UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Diagrammi delle Classi

Basi di Dati e Progettazione del Software
a.a. 2018/19

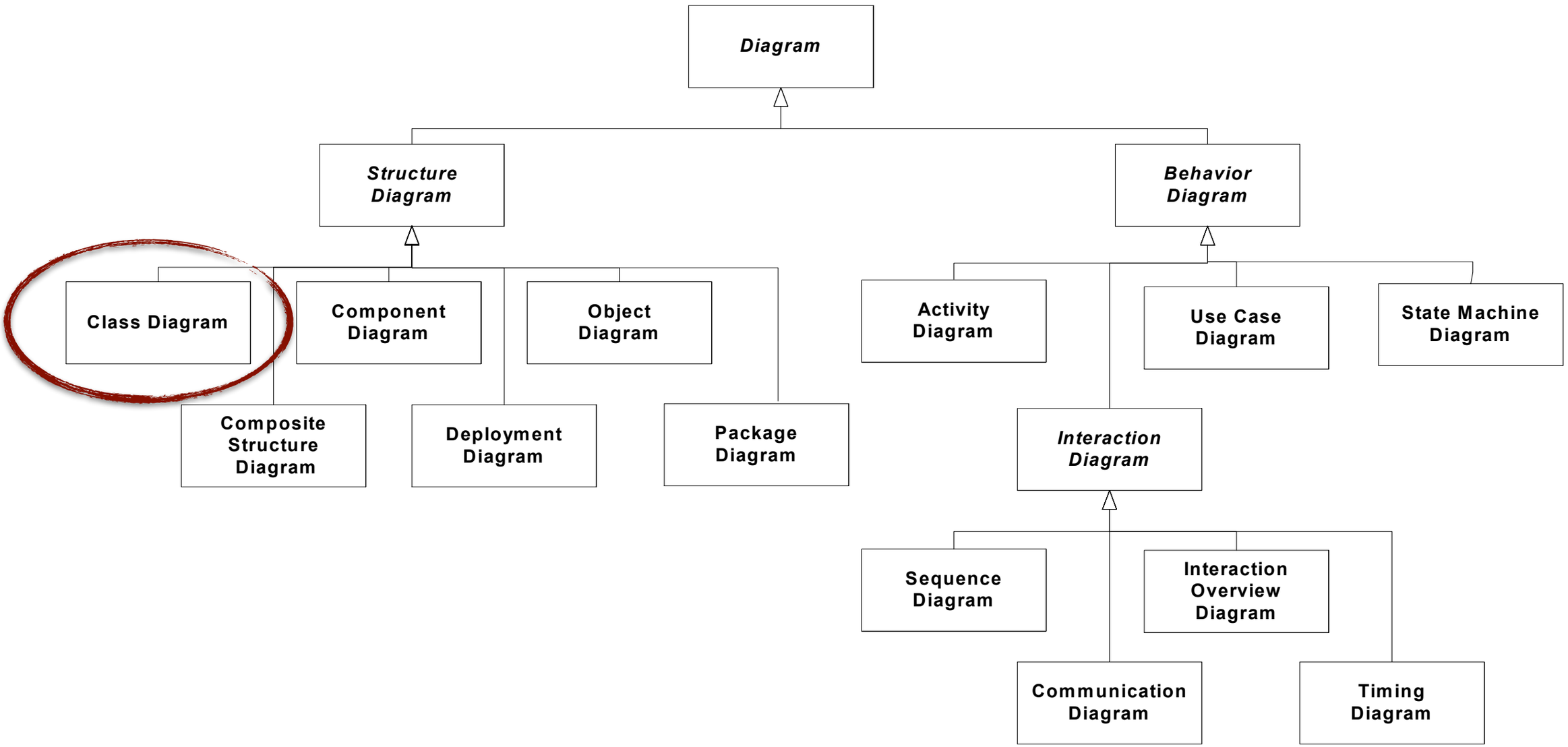
Tassonomia dei diagrammi UML

- UML 2 definisce 13 diagrammi divisi in due categorie
  - structure diagrams: come è fatto il sistema
  - behaviour diagrams: come funziona (o si comporta) il sistema

