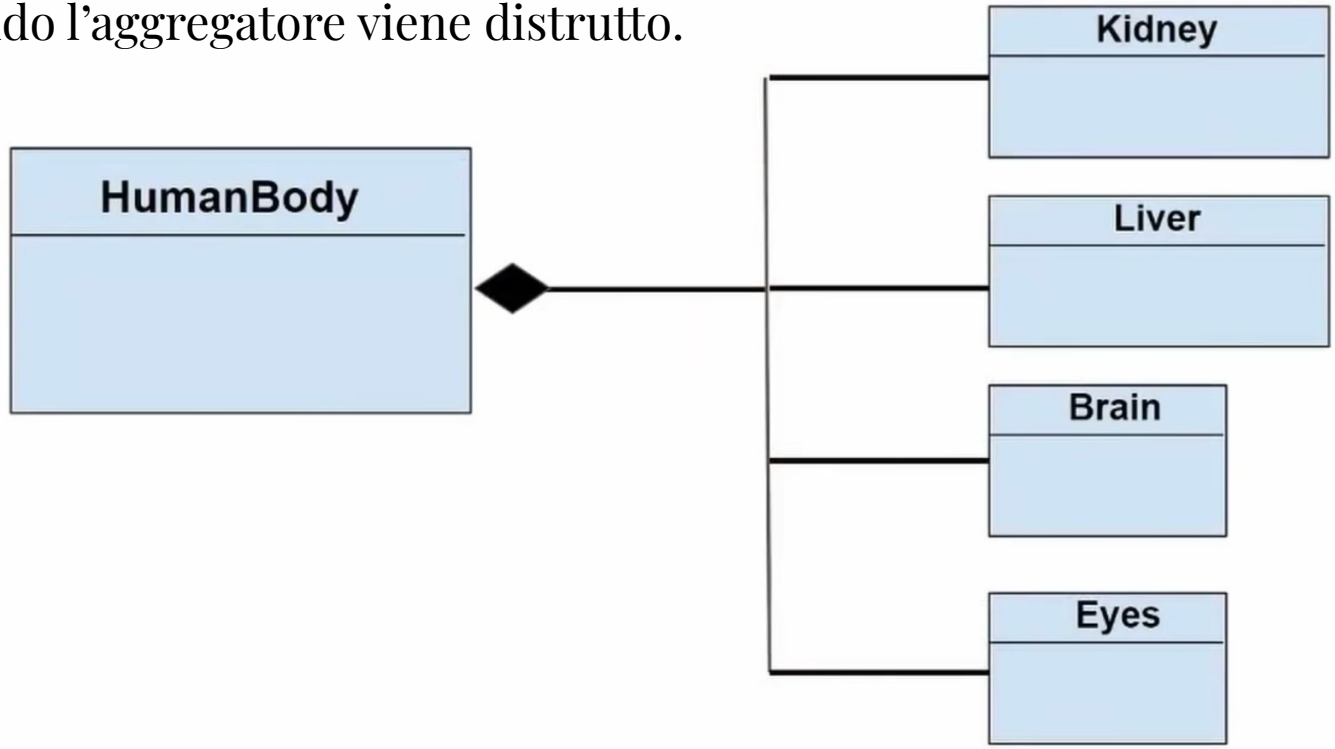
- Le associazioni sono usate per rappresentare un'ampia gamma di connessioni tra un insieme di oggetti
- Un tipo speciale di associazione si presenta frequentemente: le **aggregazioni** (denotate con una linea con testa di diamante)
  - Esempi:
    - uno *stato* contiene molti *paesi* che a loro volta contengono molte *città*
    - una *stazione di polizia* è costituita di un certo numero di *poliziotti*
    - una *directory* contiene un certo numero di *file*
  - Tali relazioni possono essere modellate con associazioni uno-a-molti
    - UML, invece, fornisce il concetto di aggregazione che consente di denotare aspetti gerarchici della relazione che può avere molteplicità sia uno-a-molti che molti-a-molti

Esempi di Aggregazione

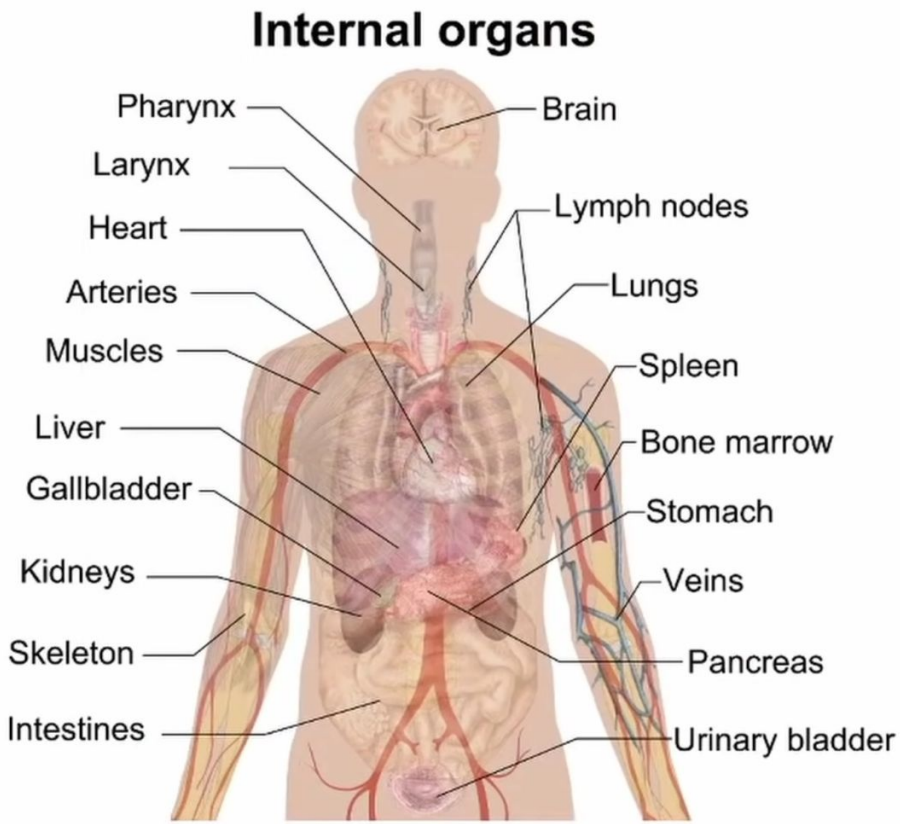


Composition

E’ un tipo di aggregazione ma le classi che formano l’aggregatore sono distrutte quando l’aggregatore viene distrutto.

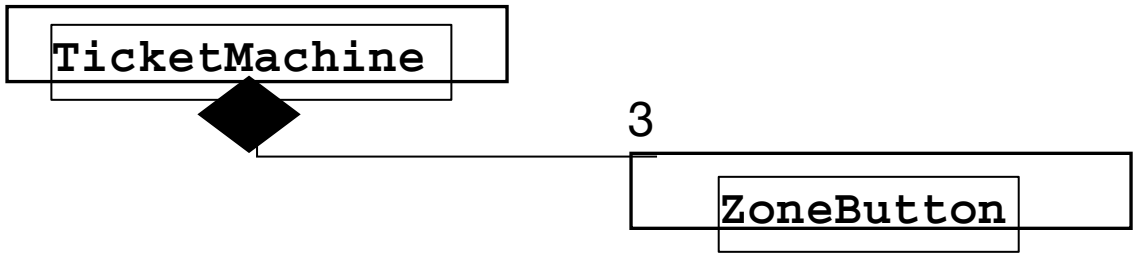


Composition



Composizione

- Un rombo solido denota una composizione
  - Una forma forte di aggregazione dove il tempo di vita delle istanze component è controllato dall'aggregato
    - Le parti non hanno esistenza autonoma ("il tutto controlla /distrugge le parti")



Aggregazione e composizione

Aggregazione e Composizione sono tipi particolari di associazione

- entrambe specificano che un oggetto di una classe è **una parte** di un oggetto di un'altra classe
- Suggerimento: prenderle in considerazione quando il nome dell'associazione sarebbe del tipo:
  - fa parte di, appartiene….

  o, simmetricamente:
  - è composto da, possiede, ha, …

Aggregazione vs Composizione

Aggregazione → relazione tra oggetti poco forte

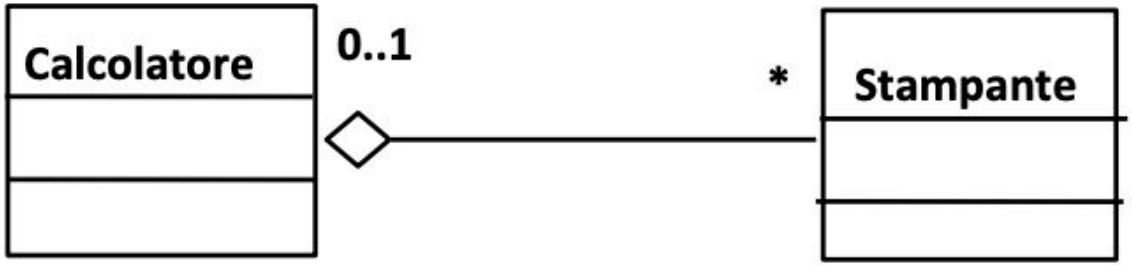
ovvero una relazione nella quale le classi parte hanno un significato anche senza che sia presente la classe tutto

Composizione → relazione tra oggetti forte

le classi parte hanno un reale significato solo se sono legate alla classe tutto

Sintassi e semantica con un esempio

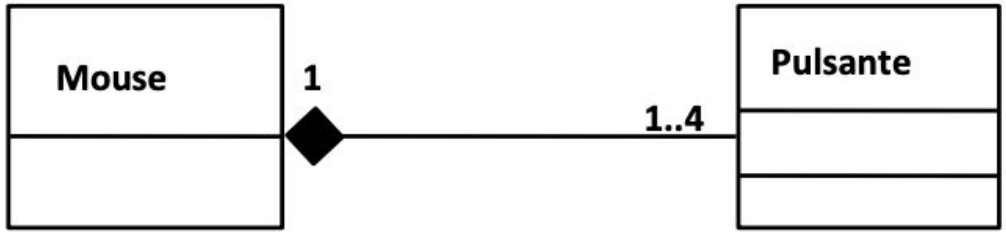
Aggregazione



- La stampante nel tempo può essere collegata a calcolatori diversi
- La stampante esiste anche senza calcolatore
- Se il calcolatore viene distrutto la stampante esiste comunque
- L'aggregazione non ha un nome

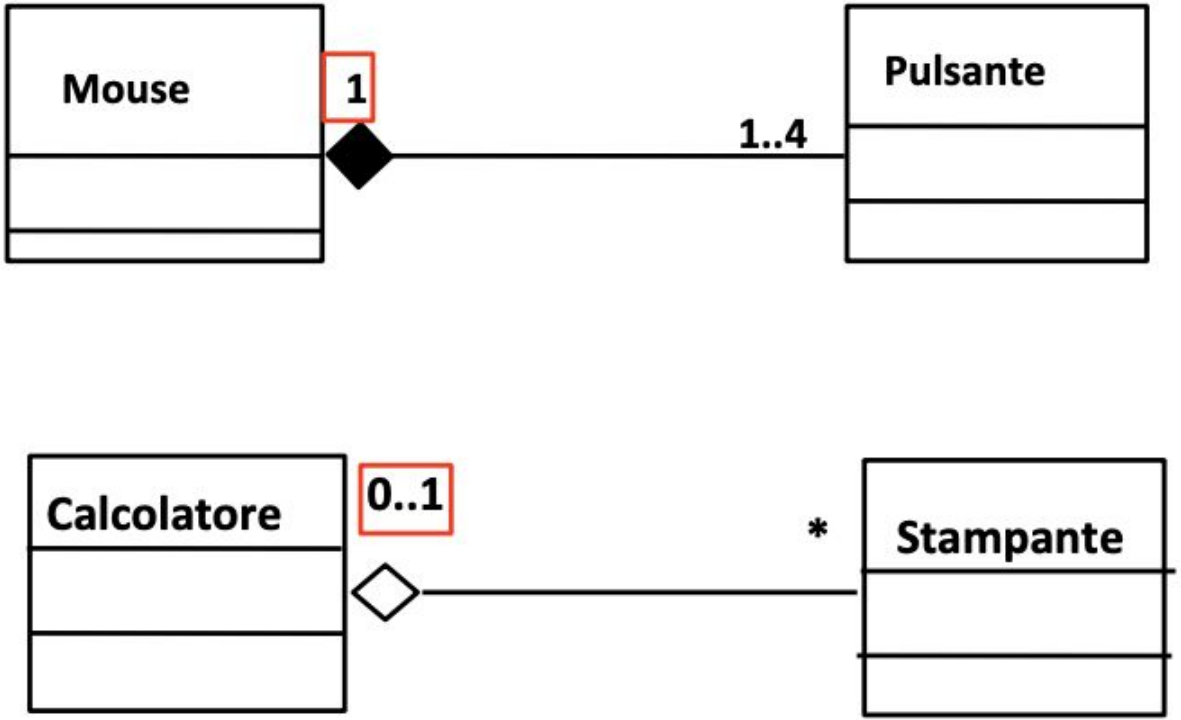
Sintassi e semantica con un esempio

Composizione



- Un pulsante appartiene a un solo mouse
- Non esiste senza il suo mouse
- Se il mouse viene distrutto vengono distrutti anche i pulsanti
- La composizione non ha un nome

