Luca Cabibbo

Analisi e Progettazione del Software

Analisi e progettazione orientata agli oggetti

La creatività non è un fungo
che spunta dal terreno, da solo.
Semmai, è il frutto
di una lunga preparazione.

Graham Wallas

Capitolo 1

marzo 2024

1.1 Contenuto del corso e sua utilità

Questo corso è basato sul libro di **Craig Larman**

- **Applicare UML e i pattern**
  analisi e progettazione orientata agli oggetti
  - quinta edizione, 2020
  - Pearson Education Italia, ISBN 978-8891904591
- è un'introduzione all'analisi e alla progettazione orientata agli oggetti (OOA/D) basata sull'*applicazione* (nel senso di "mettere in pratica") di
  - UML – per la modellazione del software
  - pattern – principi di analisi e progettazione
  - un processo iterativo – come contesto per lo sviluppo del software

Analisi e progettazione a oggetti

Sviluppare software è divertente – ma sviluppare software di qualità è un'attività complessa

- un sistema software deve
  - implementare un insieme di funzionalità
  - fornire alcune qualità
- come dominare questa complessità?

Alcuni problemi pragmatici nello sviluppo di software OO

- quali gli oggetti? quali le classi?
- che cosa deve conoscere ciascun oggetto? che deve saper fare?
- come collaborano gli oggetti?

Analisi e progettazione a oggetti

Lo sviluppo di software OO può essere sostenuto dalle seguenti attività

- analisi dei requisiti (non è OO)
- analisi orientata agli oggetti (OOA)
- progettazione orientata agli oggetti (OOD)

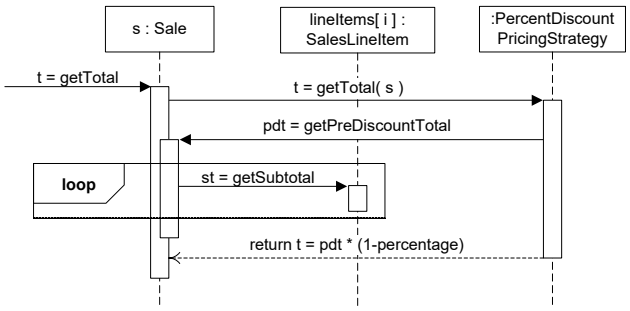
- queste attività possono essere guidate dall'applicazione di opportuni principi, pattern e linee guida

- **l'analisi e la progettazione del software sono attività altamente creative – ma, fortunatamente, le sue fondamenta possono essere studiate e apprese**

Modelli e modellazione

La modellazione è un'attività fondamentale nello sviluppo del software

- un **modello** è una semplificazione della realtà che descrive completamente un sistema da un particolare punto di vista



Gestisci Restituzione
Scenario principale di successo:
Un cliente arriva alla cassa con alcuni articoli da restituire. Il cassiere usa il sistema POS per registrare ciascun articolo restituito …
Scenari alternativi:
Se il cliente aveva pagato con la carta di credito, ...
Se ...

Modelli e modellazione

La modellazione è un'attività fondamentale nello sviluppo del software

- un **modello** è una semplificazione della realtà che descrive completamente un sistema da un particolare punto di vista
- la modellazione è importante perché può favorire i ragionamenti e la comunicazione

- **l'importanza di un modello non è in un particolare diagramma, ma è nell'idea che il diagramma intende comunicare**

Applicare UML

Unified Modeling Language (**UML**)

- una notazione visuale standard per creare modelli – relativi soprattutto al software OO