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

* : non esiste in UML 1.x

Tassonomia dei diagrammi UML 2



Overview

- Il diagramma delle classi è uno dei modelli UML più usati
  - si presta a rappresentare il maggior numero di concetti durante una fase di progettazione
- Descrive:
  - il tipo di oggetti che fanno parte di un sistema
  - le varie tipologie di relazioni statiche tra essi
  - le proprietà e le operazioni di una classe e i vincoli che si applicano ai collegamenti tra oggetti

Proprietà

- Le proprietà rappresentano le caratteristiche strutturali di una classe
- Un unico concetto
- Rappresentato con due notazioni molto diverse
  - Attributi
  - associazioni
- Aspetto grafico molto differente
- Concettualmente sono la stessa cosa

Proprietà: attributi

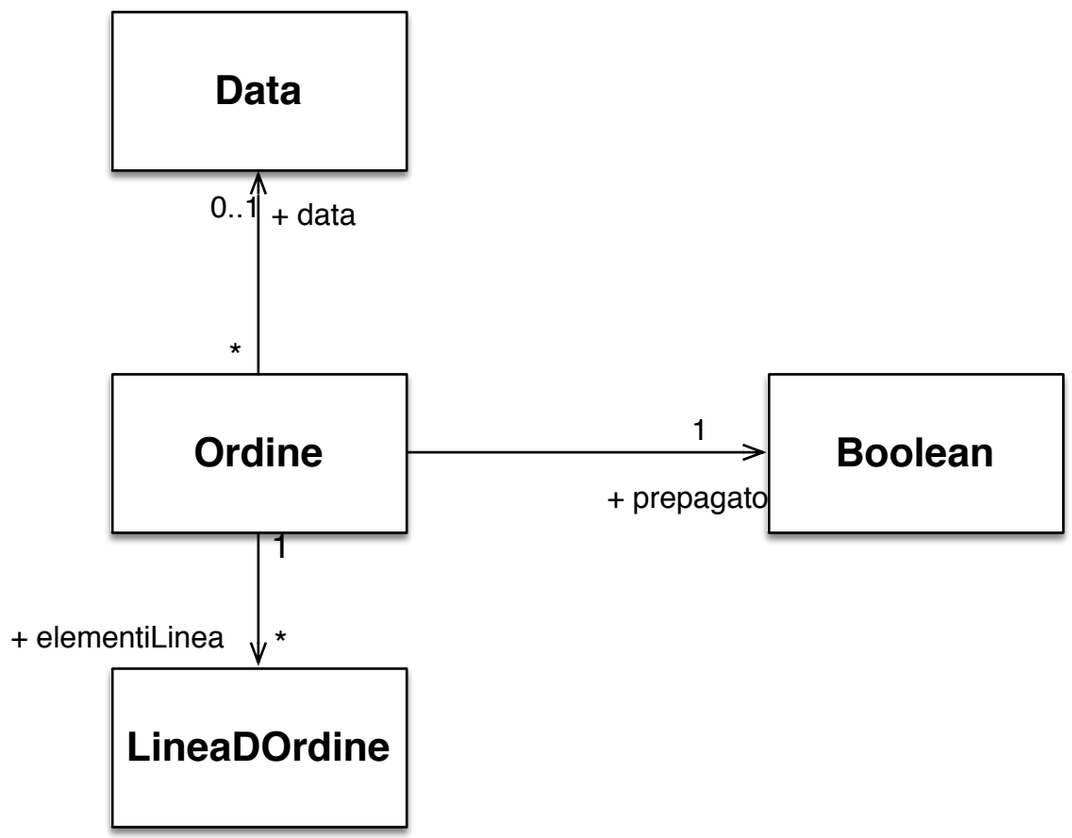
- La notazione per gli attributi descrive una proprietà con una riga di testo all'interno del box della classe
- Esempio:
  - `- titolo: String [1]= “UML distilled”`

Proprietà: associazioni

- Altro modo di rappresentare una proprietà è in forma di associazione

- Gran parte dell'informazione che può essere associata ad un attributo si applica anche alle associazioni