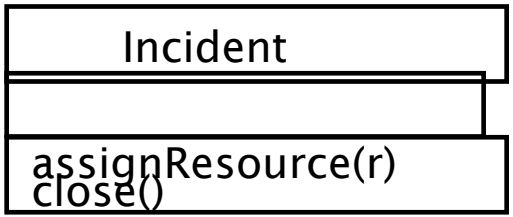
Uno sguardo alle molteplicità



Oggetti e operazioni

- Il comportamento di un oggetto è specificato dalle **operazioni**
- Un oggetto richiede l'esecuzione di un'operazione ad un altro oggetto inviandogli un messaggio
- Il messaggio è confrontato con il metodo definito dalla classe a cui l'oggetto ricevente appartiene o da una qualsiasi sua superclasse
- I metodi di una classe in un linguaggio di programmazione orientato agli oggetti sono le implementazioni di queste operazioni

Esempio di operazioni della classe Incident



Applicazione dei diagrammi delle classi

- Usati per descrivere la struttura del sistema
- Durante la fase di analisi gli ingegneri (del software) costruiscono diagrammi delle classi per formalizzare la conoscenza del dominio dell'applicazione
- Le classi rappresentano gli oggetti partecipanti individuati nei diagrammi dei casi d'uso e di interazione
  - Descrivono i loro attributi e le operazioni
- Lo scopo dei modelli di analisi è di descrivere il proposito del sistema e scoprire i suoi confini
  - Ad esempio, un analista può esaminare la molteplicità dell'associazione tra *FieldOfficer* e *EmergencyReport* e chiedere all'utente se ciò è corretto
    - E' possibile avere più di un autore per l'*EmergencyReport*?
    - Sono previsti rapporti anonimi?
- La fase di analisi tiene fuori concetti implementativi
  - I diagrammi di classe sono rifiniti durante la fase di progettazione del sistema e degli oggetti

Diagramma delle classi

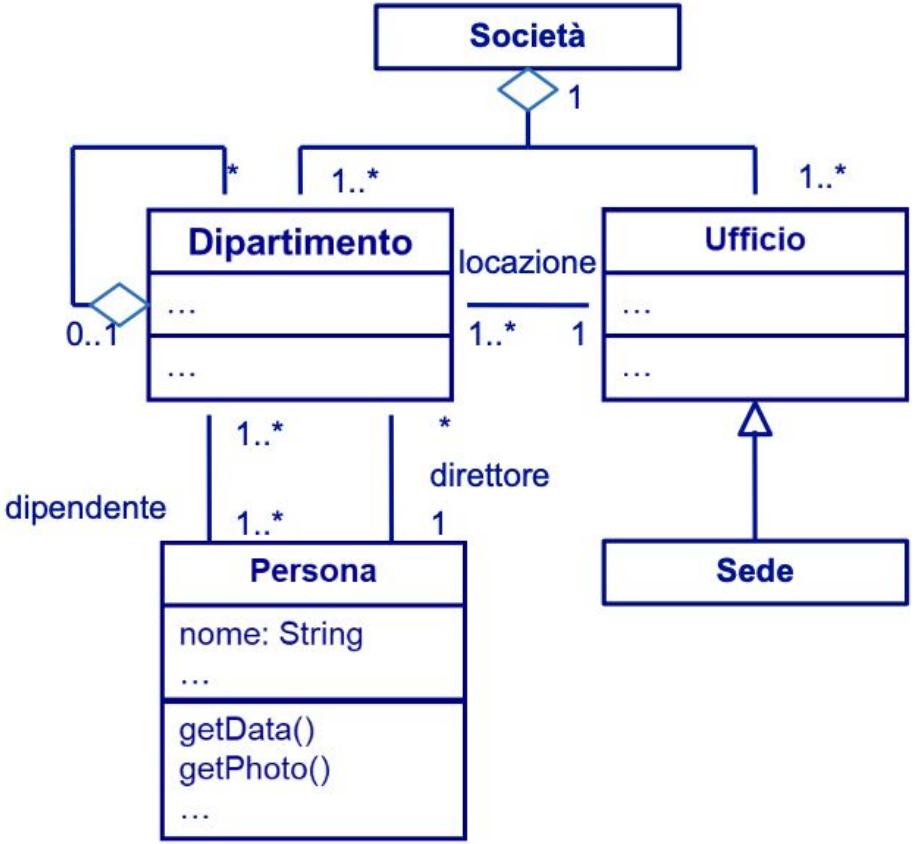
- Una classe cattura un concetto nel dominio del problema o della realizzazione

- Il diagramma delle classi descrive:
  - Il tipo degli oggetti che fanno parte di un sistema sw o del suo dominio
  - Le relazioni statiche tra essi: gli elementi e le relazioni tra essi non cambiano nel tempo

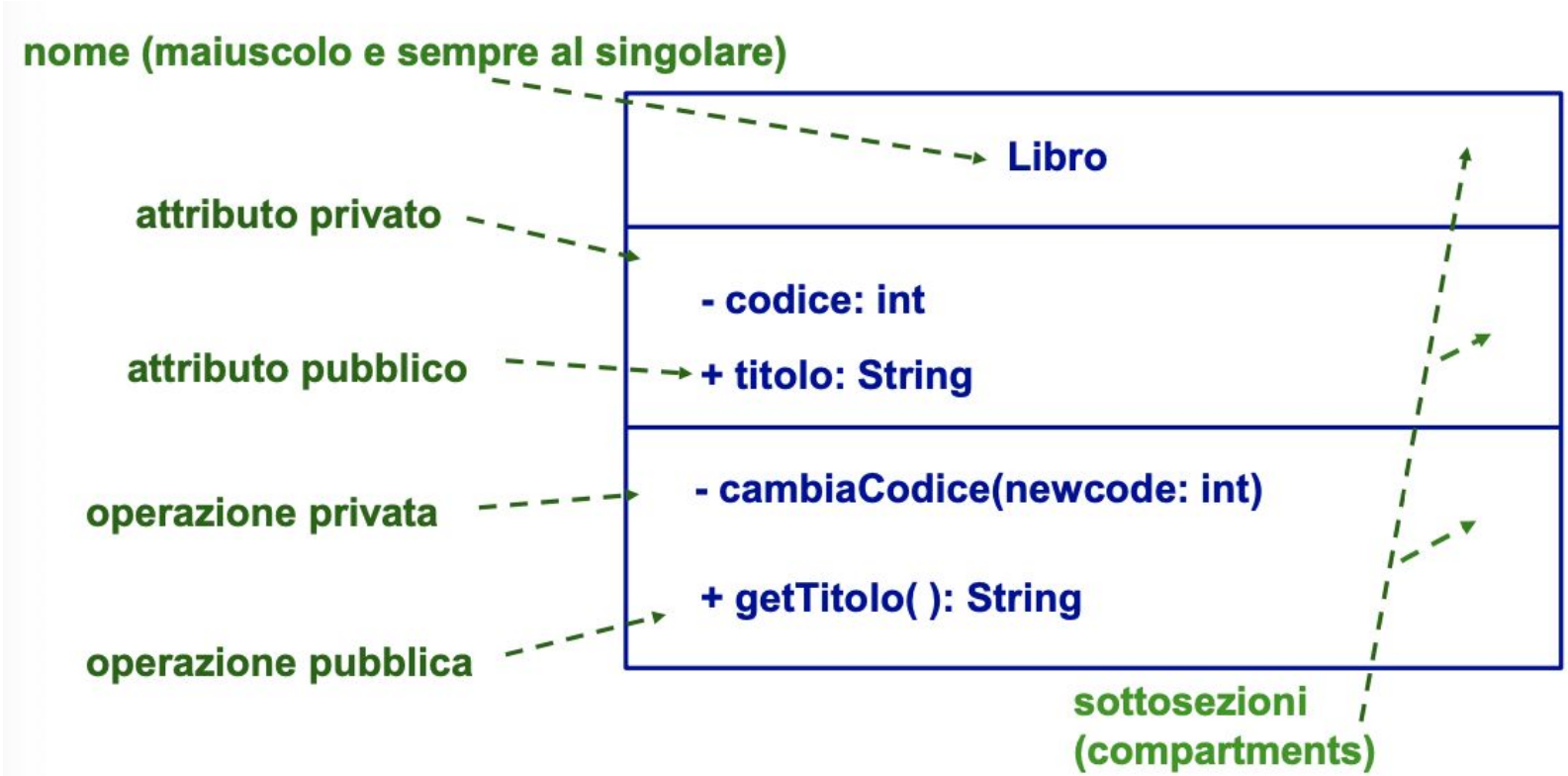
- I diagrammi delle classi mostrano anche le proprietà e le operazioni di una classe

ESEMPIO: classi

- Una società è formata da dipartimenti e uffici
- Un dipartimento ha un direttore e più dipendenti
- Un dipartimento è situato in un ufficio
- Esiste una struttura gerarchica dei dipartimenti
- Le sedi sono uffici



Sintassi della classe



Usi del diagramma delle classi

Il diagramma delle classi puo' essere usato

- a diversi livelli di dettaglio
- in diverse fasi del progetto
- fino alla generazione del codice in linguaggi OO

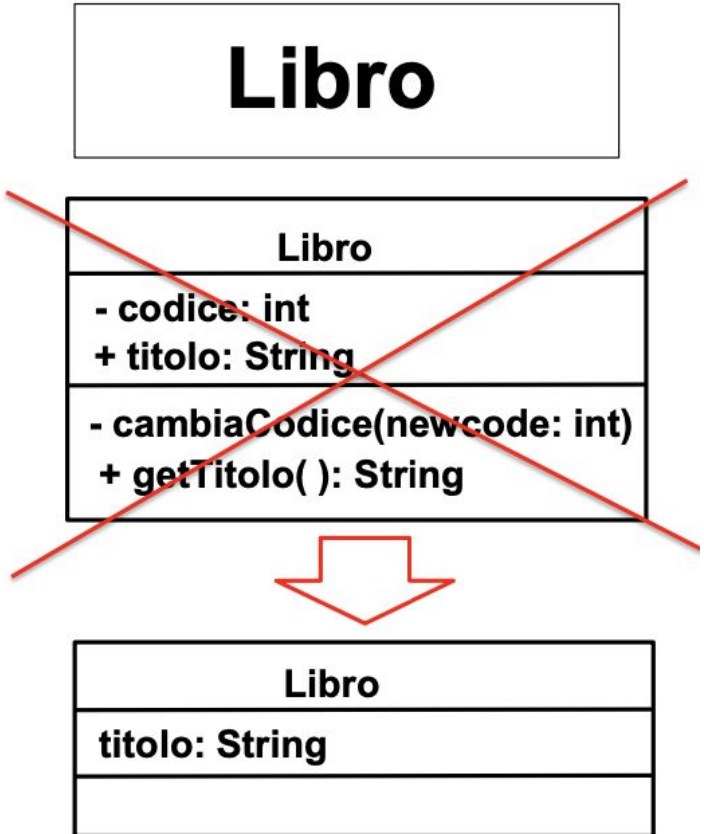
Semantica

- Un oggetto è un'entità caratterizzata da
  - Un'identità, uno stato, un comportamento
- (I valori de)gli attributi definiscono lo stato dell'oggetto
- Le operazioni definiscono il suo comportamento

Non è obbligatorio indicare attributi e operazioni

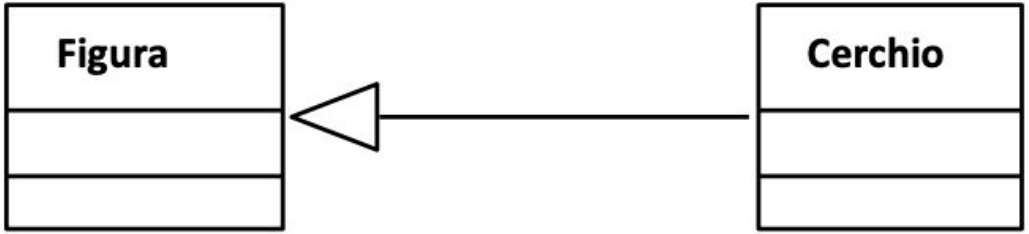
Quando si usa il diagramma delle classi per descrivere il dominio:

- le operazioni sono ritenute a un livello di dettaglio eccessivo e normalmente si omettono
  - In particolare MAI setters e getters
- gli attributi utili per caratterizzare l'elemento del dominio si specificano, dettagli implementativi no
- Le visibilità si omettono