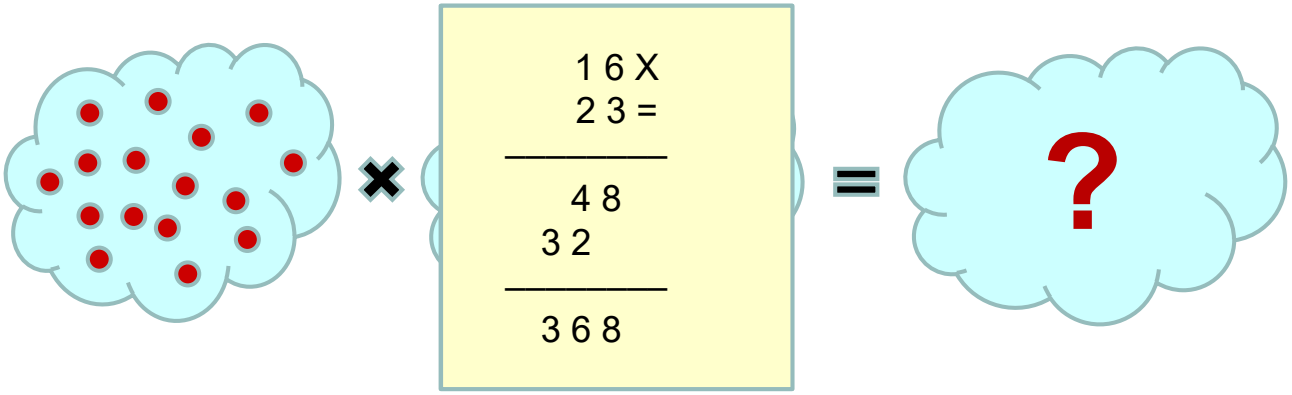
Nella modellazione, una notazione è utile – ma è molto più importante saper **pensare a oggetti**

- non è utile conoscere UML, se non si sa come ideare un buon progetto OO
- nella modellazione sono importanti le idee e poter comunicare le idee

Sull'importanza di una buona notazione

Il matematico Whitehead nel 1911 osservava

- liberando il cervello da tutto il lavoro non necessario, una buona notazione ci lascia liberi di concentrarci su problemi più avanzati – e "incrementa" il nostro "potere mentale"



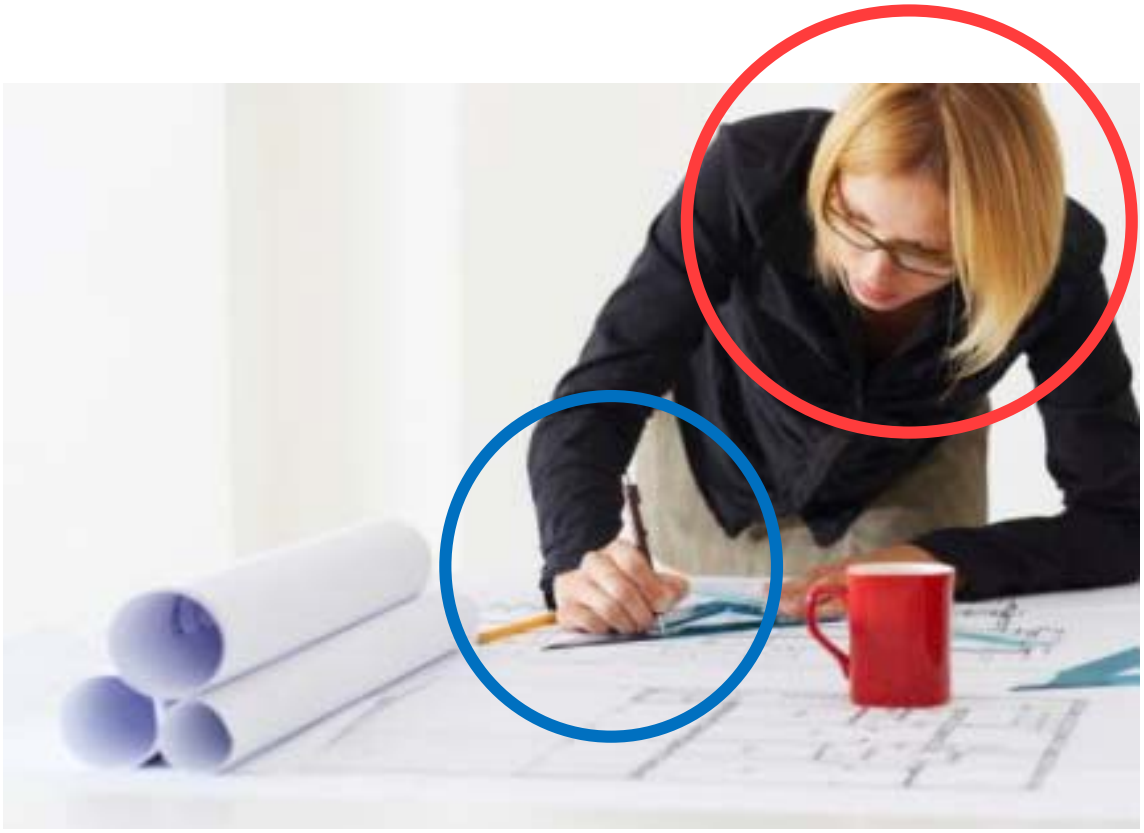
- una buona notazione è indispensabile
- ma è molto più importante è sapere cosa farci e come utilizzarla