Proprietà espresse come attributi



Proprietà espresse come associazioni

Associazioni e attributi

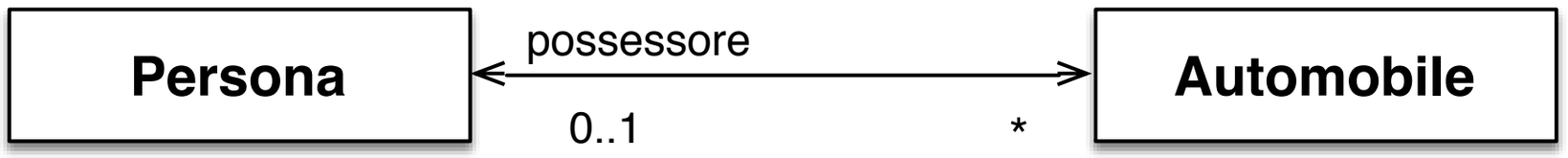
- Due notazioni per rappresentare la stessa cosa
- Quando usare una e quando l'altra?
- In generale gli attributi si usano per "le piccole cose"
  - Date o campi booleani
- Le associazioni sono riservate alle classi più importanti

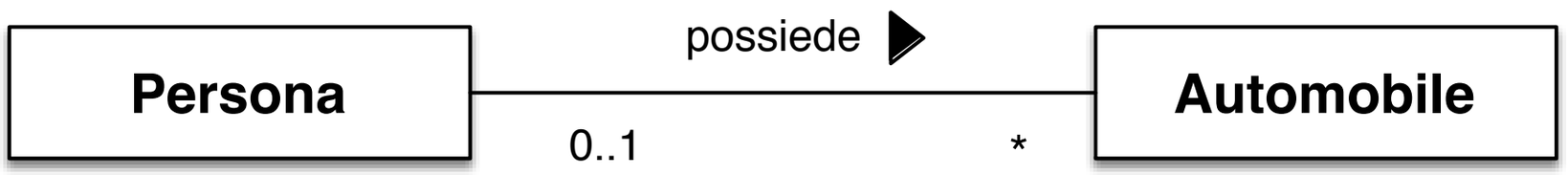
Molteplicità

- La molteplicità di una proprietà è l'indicazione di quanti oggetti possono entrare a farvi parte
- Molteplicità più comuni:
  - 1 - un ordine deve essere fatto da un cliente
  - 0..1 - un'azienda cliente può avere un rappresentante o no
  - * - il numero di ordini va da zero a molti
- In generale le molteplicità si possono definire usando due estremi
- se i due estremi coincidono si può usare un solo numero
- * È un'abbreviazione di 0..*