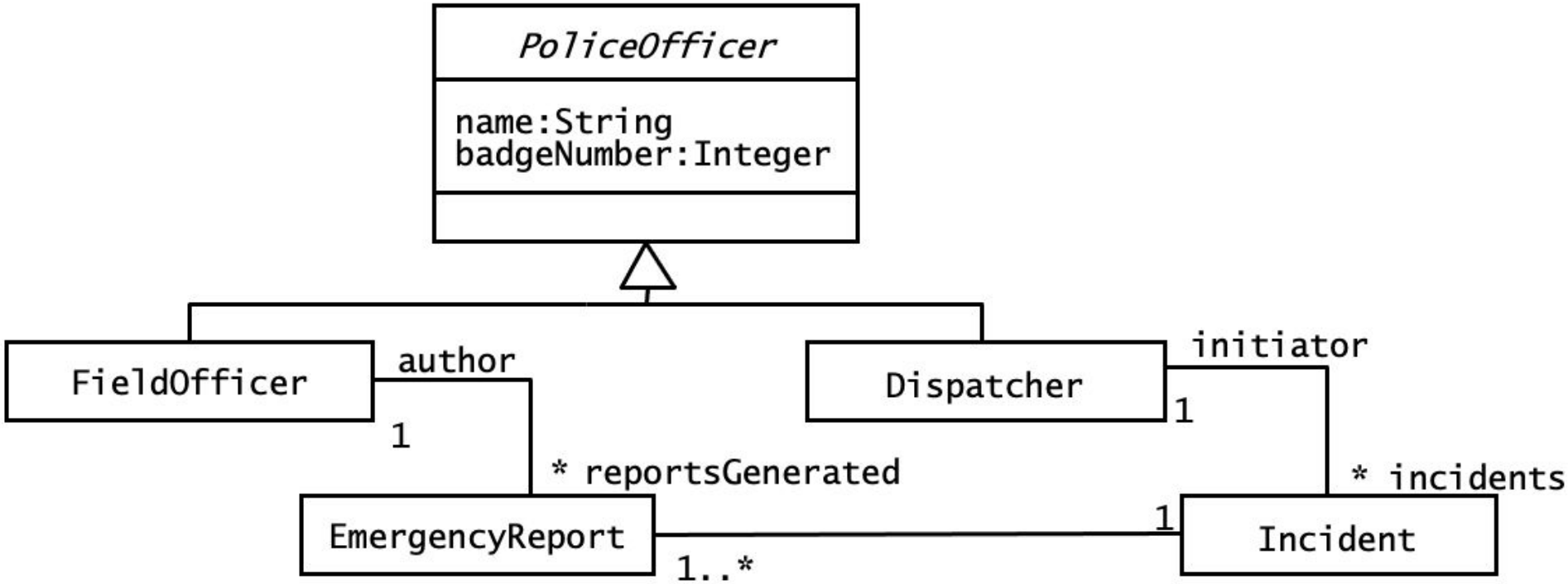


Generalizzazione

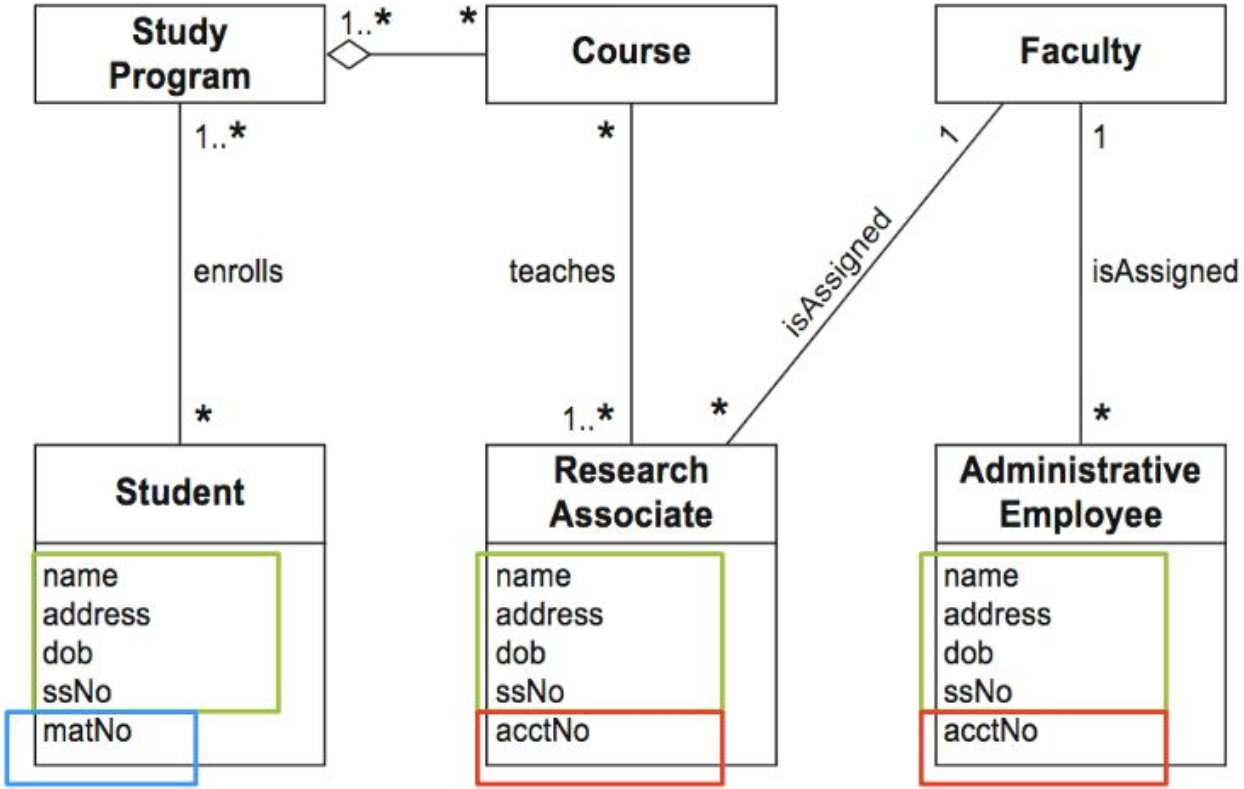
- Relazione tra un elemento generico e uno più specializzato
- L'elemento più specializzato è completamente consistente con quello più generico ma contiene più informazione
- Vale il principio di sostituzione della Liskov: l'elemento specializzato può essere usato al posto dell'elemento generico
- "è un tipo di"



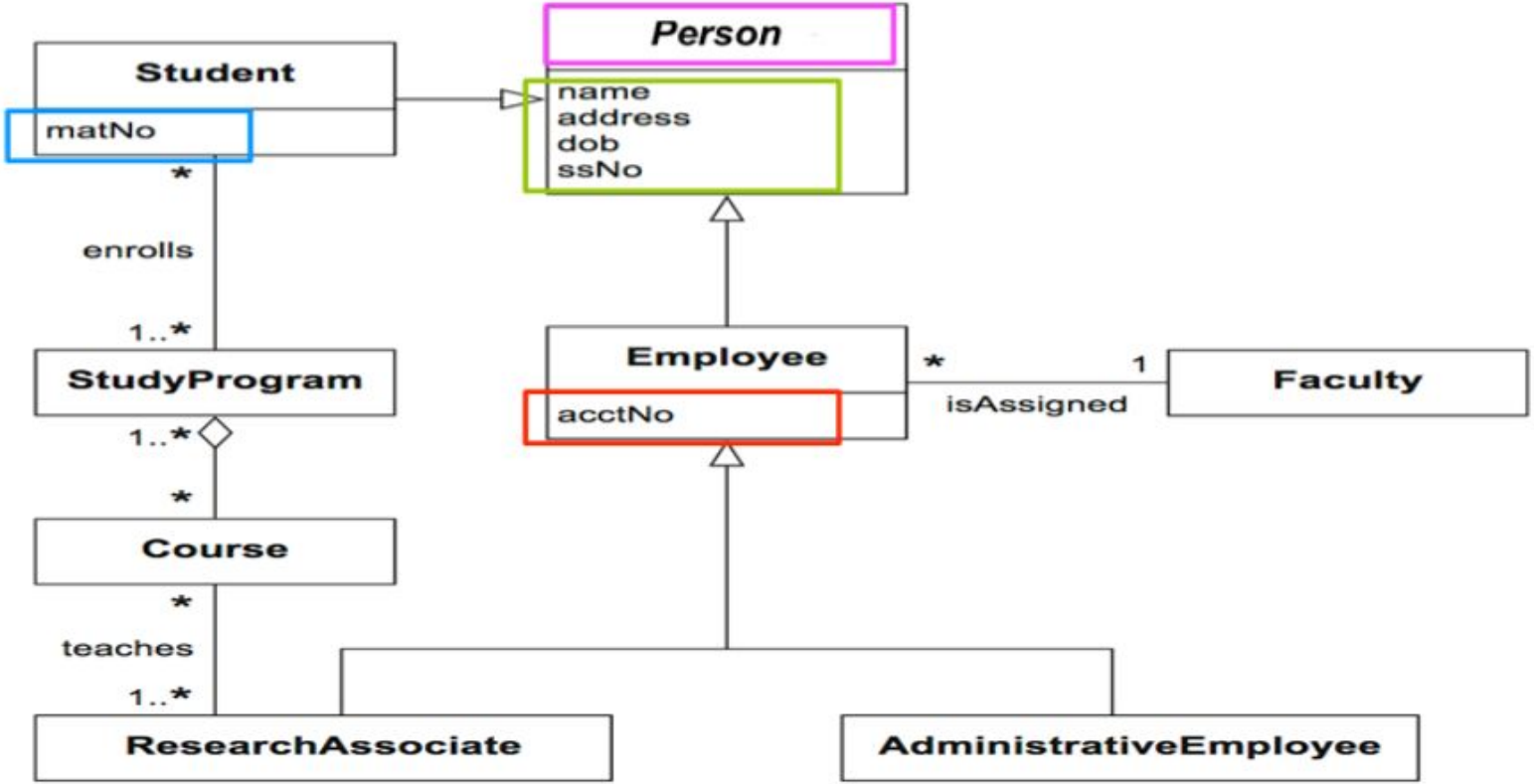
Esempi di generalizzazione



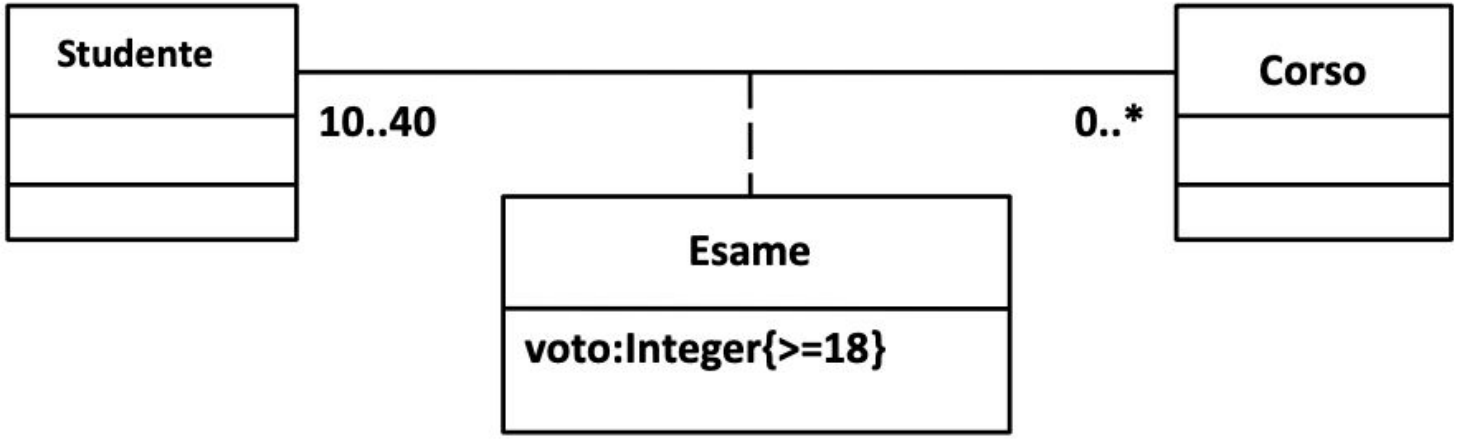
Esempio generalizzazione



Esempio generalizzazione



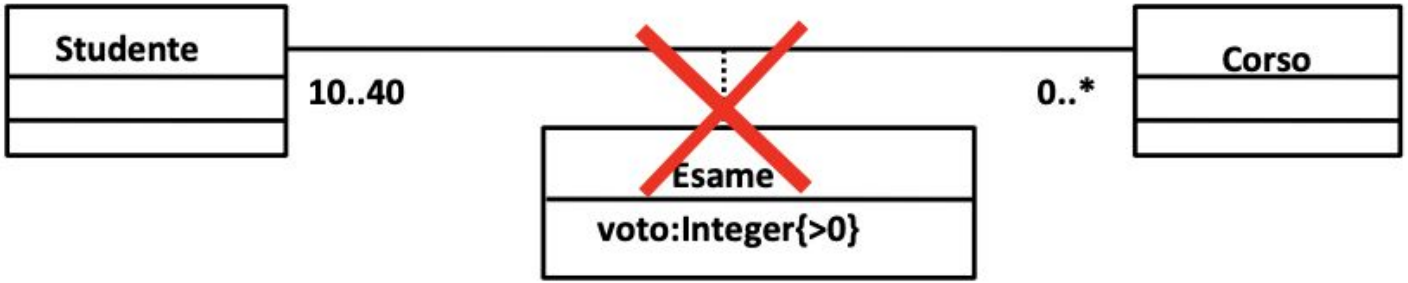
Classi associazione



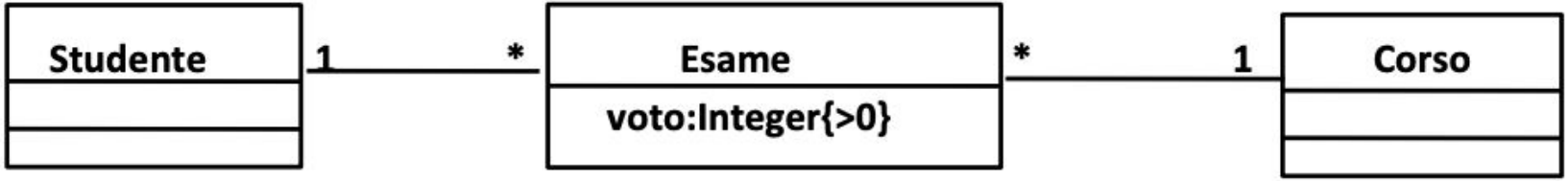
- **Un'associazione può avere attributi propri, rappresentati con una classe associazione.**
- **Le istanze sono collegamenti con attributi propri.**
- **Voto non è attributo né di Corso né di Studente**