Applicare UML

I segreti della modellazione

- lo *scopo primario della modellazione* è **comprendere** – non documentare
- il *valore primario della modellazione* è nella **discussione durante la modellazione** – noi modelliamo per poter conversare
- **UML può essere applicato per descrivere idee, ragionamenti e scelte, di analisi e progettazione**

Che cosa è importante?



Sviluppo di software OO

Sviluppo di software OO

- oggetti e classi
- ciascun oggetto conosce delle informazioni
- ciascun oggetto sa eseguire delle operazioni

Problemi nello sviluppo di software OO

- quali sono gli oggetti e le classi?
- che cosa deve conoscere ciascun oggetto/classe?
- che cosa deve fare ciascun oggetto/classe?
- come **collaborano** gli oggetti?
- in generale, come vanno allocate le **responsabilità** (operazioni e dati) agli oggetti?

Progettazione a oggetti: principi e pattern

Lo sviluppo di software OO può essere basato sull'applicazione di opportuni principi e pattern – in questo corso

- **progettazione guidata dalle responsabilità**
  - una responsabilità è un impegno a eseguire un compito o di conoscere un'informazione
- mediante l'**applicazione** di **pattern**
  - un pattern è una soluzione esemplare a un problema di progettazione ricorrente

Studi di caso

Il corso illustra l'applicazione di tutti gli strumenti insegnati con riferimento ad alcuni studi di caso

- gli studi di caso del libro
- altri studi di caso

Non solo OOA/D

L'enfasi del corso è sull'OOA/D

- viene presentata anche l'attività preliminare dell'**analisi dei requisiti** – *casi d'uso* e *storie utente*
- viene anche studiata l'attività correlata di **trasformazione dei progetti in codice** – *Object-Oriented Programming* (*OOP*)

Sviluppo iterativo e modellazione agile

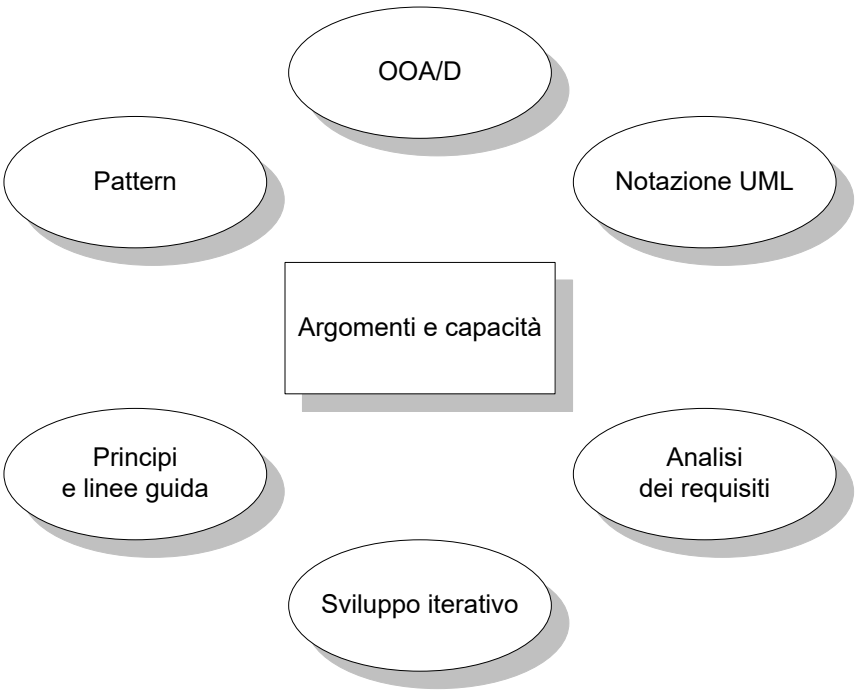
È utile descrivere le attività di analisi e progettazione nel contesto di un **processo per lo sviluppo del software**

- questo contesto ci aiuta a capire la relazione tra le diverse attività dello sviluppo del software
- facciamo riferimento a processi di sviluppo **iterativi** e **agili** – *Unified Process* (*UP*) e *Scrum*
- i concetti presentati sono rilevanti anche per altri processi e approcci

In sintesi

Obiettivo del corso è sostenere lo sviluppo di software di qualità

- che cosa serve per sviluppare software di qualità?