Associazioni bidirezionali

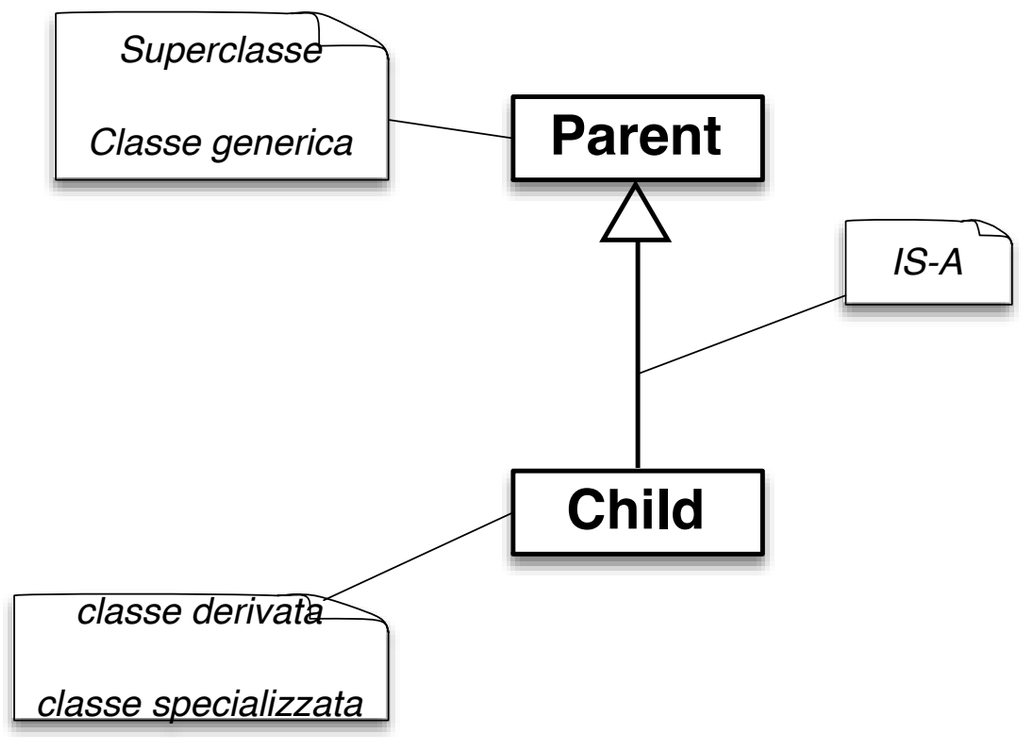
- Finora associazioni unidirezionali
- Associazione bidirezionale:
  - È costituita da una coppia di proprietà collegate delle quali una è l' inverso dell'altra
- Esempio:
  - La classe **`Automobile`** ha la proprietà **`proprietario`**: **`Persona[1]`**
  - La classe **`Persona`** ha la proprietà **`automobili: Automobile[*]`**


Persona
possessore
0..1
*
Automobile


Persona
possiede
0..1
*
Automobile

Generalizzazione

- Specializzazione/ereditarietà: una relazione tassonomica tra un elemento più generale ed uno più specifico



Aggregazione e composizione

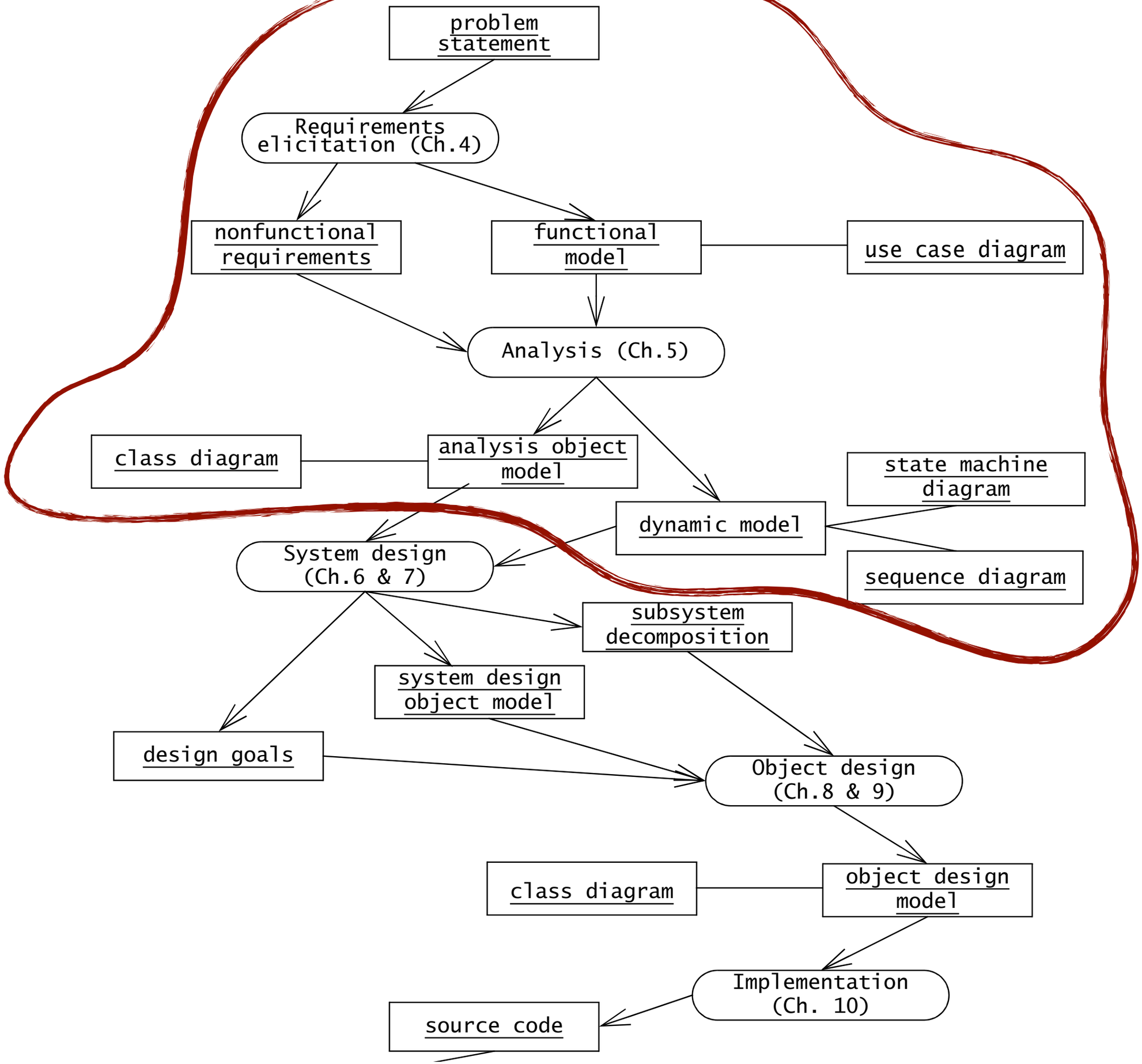
- Aggregazione: è un caso speciale di composizione e rappresenta la gerarchia di “parte-di”
- Associazione: una forma forte di aggregazione dove la vita dell’elemento componente è controllata dall’elemento aggregante, le parti non esistono se non esiste il contenitore