Classi associazione (2)

Per ogni coppia di oggetti collegati tra loro può esistere un unico oggetto della classe associazione



Se vogliamo tenere traccia dei voti negativi non si possono usare le classi associazione



Identificazione delle classe

- Principali tecniche
  - Approccio data driven: tipico della fase di analisi
    - Si identificano tutti i dati del sistema e si dividono in classi (ad esempio mediante identificazione dei sostantivi)
  - Approccio responsibility driven: soprattutto durante la progettazione
    - Si identificano le responsabilità e si dividono in classi

Analisi nome-verbo

- Sostantivi → classi o attributi
- Verbi → responsabilità o operazioni
- Passi:
  - Individuazione delle classi
  - Assegnazione di attributi e responsabilità alle classi
  - Individuazione di relazioni tra le classi

Analisi nome-verbo

- Problemi ricorrenti :
  - Tagliare le classi inutili
    - Trattare i casi di sinonimia
  - Individuare le classi nascoste cioè le classi implicite del dominio del problema che possono anche non essere mai menzionate esplicitamente
    - In un sistema di gestione degli orari delle lezioni di un corso universitario, nella descrizione testuale potrebbe non essere mai nominata l'aula, che invece deve essere inserita nel modello

Dalla documentazione ufficiale UML *

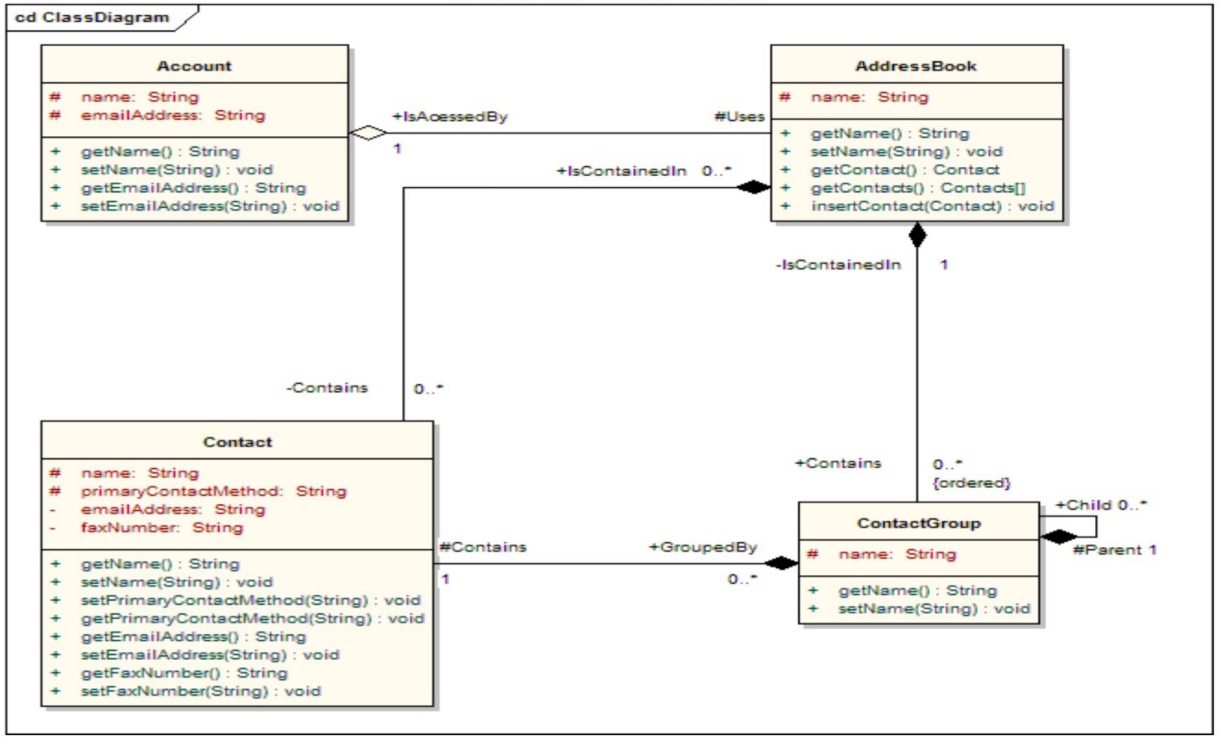
*dispensa

UML 2 Class Diagram

Class Diagrams

The Class diagram shows the building blocks of any object-orientated system. Class diagrams depict the static view of the model or part of the model, describing what attributes and behaviour it has rather that detailing the methods for achieving operations. Class diagrams are most useful to illustrate relationships between classes and interfaces. Generalizations, aggregations, and associations are all valuable in reflecting inheritance, composition or usage, and connections, respectively.

The diagram below illustrates aggregation relationships between classes. The lighter aggregation indicates that the class Account uses AddressBook, but does not necessarily contain an instance of it. The strong, composite aggregations by the other connectors indicate ownership or containment of the source classes by the target classes, for example Contact and ContactGroup values will be contained in AddressBook.



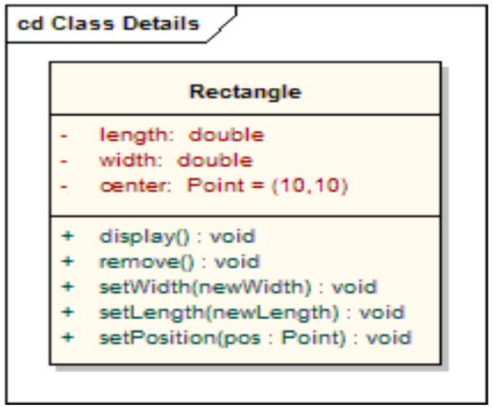
Classes

A class is an element that defines the attributes and behaviours that an object is able to generate. The behaviour is the described by the possible messages the class is able to understand along with operations that are appropriate for each message. Classes may also contain definitions of constraints tagged values and stereotypes.

Class Notation

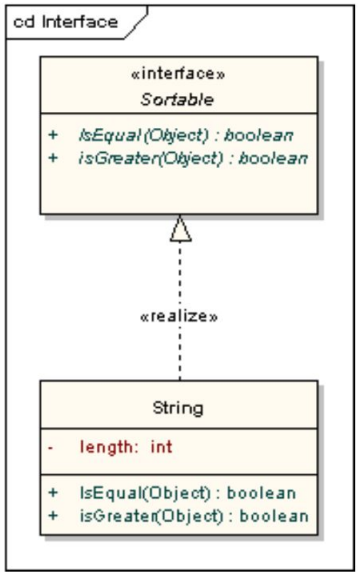
Classes are represented by rectangles which show the name of the class and optionally the name of the operations and attributes. Compartments are used to divide the class name, attributes and operations. Additionally constraints, initial values and parameters may be assigned to classes.

In the diagram below the class contains the class name in the topmost compartment, the next compartment details the attributes, with the "center" attribute showing initial values. The final compartment shows the operations, the setWidth, setLength and setPosition operations showing their parameters. The notation that precedes the attribute or operation name indicates the visibility of the element, if the + symbol is used the attribute or operation has a public level of visibility, if a - symbol is used the attribute or operation is private. In addition the # symbol allows an operation or attribute to be defined as protected and the ~ symbol indicates package visibility.

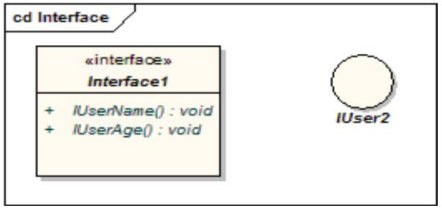


Interfaces

An interface is a specification of behaviour that implementers agree to meet. It is a contract. By realizing an interface, classes are guaranteed to support a required behaviour, which allows the system to treat non-related elements in the same way – i.e. through the common interface.

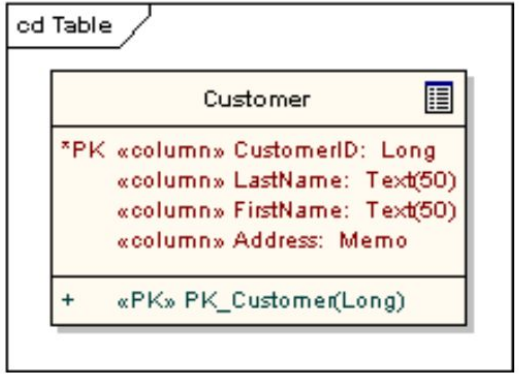


Interfaces may be drawn in a similar style to a class, with operations specified, as shown below. They may also be drawn as a circle with no explicit operations detailed. When drawn as a circle, realization links to the circle form of notation are drawn without target arrows.



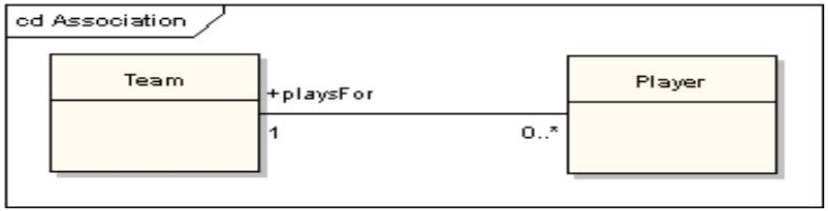
Tables

A table is a stereotyped class. It is drawn with a small table icon in the upper right corner. Table attributes are stereotyped «column». Most tables will have a primary key, being one or more fields that form a unique combination used to access the table, plus a primary key operation which is stereotyped «PK». Some tables will have one or more foreign keys, being one or more fields that together map onto a primary key in a related table, plus a foreign key operation which is stereotyped «FK».



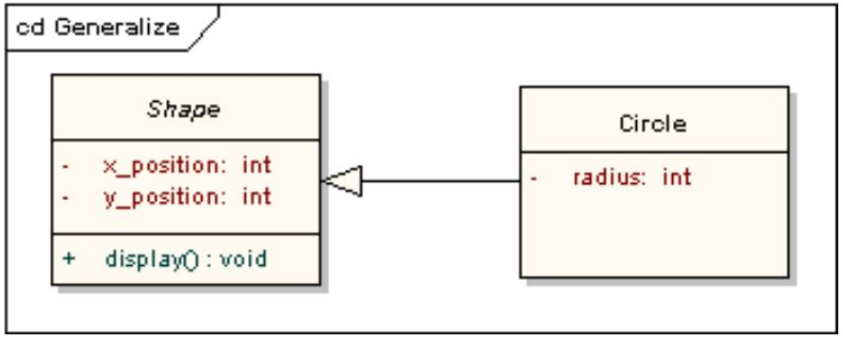
Associations

An association implies two model elements have a relationship - usually implemented as an instance variable in one class. This connector may include named roles at each end, cardinality, direction and constraints. Association is the general relationship type between elements. For more than two elements, a diagonal representation toolbox element can be used as well. When code is generated for class diagrams, associations become instance variables in the target class.

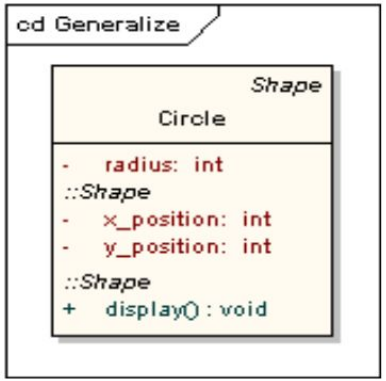


Generalizations

A generalization is used to indicate inheritance. Drawn from the specific classifier to a general classifier, the generalize implication is that the source inherits the target's characteristics. The following diagram shows a parent class generalizing a child class. Implicitly, an instantiated object of the Circle class will have attributes x_position, y_position and radius and a method display(). Note that the class Shape is abstract, shown by the name being italicized.



The following diagram shows an equivalent view of the same information.

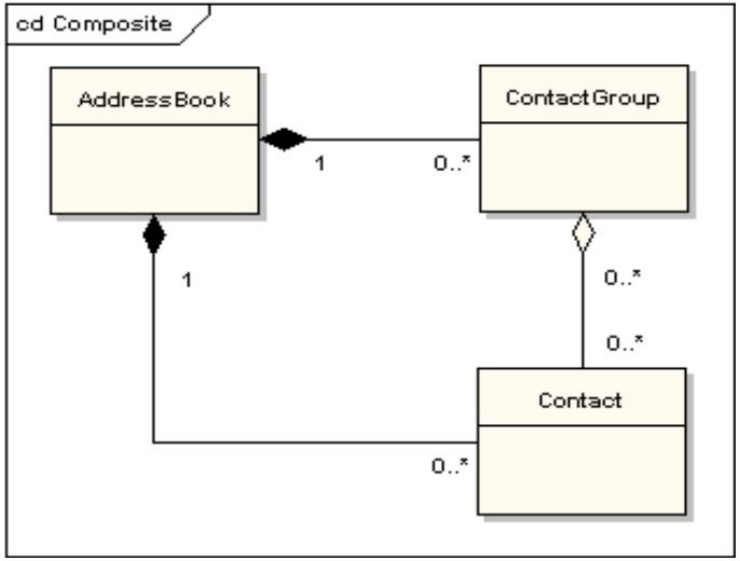


Aggregations

Aggregations are used to depict elements which are made up of smaller components. Aggregation relationships are shown by a white diamond-shaped arrowhead pointing towards the target or parent class.