1.2 L'obiettivo di apprendimento principale

Qual è la singola capacità più importante nell'OOA/D?

- **una capacità critica nello sviluppo OO è quella di assegnare in modo abile responsabilità agli oggetti software**

In questo corso, l'OOD è fortemente basata su principi per l'assegnazione di **responsabilità**, codificati come pattern

- *pattern GRASP* e *design pattern GoF*

1.3 Che cosa sono analisi e progettazione

L'**analisi** enfatizza un'**investigazione** di un problema e dei suoi requisiti – ***che cosa***

- non è interessata direttamente alle soluzioni del problema

La **progettazione** enfatizza una **soluzione concettuale** che soddisfa i requisiti del problema – ***come***

- non è interessata direttamente alla realizzazione (implementazione) della soluzione

Analisi e progettazione hanno obiettivi diversi, perseguiti in modo diverso – tuttavia sono attività fortemente sinergiche e inseparabili

1.4 Che cosa sono OOA e OOD

L'**analisi orientata agli oggetti** (**OOA**, **Object-Oriented Analysis**) si basa principalmente sull'identificazione dei *concetti nel dominio del problema* – e su una loro descrizione a oggetti

- ad es., in un sistema per la gestione di voli, **Aereo**, **Volo** e **Pilota**

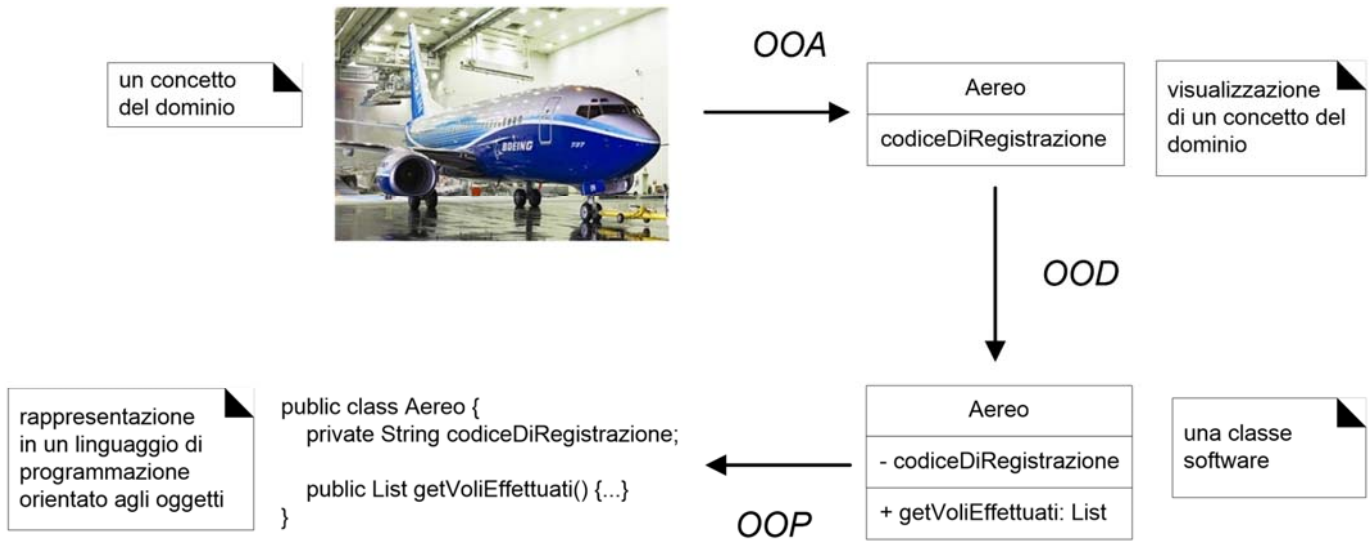
La **progettazione orientata agli oggetti** (**OOD**, **Object-Oriented Design**) si basa principalmente sulla specifica di una comunità di *oggetti software che collaborano* per soddisfare i requisiti

- ad es., un oggetto software **Aereo** che ha un attributo **codiceDiRegistrazione** e un metodo **getVoliEffettuati**

Le classi scelte durante l'OOD vengono implementate durante la **programmazione orientata agli oggetti** (**OOP**, **Object-Oriented Programming**)

- ad es., una classe **Aereo** in Java

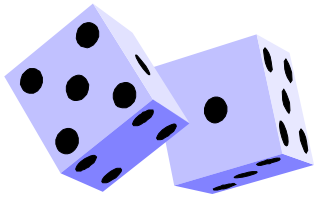
OOA ⇒ OOD ⇒ OOP



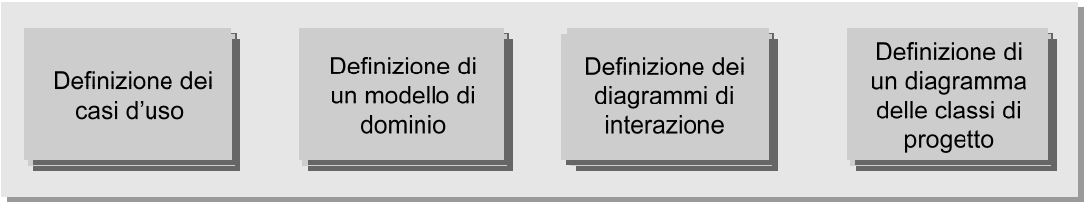
1.5 Un breve esempio

Gioca una Partita a Dadi

- un giocatore tira due dadi
- se il totale è sette, ha vinto
- altrimenti, ha perso



Attività



Definizione dei casi d'uso

Analisi dei requisiti

- un **caso d'uso** descrive una modalità di uso del sistema

Caso d'uso per ***Gioca una Partita a Dadi***

- il Giocatore chiede di lanciare i dadi
- il Sistema presenta il risultato: se il valore totale delle facce è sette, il giocatore ha vinto, altrimenti ha perso

Definizione di un modello di dominio

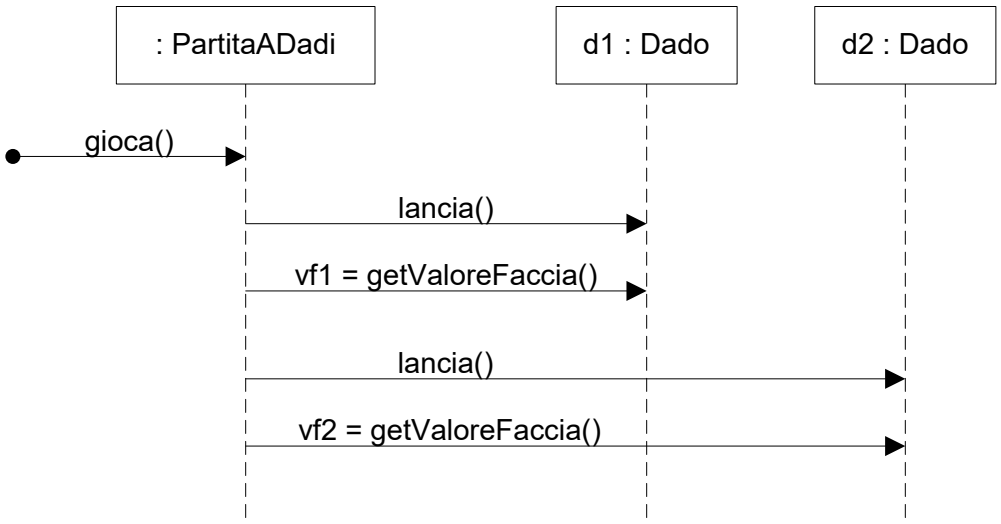
OOA – descrivere il dominio di interesse sulla base di un modello concettuale

- un **modello di dominio** rappresenta i concetti, le associazioni e gli attributi significativi del dominio di interesse

Definizione dei diagrammi di interazione

OOD – definire gli oggetti software e le loro collaborazioni (aspetti dinamici)