Usiamo i Class Diagrams

Modellare le relazioni tra gli oggetti

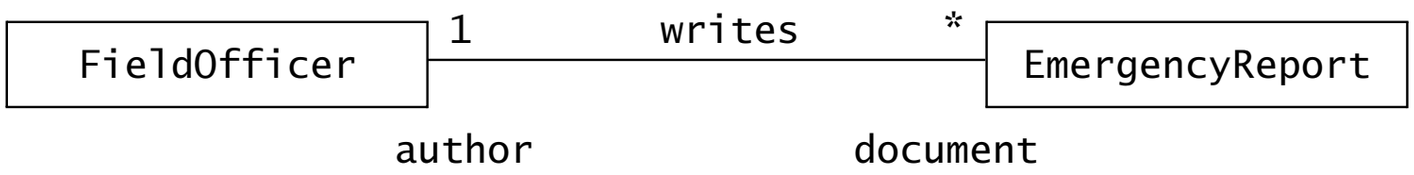
Modellare le relazioni tra gli oggetti

- Finora abbiamo:
  - identificato i requisiti non funzionali
  - identificato i requisiti funzionali e li abbiamo rappresentati attraverso diagrammi dei casi d'uso - *functional model*
  - abbiamo documentato il flusso degli eventi nei casi d'uso
  - abbiamo identificato di oggetti **tramite l'euristica di Abbott**:
    - entity
    - boundary
    - control
  - abbiamo legato i casi d'uso agli oggetti tramite i sequence diagrams - *dynamic model*
- …. adesso modelliamo le relazioni tra gli oggetti —> ***analysis object model***


problem statement
Requirements elicitation (Ch.4)
nonfunctional requirements
functional model
use case diagram
Analysis (Ch.5)
class diagram
analysis object model
state machine diagram
dynamic model
sequence diagram
System design (Ch.6 & 7)
subsystem decomposition
system design object model
design goals
Object design (Ch.8 & 9)
class diagram
object design model
Implementation (Ch. 10)
source code

Identificare le associazioni

- Una relazione tra due o più classi
- Identificare le associazioni ha due vantaggi
  - Chiarisce il modello di analisi rendendo esplicita la relazione tra gli oggetti
  - Consente allo sviluppatore di identificare casi limite associati alle relazioni
- Per esempio: è intuitivo assumere che l' **`EmergencyReports`** è scritto da un **`FieldOfficer`**, ma questo report può essere scritto da più **`FieldOfficer`**? Il sistema deve permettere **`EmergencyReport`** anonimi?