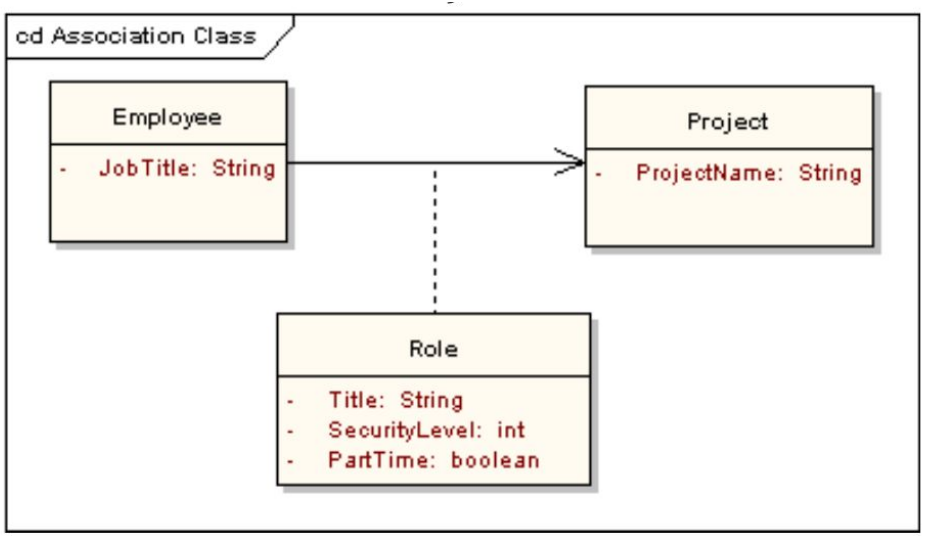
A stronger form of aggregation - a composite aggregation - is shown by a black diamond-shaped arrowhead and is used where components can be included in a maximum of one composition at a time. If the parent of a composite aggregation is deleted, usually all of its parts are deleted with it; however a part can be individually removed from a composition without having to delete the entire composition. Compositions are transitive, asymmetric relationships and can be recursive.

The following diagram illustrates the difference between weak and strong aggregations. An address book is made up of a multiplicity of contacts and contact groups. A contact group is a virtual grouping of contacts; a contact may be included in more than one contact group. If you delete an address book, all the contacts and contact groups will be deleted too; if you delete a contact group, no contacts will be deleted.



Association Classes

An association class is a construct that allows an association connection to have operations and attributes. The following example shows that there is more to allocating an employee to a project than making a simple association link between the two classes: the role that the employee takes up on the project is a complex entity in its own right and contains detail that does not belong in the employee or project class. For example, an employee may be working on several projects at the same time and have different job titles and security levels on each.



Dependencies

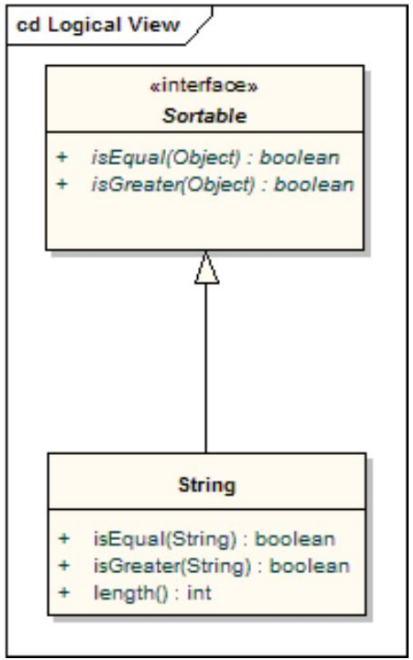
A dependency is used to model a wide range of dependent relationships between model elements. It would normally be used early in the design process where it is known that there is some kind of link between two elements but it is too early to know exactly what the relationship is. Later in the design process, dependencies will be stereotyped (stereotypes available include «instantiate» «trace», «import» and others) or replaced with a more specific type of connector.

Traces

The trace relationship is a specialization of a dependency, linking model elements or sets of elements that represent the same idea across models. Traces are often used to track requirements and model changes. As changes can occur in both directions, the order of this dependency is usually ignored. The relationship's properties can specify the trace mapping, but the trace is usually bi-directional, informal and rarely computable.

Realizations

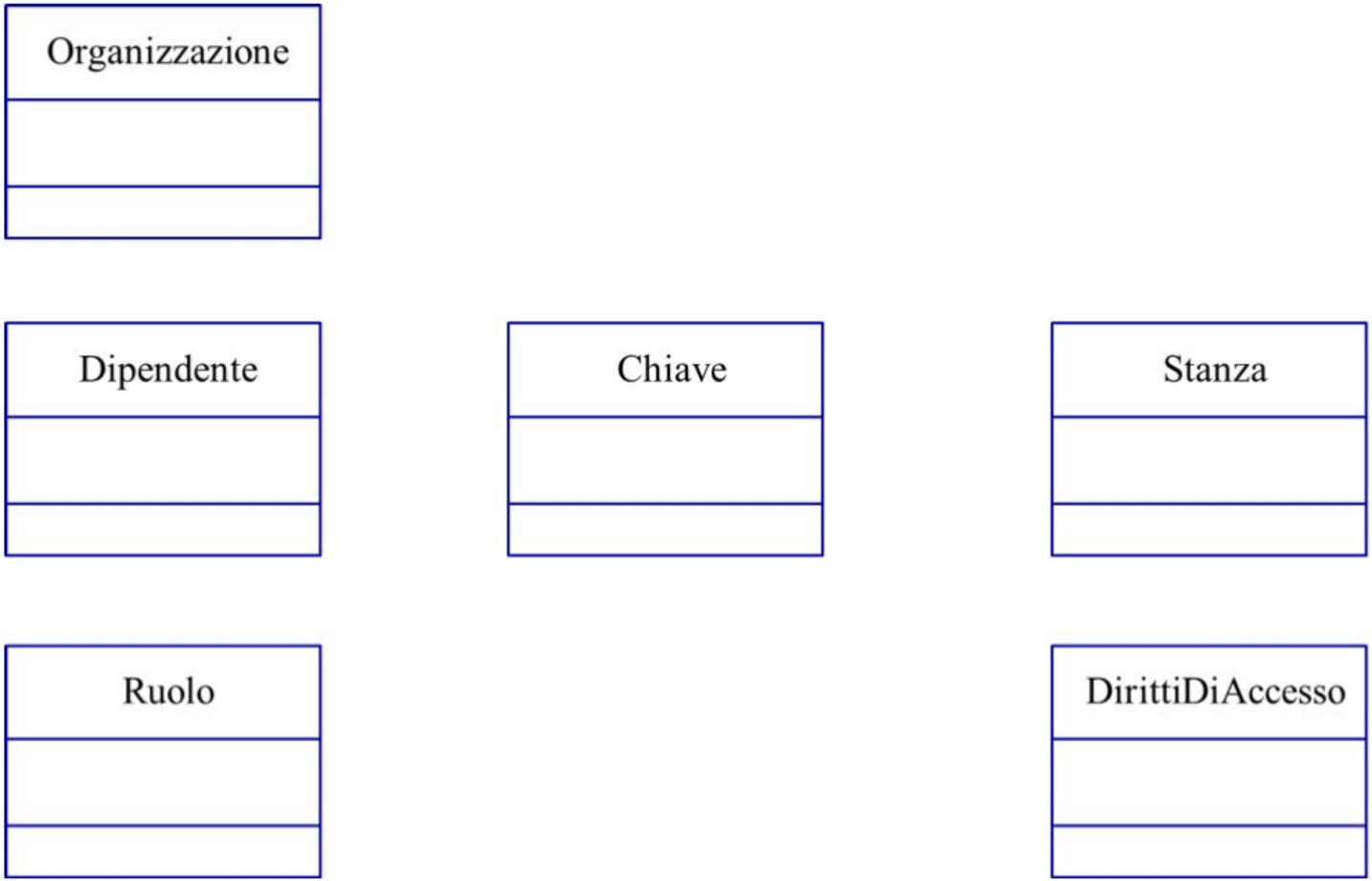
The source object implements or realizes the destination. Realize is used to express traceability and completeness in the model - a business process or requirement is realized by one or more use cases which are in turn realized by some classes, which in turn are realized by a component, etc. Mapping requirements, classes, etc. across the design of your system, up through the levels of modelling abstraction, ensures the big picture of your system remembers and reflects all the little pictures and details that constrain and define it. A realization is shown as a dashed line with a solid arrowhead and the «realize» stereotype.



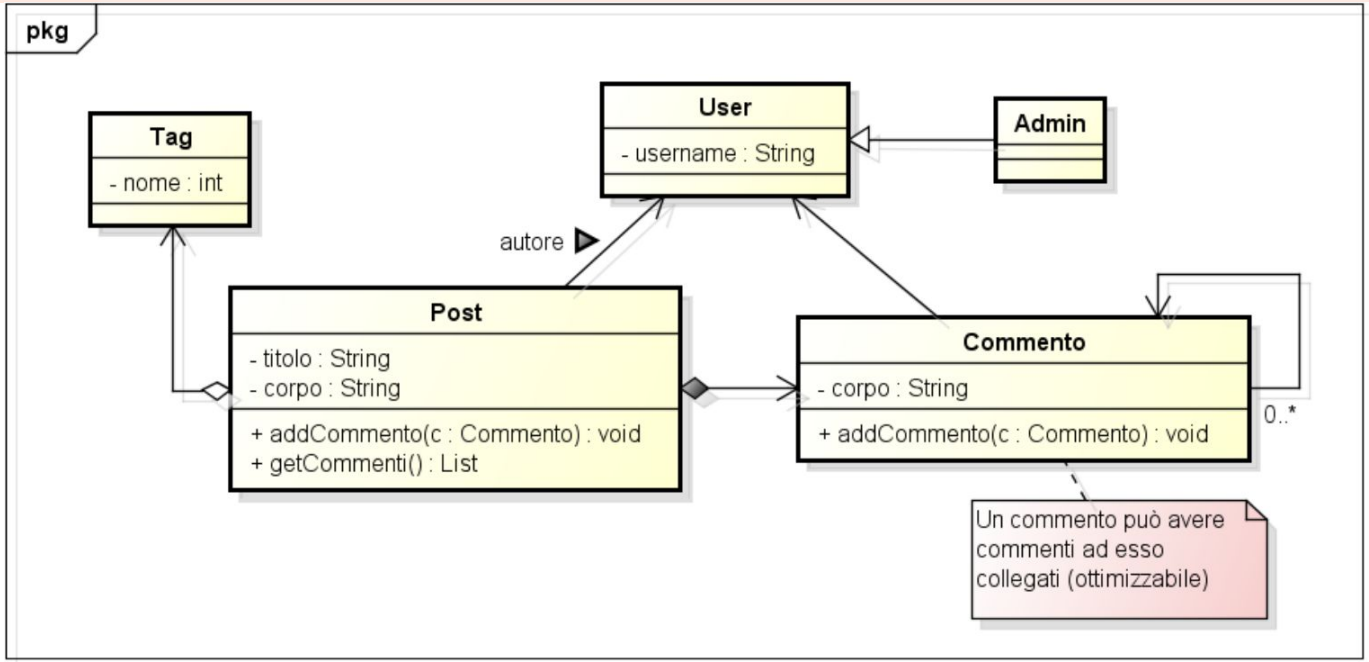
Esempi

Individuazione classi: chiavi magnetiche

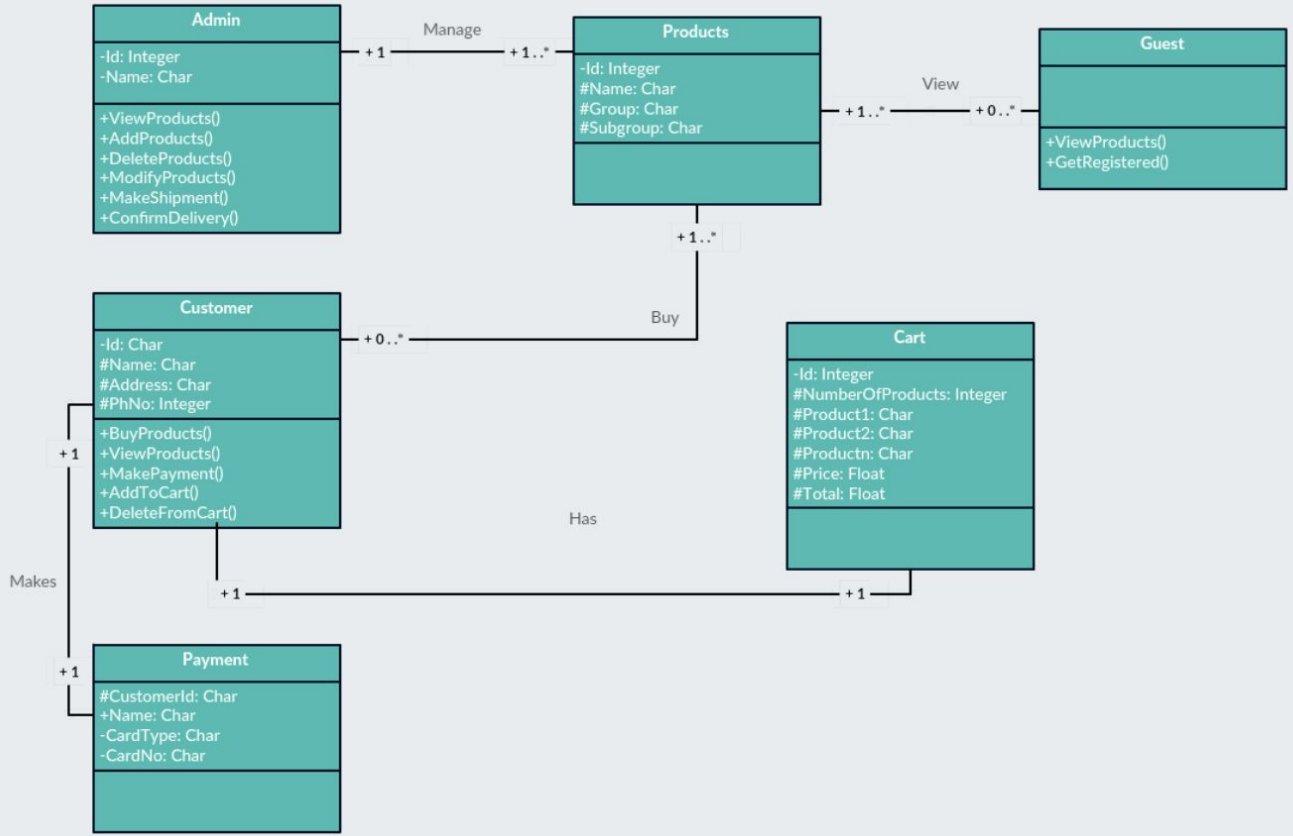
Per motivi di sicurezza, un'organizzazione ha deciso di realizzare un sistema secondo il quale a ogni dipendente è assegnata una chiave magnetica per accedere (aprire) determinate stanze. I diritti di accesso dipenderanno in generale dalla posizione e dalle responsabilità del dipendente. Quindi sono necessarie operazioni per modificare i diritti di accesso posseduti da una chiave se il suo proprietario cambia ruolo nell'organizzazione.

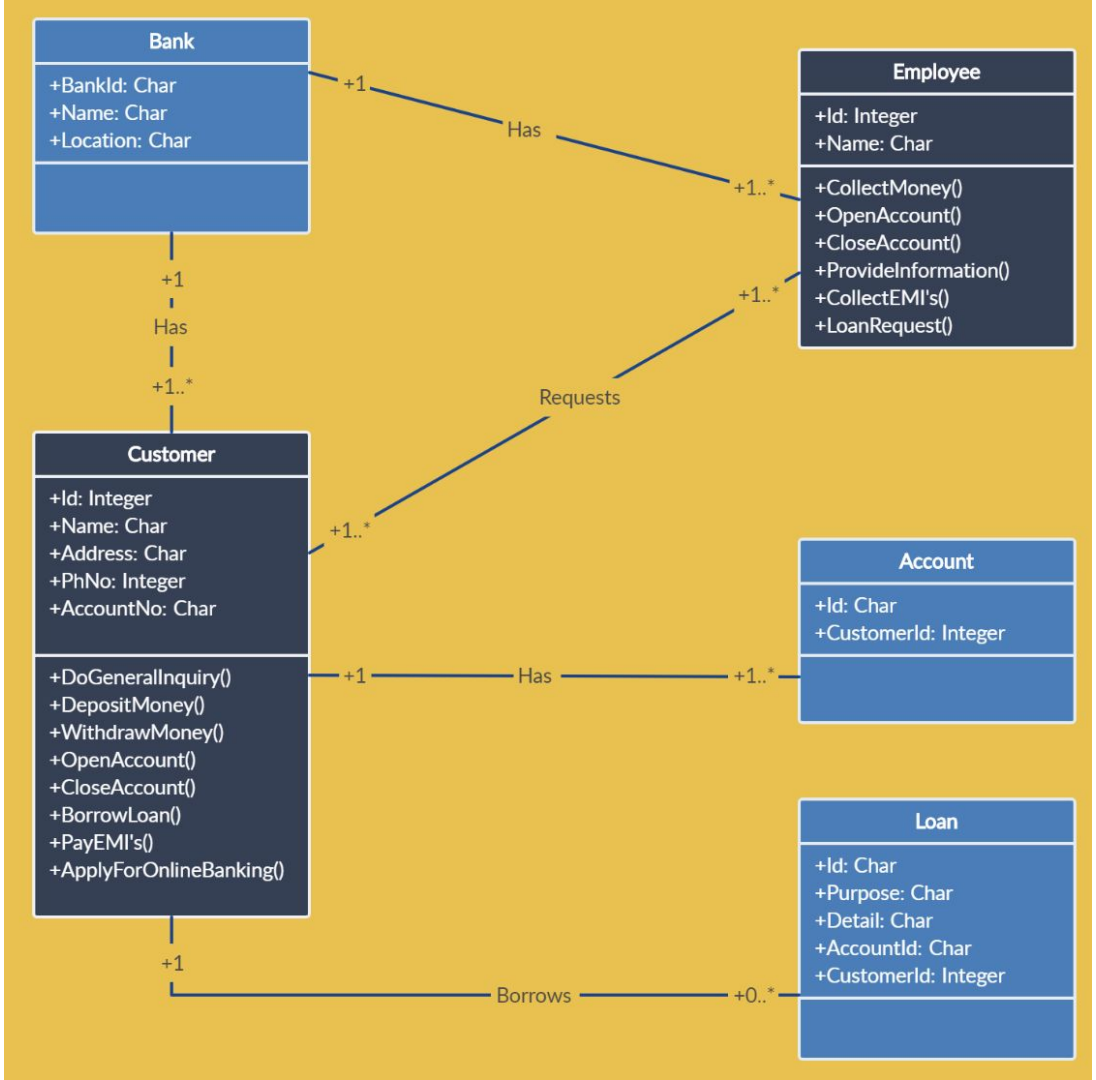

Organizzazione
Dipendente
Chiave
Stanza
Ruolo
DirittiDiAccesso

È richiesto lo sviluppo di un'applicazione che permetta la gestione di un semplice blog. In particolare devono essere disponibili almeno tutte le funzionalità base di un blog: deve essere possibile per un utente inserire un nuovo post e successivamente per gli altri utenti deve essere possibile commentarlo. Queste due operazioni devono essere disponibili unicamente agli utenti registrati all'interno del sistema. La registrazione avviene scegliendo una username e una password. La username deve essere univoca all'interno del sistema.

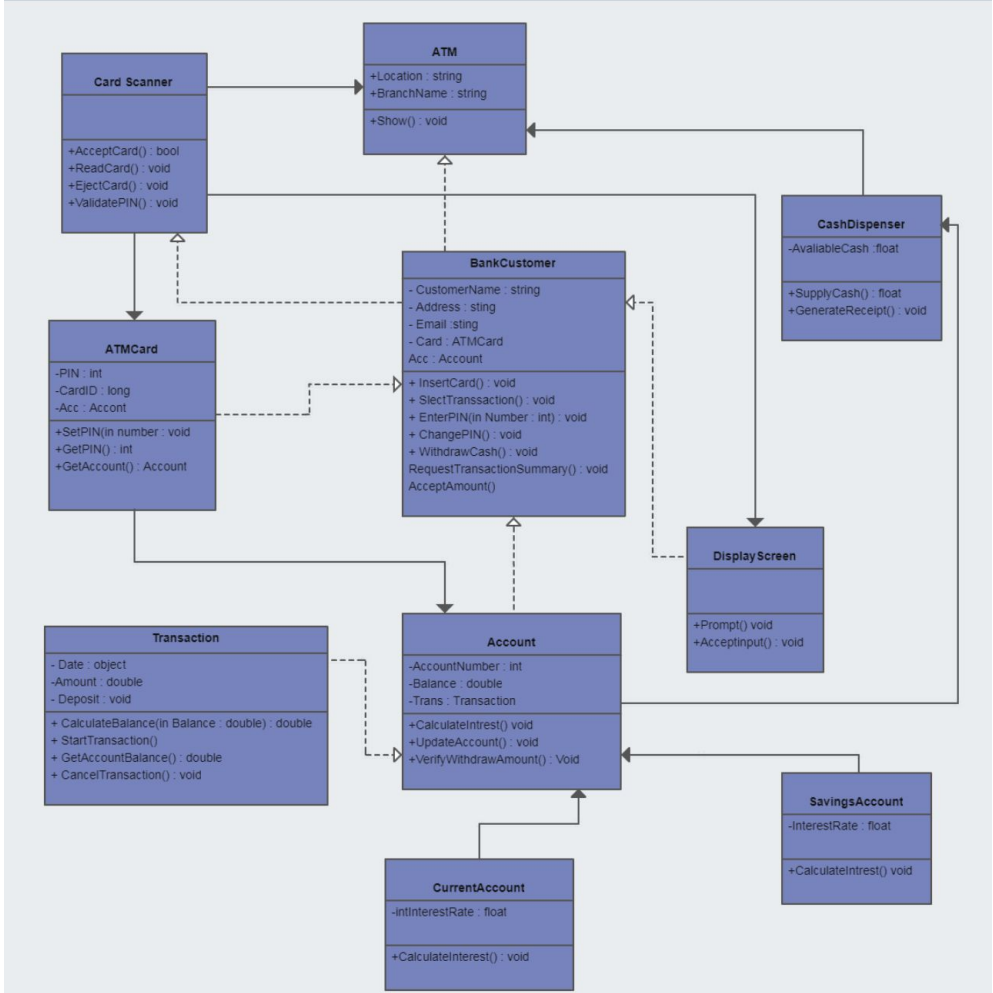


Online Shopping System



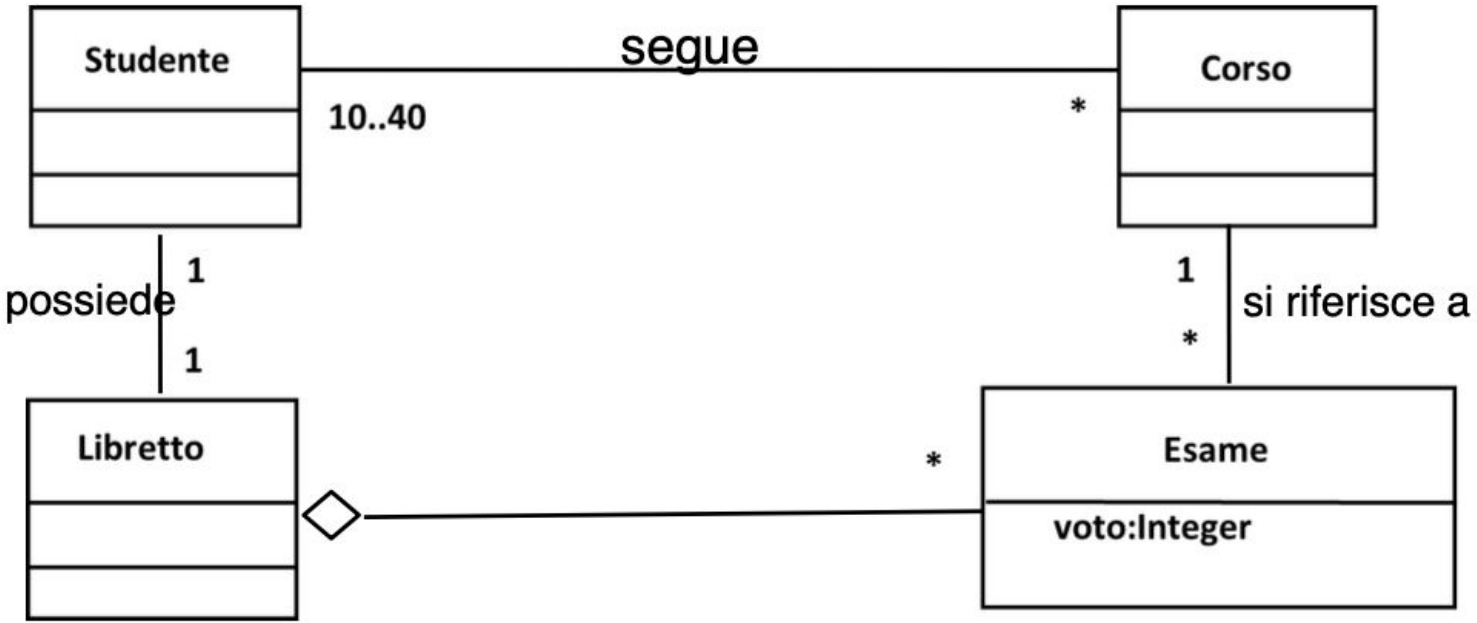

Bank
+BankId: Char
+Name: Char
+Location: Char
+1
Has
+1..*
Employee
+Id: Integer
+Name: Char
+CollectMoney()
+OpenAccount()
+CloseAccount()
+ProvideInformation()
+CollectEMI's()
+LoanRequest()
+1
Has
+1..*
+1..*
Requests
Customer
+Id: Integer
+Name: Char
+Address: Char
+PhNo: Integer
+AccountNo: Char
+DoGeneralInquiry()
+DepositMoney()
+WithdrawMoney()
+OpenAccount()
+CloseAccount()
+BorrowLoan()
+PayEMI's()
+ApplyForOnlineBanking()
+1..*
Account
+Id: Char
+CustomerId: Integer
+1
Has
+1..*
Loan
+Id: Char
+Purpose: Char
+Detail: Char
+AccountId: Char
+CustomerId: Integer
+1
Borrows
+0..*

Class Diagram for ATM System

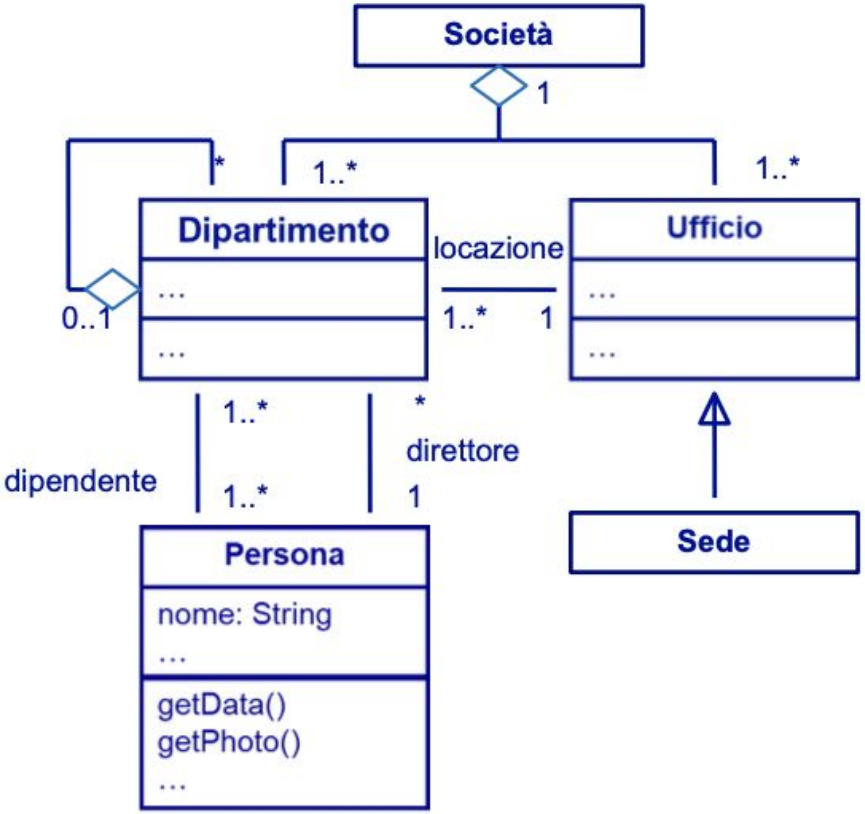


Esempio

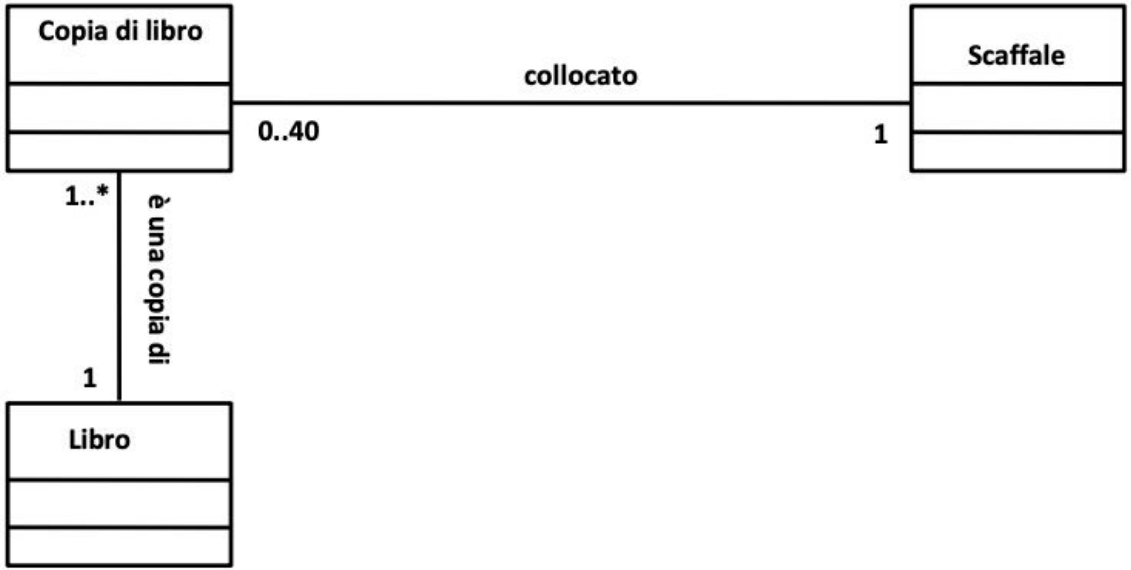
- Studente, Libretto, Esame, Corso



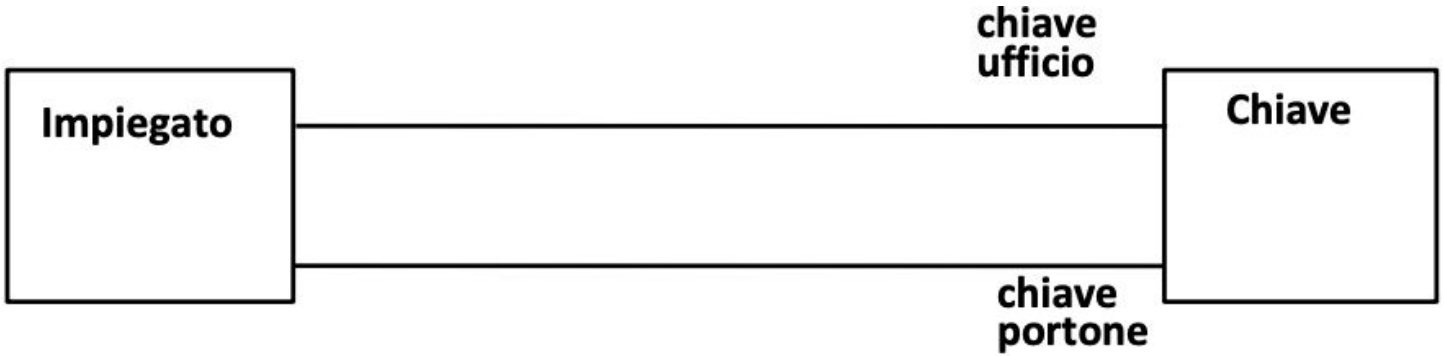
Esempio: classi



Esempio “libreria”



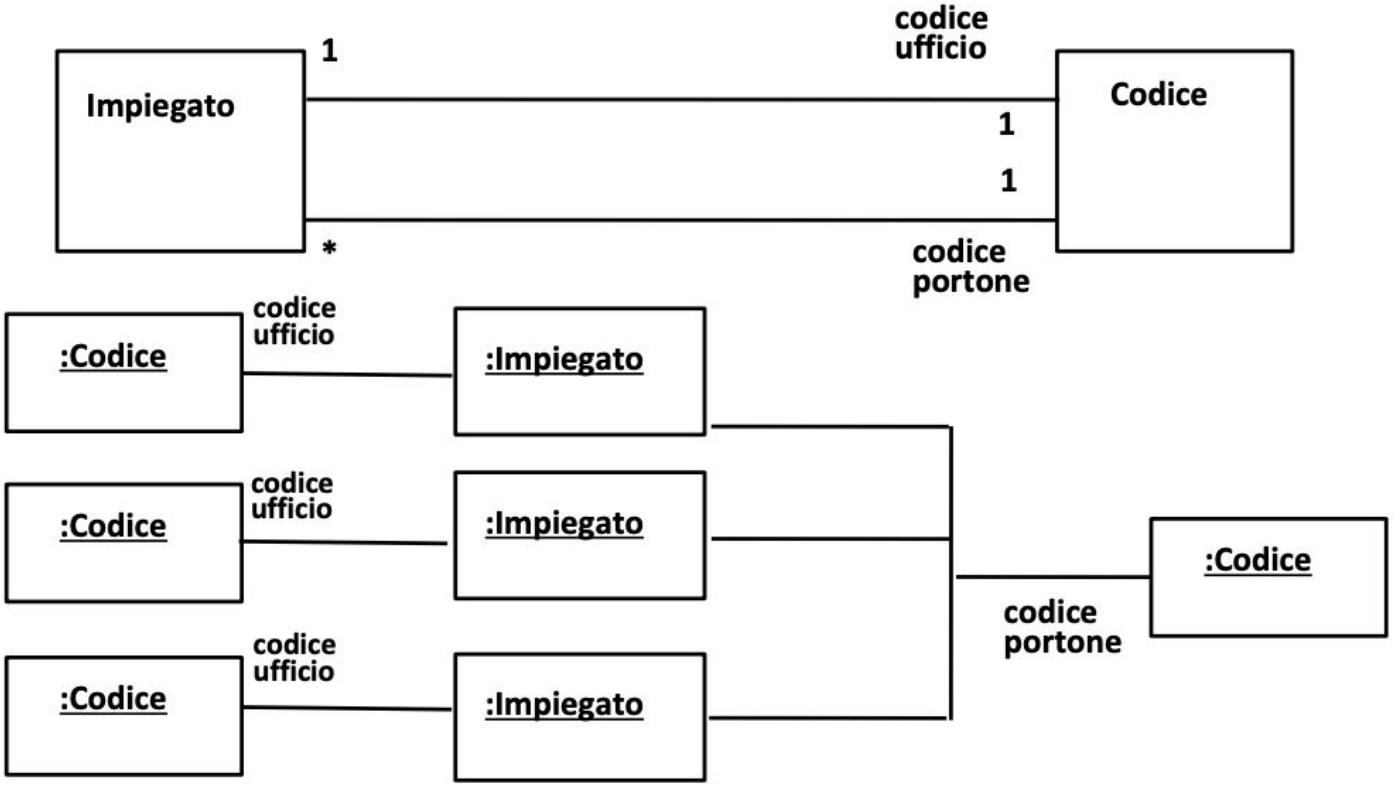
Associazione: caso particolare - ruoli importanti



Individuazione chiavi: chiavi magnetiche

Per motivi di sicurezza, un'organizzazione ha deciso di realizzare un sistema secondo il quale a ogni dipendente è assegnata una chiave magnetica per accedere (aprire) determinate stanze. I diritti di accesso dipenderanno in generale dalla posizione e dalle responsabilità del dipendente. Quindi sono necessarie operazioni per modificare i diritti di accesso posseduti da una chiave se il suo proprietario cambia ruolo nell'organizzazione.

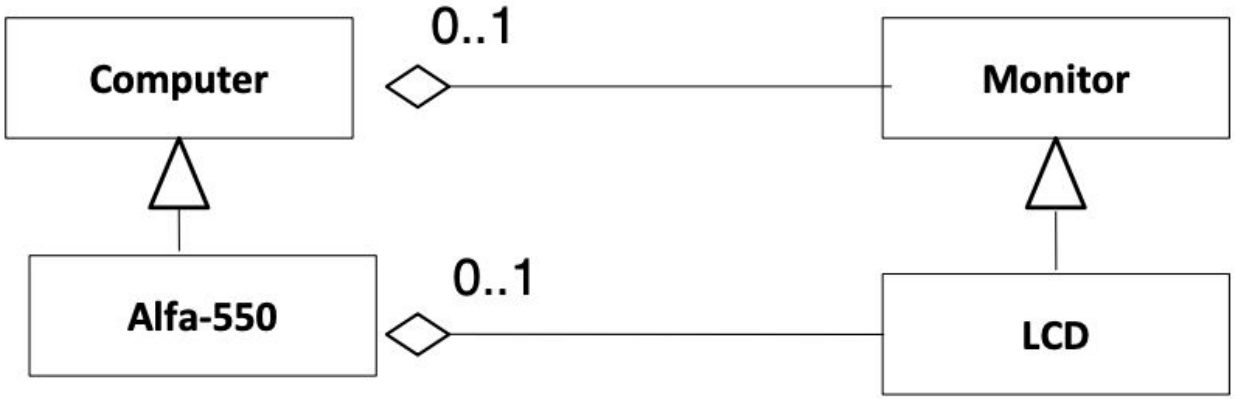
Molteplicità: esempio



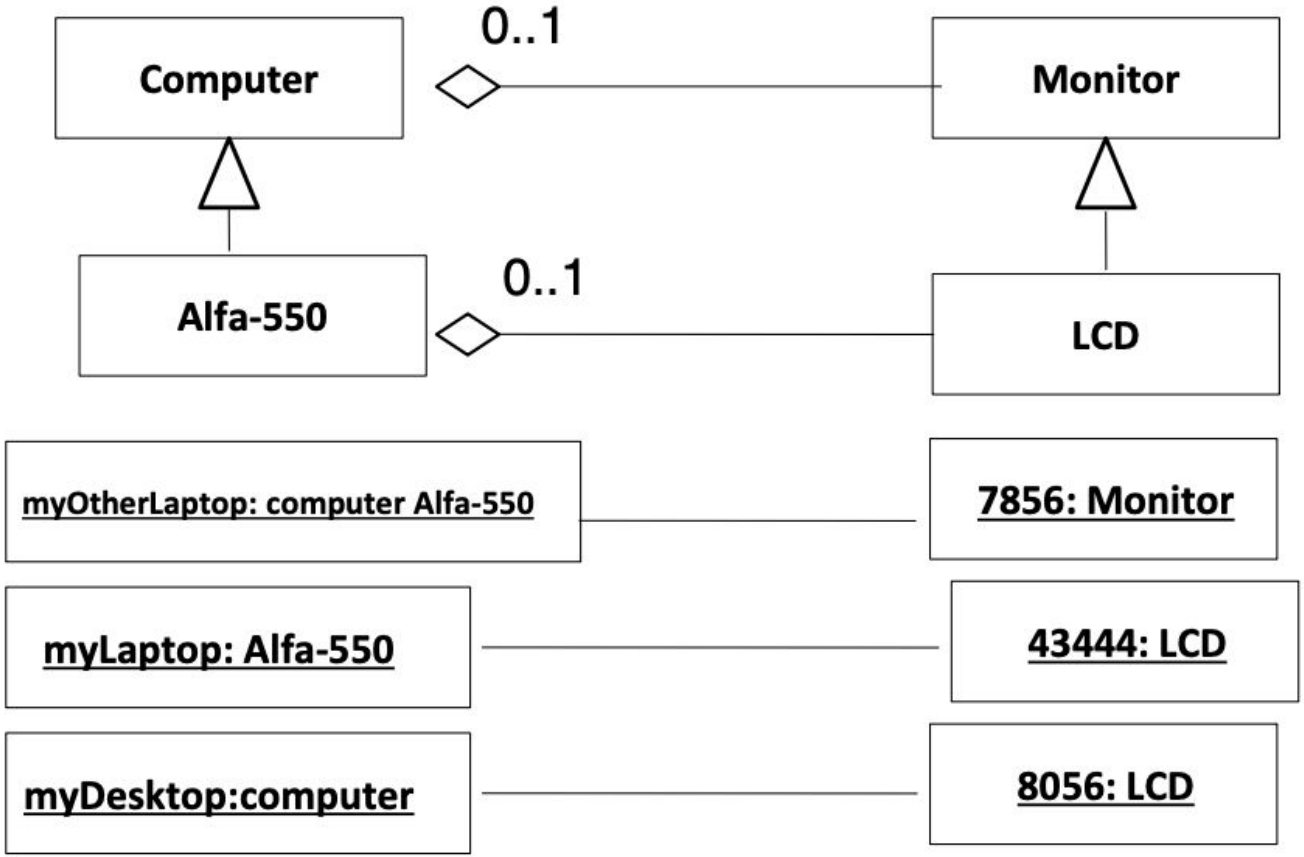
Ruoli e aggregazione: un caso speciale

- Nel diagramma delle classi

Un esempio con generalizzazione e aggregazione

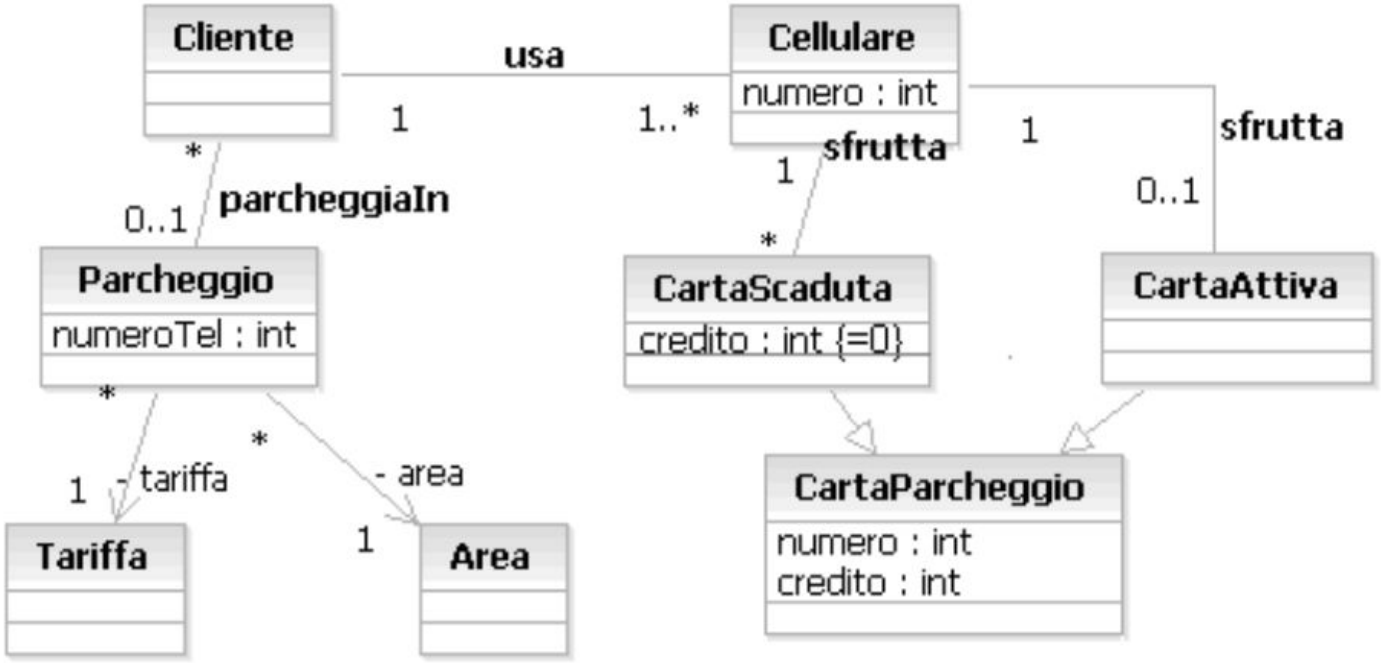


Un esempio con generalizzazione e aggregazione



Easy Park

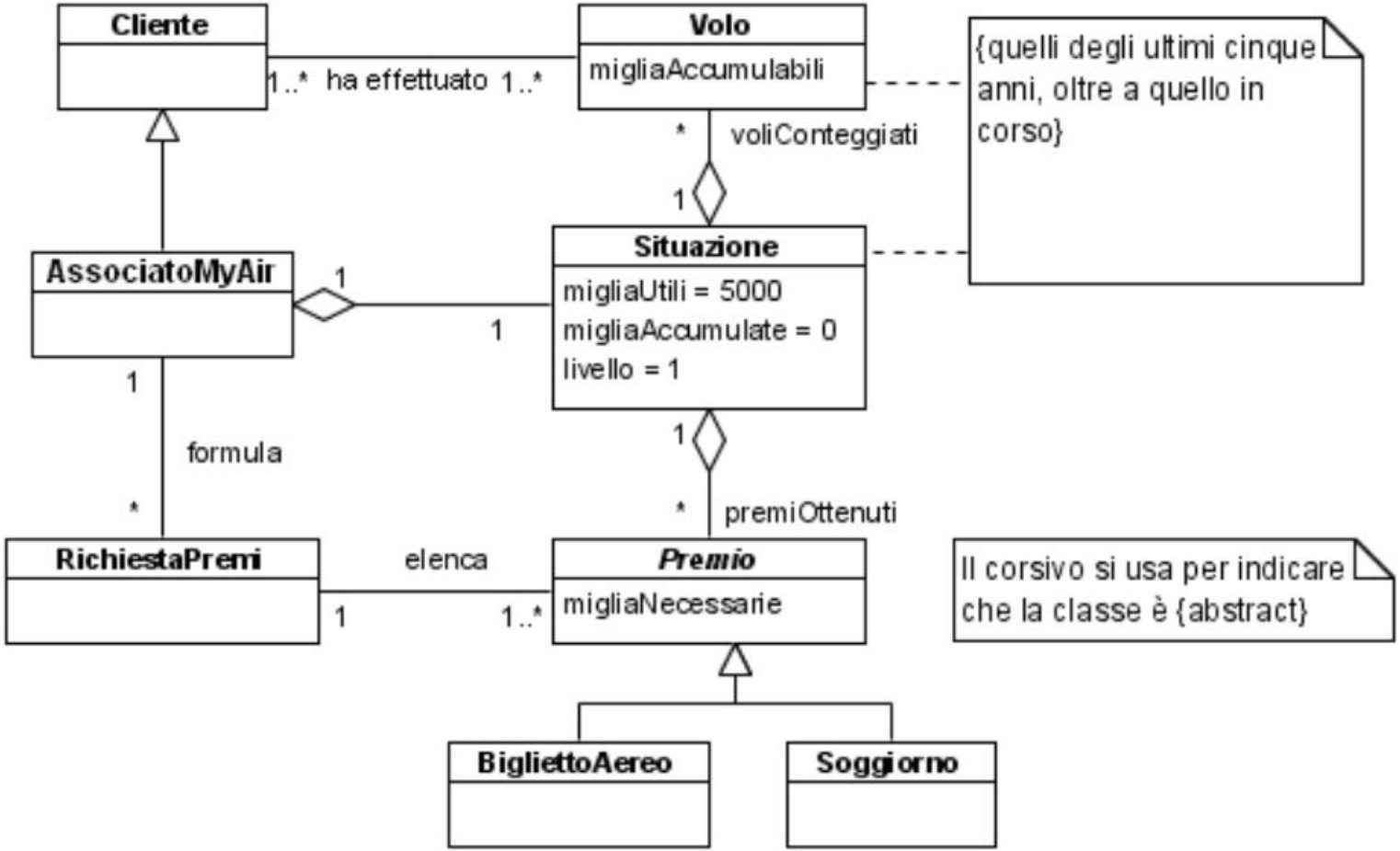
- Soluzione per il pagamento del parcheggio via telefono cellulare.
- 1. Il cliente acquista una Carta Parcheggio prepagata e l'attiva indicando il proprio numero di cellulare. Durante l'attivazione, il sistema trasferisce sulla nuova carta l'eventuale credito residuo su una carta già associata al numero di telefono indicato.
- 2. Il cliente parcheggia ed espone sul cruscotto la Carta Parcheggio. Nel cartellone del Parcheggio verifica qual è il numero di telefono che identifica l'area e la tariffa. Il cliente telefona a questo numero, il cliente è identificato attraverso il proprio numero di telefono cellulare e il sistema attiva il pagamento della sosta.
- 3. Il Controllore controlla l'effettivo pagamento della sosta inserendo il numero della Carta Parcheggio in un applicativo fruibile tramite Pocket PC connesso a internet o Telefono Cellulare.
- 4. Disattivazione della sosta con chiamata via cellulare: l'utente chiama il numero associato al parcheggio, il sistema riconosce l'utente e disattiva il pagamento. Inoltre il sistema comunica vis SMS la disattivazione, la somma pagata, la durata della sosta e il residuo presente sulla Carta Parcheggio.



Modellare anche sosta, con inizio, fine costo (classe associazione)

Esercizio: My Air

- Iscriviti al programma e da semplice cliente diventerai un associato MyAir, guadagnando immediatamente un bonus di 5.000 miglia utili.
- Ogni volta che volerai con MyAir le miglia accumulabili del volo saranno sommate alle tue miglia utili, permettendoti di raggiungere in poco tempo le miglia necessarie per richiedere uno dei nostri premi (omaggio biglietti aereo o soggiorni in località da sogno).
- I premi riscossi danno luogo a una diminuzione immediata delle miglia utili. La situazione è aggiornata il 31 dicembre, mantenendo solo le miglia dei voli effettuati negli ultimi 5 anni.
- Inoltre se accumulerai almeno 15.000 miglia (miglia accumulate) sarai promosso dal livello standard al livello argento. Se invece accumulerai almeno 100.000 miglia entrerai a far parte del ristretto numero di associati del livello oro2.
- Tutte le condizioni si riferiscono esclusivamente alle miglia accumulate in un anno. Il passaggio da un livello all'altro è effettuato il 31 dicembre. La permanenza nel livello da un anno all'altro è soggetta al rispetto degli stessi requisiti per entrare nel livello. Il bonus iniziale non concorre al raggiungimento delle miglia richieste per cambiare o mantenere un livello.


Cliente
Volo
migliaAccumulabili
{quelli degli ultimi cinque anni, oltre a quello in corso}
1..* ha effettuato 1..*
* voliConteggiati
1
AssociatoMyAir
1
1
Situazione
migliaUtili = 5000
migliaAccumulate = 0
livello = 1
1
formula
*
1
* premiOttenuti
RichiestaPremi
elenca
Premio
migliaNecessarie
1
1..*
Il corsivo si usa per indicare che la classe è {abstract}
BigliettoAereo
Soggiorno

Qualche template

- https://app.creately.com/d/wz49SfJCEoo/edit
- https://app.creately.com/d/PITk2j3fNNA/edit
- https://app.creately.com/d/create?templateId=hvc97m5d
- https://app.creately.com/d/create?templateId=ilezbdqz1
- https://app.creately.com/d/create?templateId=hvxe4w5q1
- https://app.creately.com/d/create?templateId=i3dg9g071