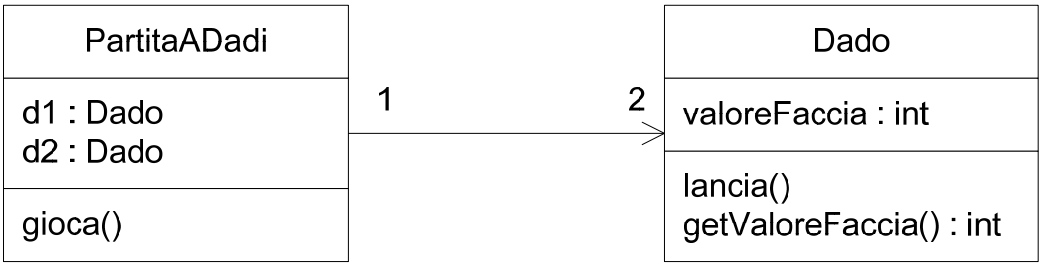
- un **diagramma di interazione** mostra le collaborazioni tra alcuni oggetti software
- descrive scelte progettuali circa l'assegnazione di responsabilità



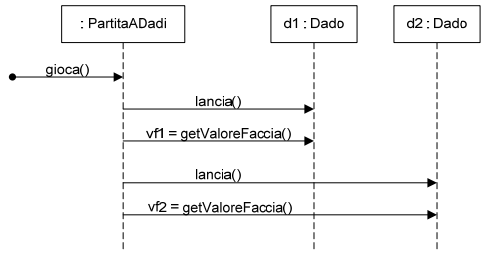
Diagrammi delle classi di progetto

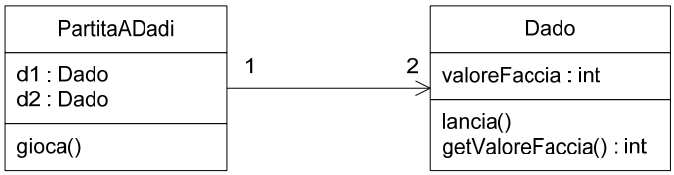
OOD – definire gli oggetti software e le loro collaborazioni (aspetti statici)

- un **diagramma delle classi di progetto** è una descrizione statica della struttura delle classi software, con i loro attributi e metodi



Dal progetto al codice






```
public class PartitaADadi {
    private Dado d1;
    private Dado d2;
    public PartitaADadi() {
        d1 = new Dado();
        d2 = new Dado();
    }
    public void gioca() {
        int vf1, vf2;
        d1.lancia();
        vf1 = d1.getValoreFaccia();
        d2.lancia();
        vf2 = d2.getValoreFaccia();
        ...
    }
}
```

```
public class Dado {
    private int valoreFaccia;
    public Dado() {
        ...
    }
    public void lancia() {
        ...
    }
    public int getValoreFaccia() {
        return valoreFaccia;
    }
}
```

Avete notato la relazione tra requisiti, analisi, progettazione, e programmazione?

1.6 Che cosa è UML

UML è una notazione grafica standard per la modellazione OO

- UML (Unified Modeling Language) è un linguaggio visuale per la specifica, la costruzione e la documentazione degli elaborati di un sistema (software e non) [OMG]
- **www.uml.org**

Questo corso presenta UML, ma in modo parziale

- l'enfasi del corso non è su UML, ma sull'**applicazione** di UML nell'OOA/D

Tipi di diagrammi di UML

UML definisce diversi tipi "grezzi" di diagrammi

- ad es., i diagrammi delle classi

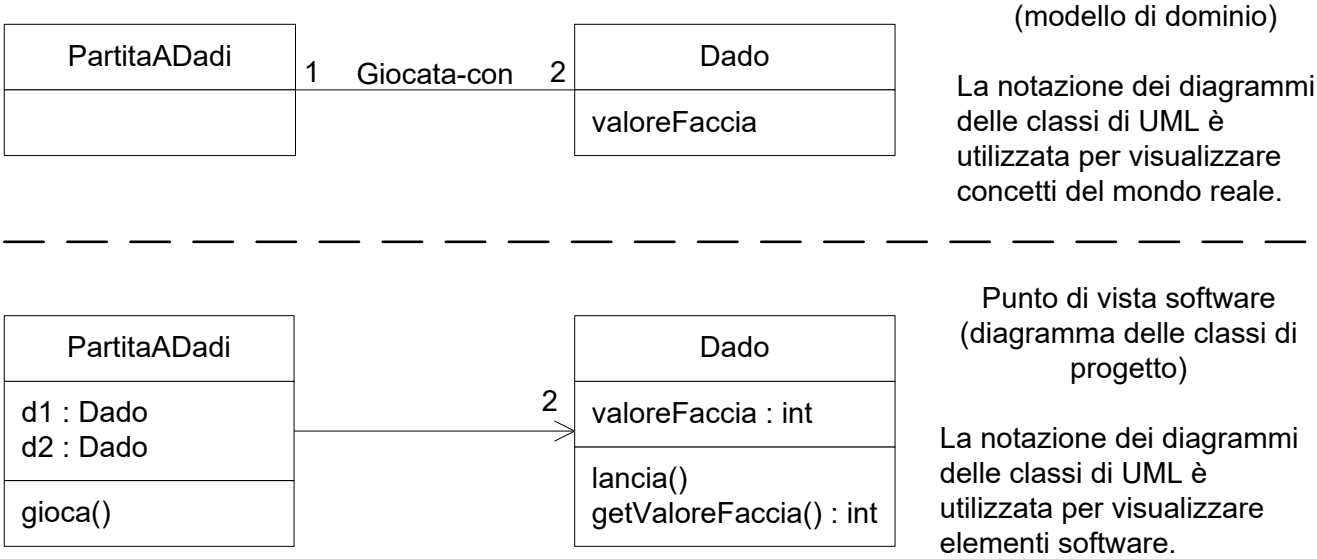
- tuttavia, UML non impone né un metodo né un modo preferito di usare questi diagrammi

Punti di vista per applicare UML

Due *punti di vista* per l'applicazione di UML [Fowler, UML Distilled]

- punto di vista **concettuale**
  - i diagrammi descrivono oggetti del mondo reale o in un dominio di interesse
- punto di vista **software**
  - i diagrammi descrivono astrazioni o componenti software
  - implementazione o specifica
- attenzione: la notazione è sempre la stessa!

Punti di vista differenti



Il significato di "classe" nei diversi punti di vista

In un diagramma delle classi di UML, un rettangolo denota una **classe**

- **classe concettuale**
  - cosa o concetto del mondo reale – nel modello di dominio
- **classe software**
  - una classe che rappresenta un componente software
  - può rappresentare la *specifica* oppure l'*implementazione* di una classe
- **classe**
  - classe concettuale o classe software

Tre modi per applicare UML

Tre *modi* per applicare UML [Fowler, UML Distilled]

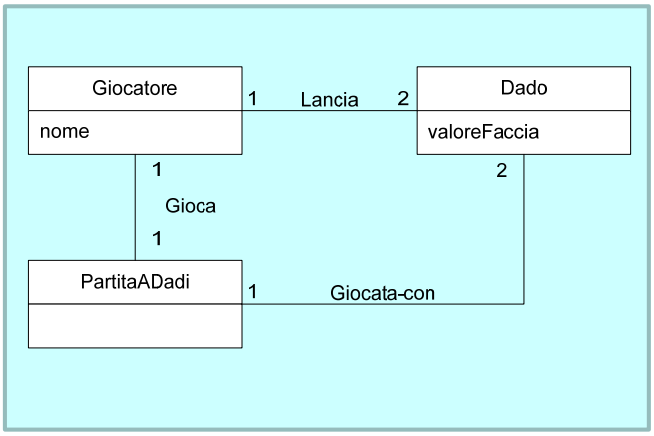
- **UML come abbozzo** (sketch)
  - diagrammi informali e incompleti
- **UML come progetto** (blueprint)
  - diagrammi relativamente dettagliati
- **UML come linguaggio di programmazione**

- la modellazione agile enfatizza l'applicazione di **UML come abbozzo**
- tuttavia, si può imparare ad applicare bene UML come abbozzo solo se si è imparato ad applicare **UML come progetto**

1.7 Vantaggi della modellazione visuale

L'uso di UML implica che si sta lavorando *in modo visuale*

- per sfruttare la capacità del nostro cervello di comprendere più rapidamente concetti e relazioni mostrati con una notazione grafica (prevalentemente bidimensionale)



Classi concettuali e attributi
Giocatore – nome
Dado – valoreFaccia
PartitaADadi

Associazioni
una **PartitaADadi** è giocato da un **Giocatore**
la **PartitaADadi** viene giocata con due **Dadi**
il **Giocatore** lancia i due **Dadi**

- quale di questi due modelli è più facilmente comprensibile? si immagini anche il caso di un modello con dozzine di classi e associazioni