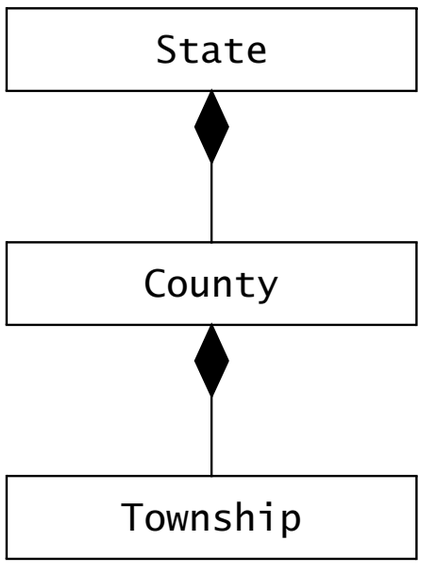


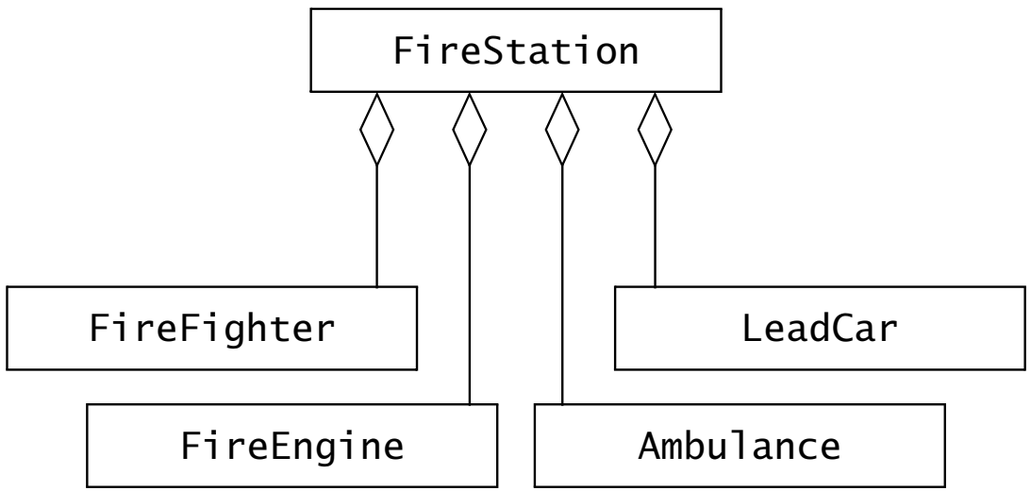
Identificare le associazioni

- Inizialmente le associazioni tra oggetti entità sono le più importanti
  - Rivelano più informazioni sul dominio di applicazione
- Secondo l'euristica di Abbott
  - identificare verbi e frasi particolari contenenti verbi
    - has, is part of, manages, reports to, is triggered by, is contained in, talks to, includes
- ad ogni associazione deve essere dato un nome ed un ruolo

Identificare gli aggregati

- Le associazioni di composizione e aggregazione sono usate per rappresentare i concetti di *tutto-parte (whole-part)*
- Aggregazione e associazione aggiungono informazioni a modello
- Le aggregazioni sono spesso usate nelle interfacce utente





Identificare gli attributi

- Gli attributi sono le proprietà del singolo oggetto
  - per esempio: **`EmergencyReport`** ha un emergencyType, una location, ed una descrizione
- Le proprietà che sono rappresentate da altri oggetti non sono attributi
  - per esempio: **`EmergencyReport`** ha un autore che è rappresentato da una classe **`FieldOfficer`**

EmergencyReport
emergencyType:{fire,traffic,other} location:String description:String

Documentazione

RAD

Requirements Analysis Document

Requirements Analysis Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - *3.4.3 Object model*
     - *3.4.4 Dynamic model*
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary