UML
Analisi
Modellazione e altro

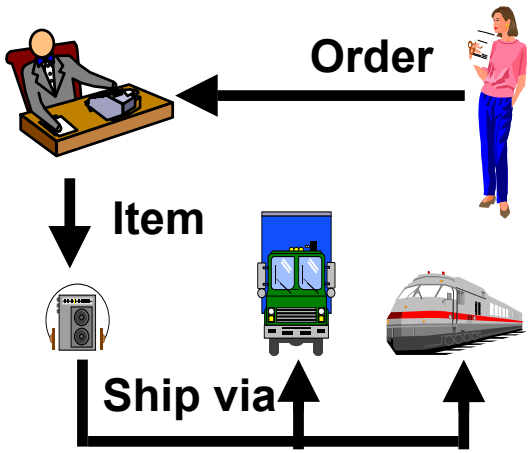
UML (cos'è)

- Linguaggio visuale (sostanzialmente)
- Non proprietario
- Sintesi di notazioni usate in passato (ER, State chart, OMT, ..)
  - in larga misura la notazione è specificatamente OO
  - ma presenta anche caratteristiche tipiche di altre aree
- Autori: Grady Booch, Ivar Jacobson, Jim Rumbaugh

UML (cosa non è)

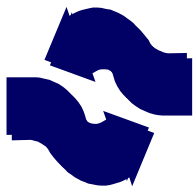
- Non è una metodologia
- Non prescrive un processo software: non dice “prima bisogna fare questa attività, poi quest’altra”

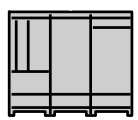
What is Visual Modeling?



Business Process

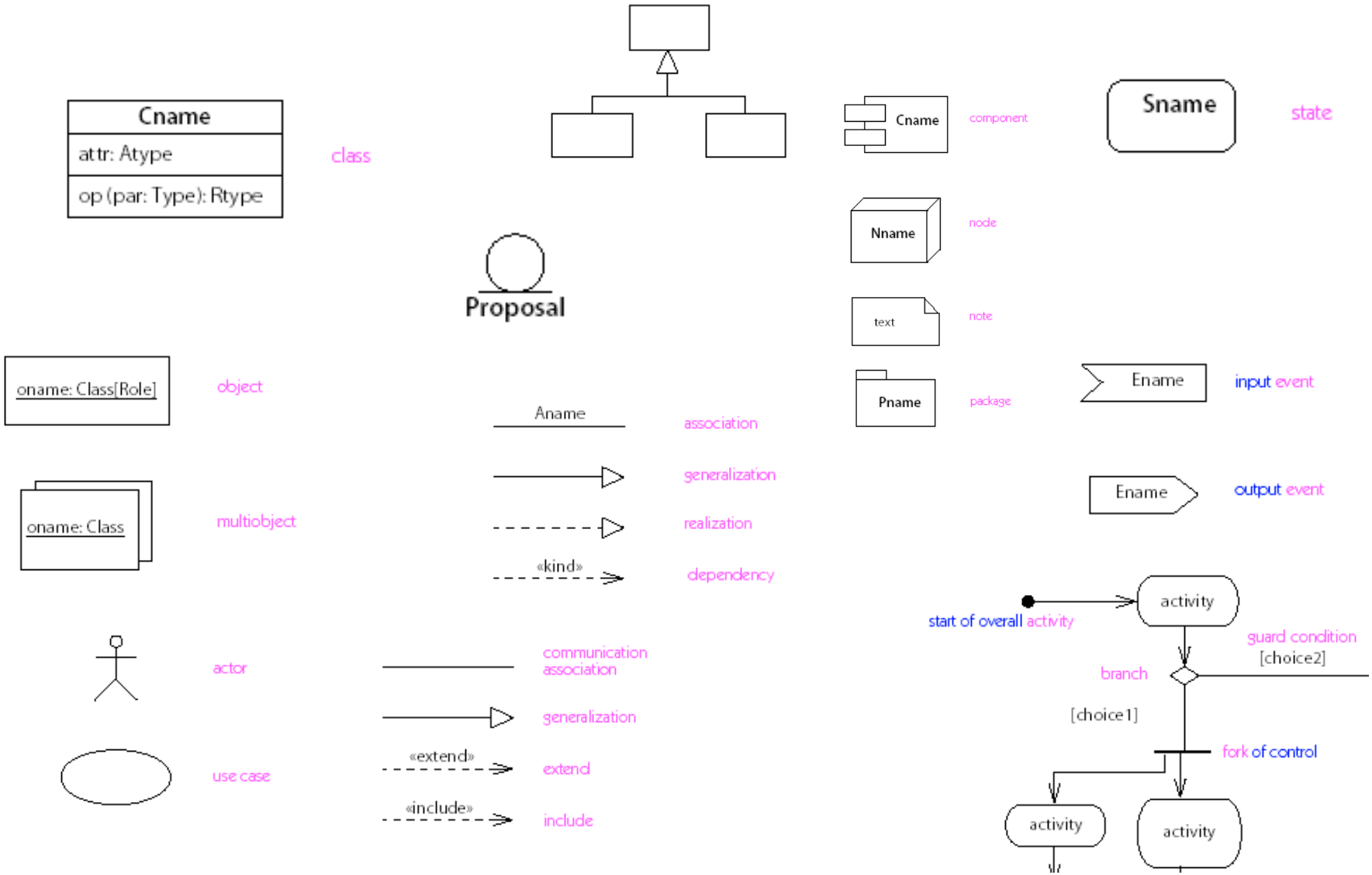
"Modeling captures essential parts of the system."





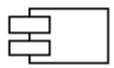
Visual Modeling is modeling using standard graphical notations

Notazione grafica



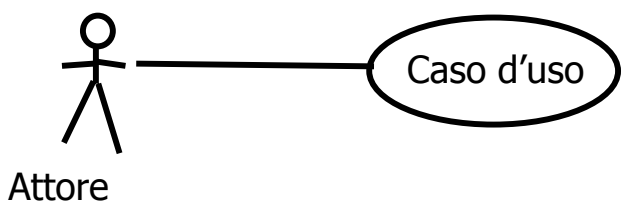
Tipi di diagrammi

- Punto di vista logico
  - Diagramma dei casi d'uso
  - Diagramma delle classi (vista statica)
  - Diagrammi di comportamento (vista dinamica)
    - diagramma di stato
    - diagramma delle attività
    - diagramma di interazione
      - Diagramma di sequenze
      - Diagramma di collaborazione
- Punto di vista fisico (vista statica)
    - Diagramma dei componenti
    - Diagramma di dispiegamento

Classifier	*Function*	*Notation*
actor	An outside user of a system	
class	A concept from the modeled system	Name
class-in-state	A class restricted to being in a given state	Name[S]
classifier role	A classifier restricted to a particular usage in a collaboration	role:Name
component	A physical piece of a system	
data type	A descriptor of a set of primitive values that lack identity	Name
interface	A named set of operations that characterize behavior	 Iname
node	A computational resource	
signal	An asynchronous communication among objects	«signal»
subsystem	A package that is treated as a unit with a specification, implementation, and identity	«subsystem»
use case	A specification of the behavior of an entity in its interaction with outside agents	

Casi d'uso

- Fondamentali nell'analisi.
- Mostrano le relazioni tra "*attori*" e "*casi d'uso*" del sistema.
- Mettono in evidenza le funzionalità di un sistema (o di parte o di una classe) così come le percepisce chi interagisce con esso dal mondo esterno.
- Centrati sull'utente

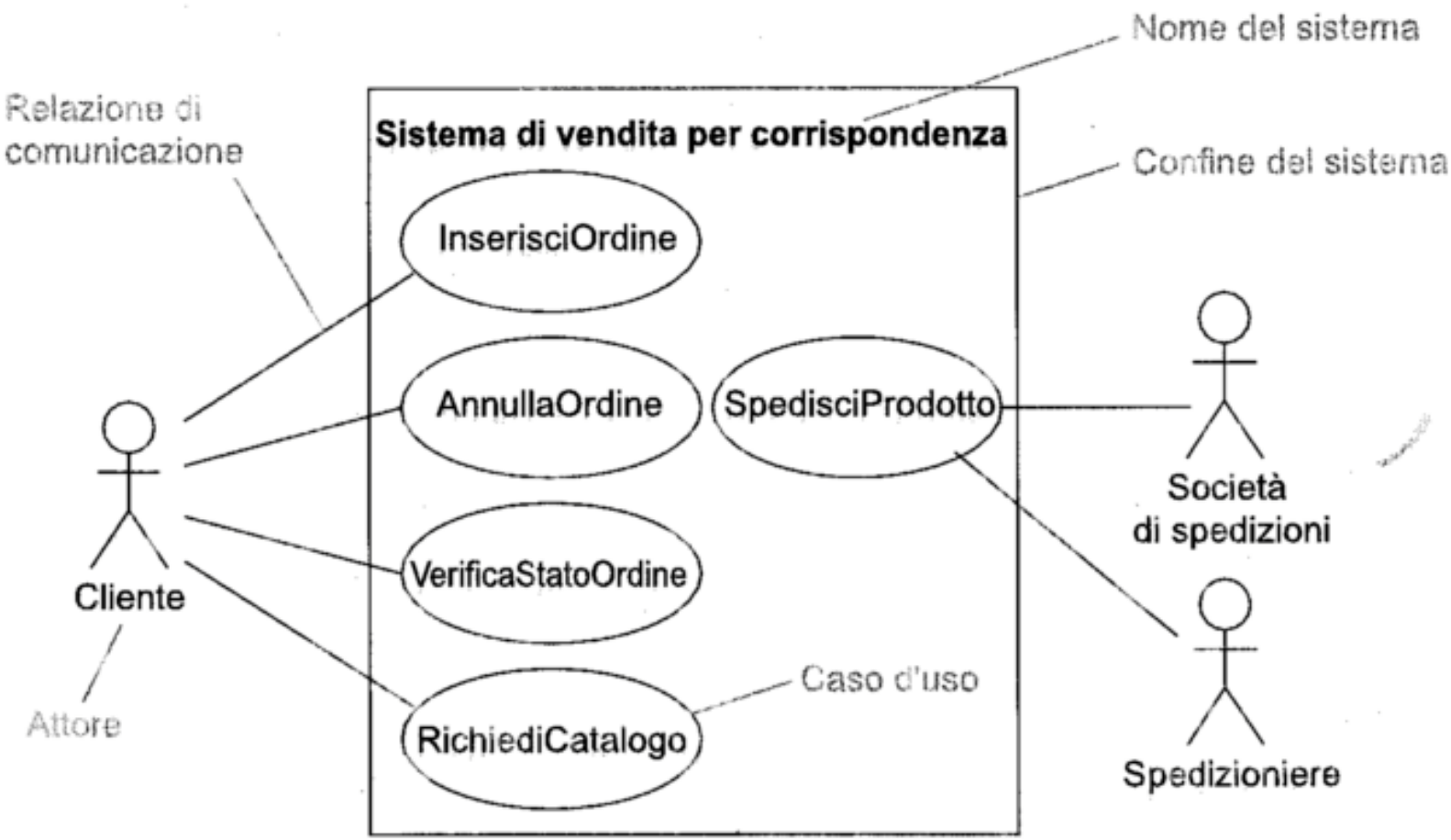


Casi d'uso

- Un caso d'uso è una coerente funzionalità di un sistema o di una classe; la funzionalità corrisponde a una sequenza di messaggi scambiati tra sistema e uno o più attori esterni assieme alle azioni svolte dal sistema stesso.
- Un caso d'uso è schematizzato con una

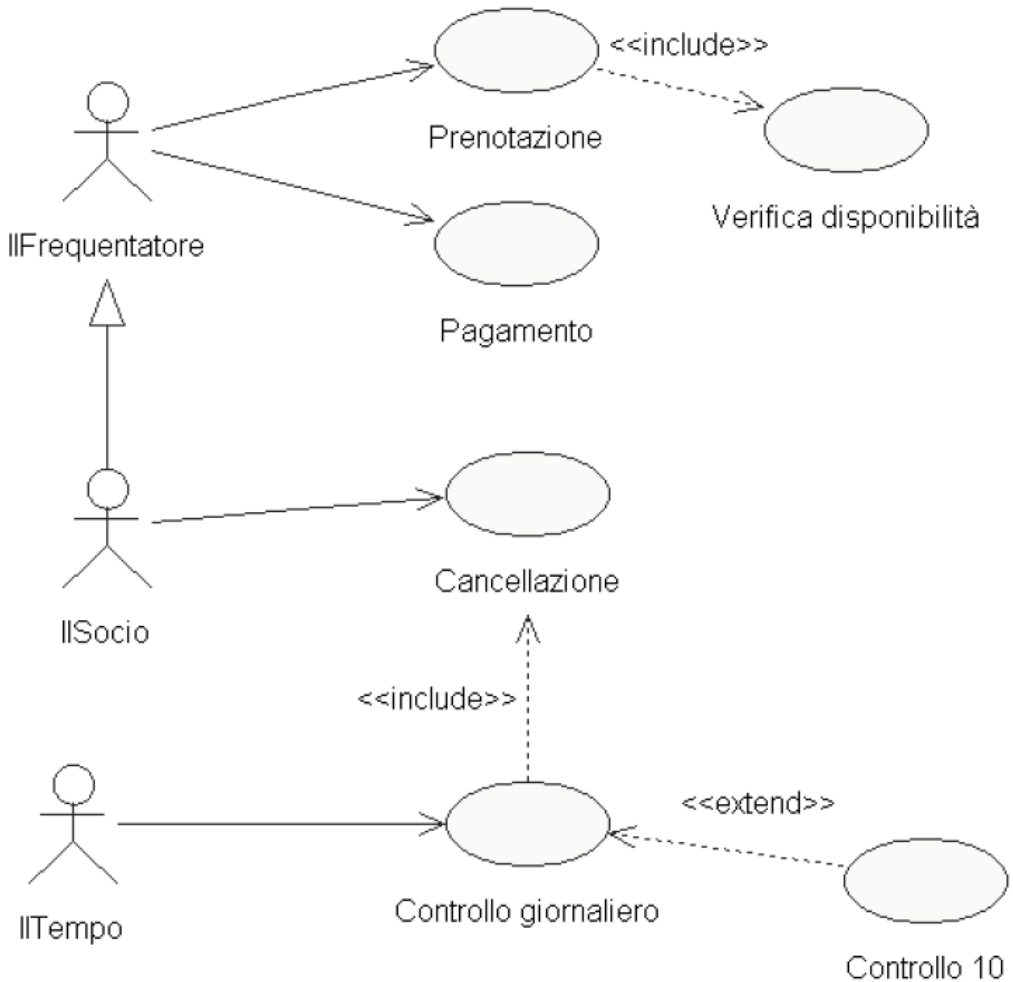
  ellisse contenente il nome del caso d'uso.
- Un diagramma di caso d'uso è un grafo comprendente attori, casi d'uso (racchiusi dalla scatola del sistema) e associazioni (partecipazioni) tra attori e casi d'uso.

Esempio

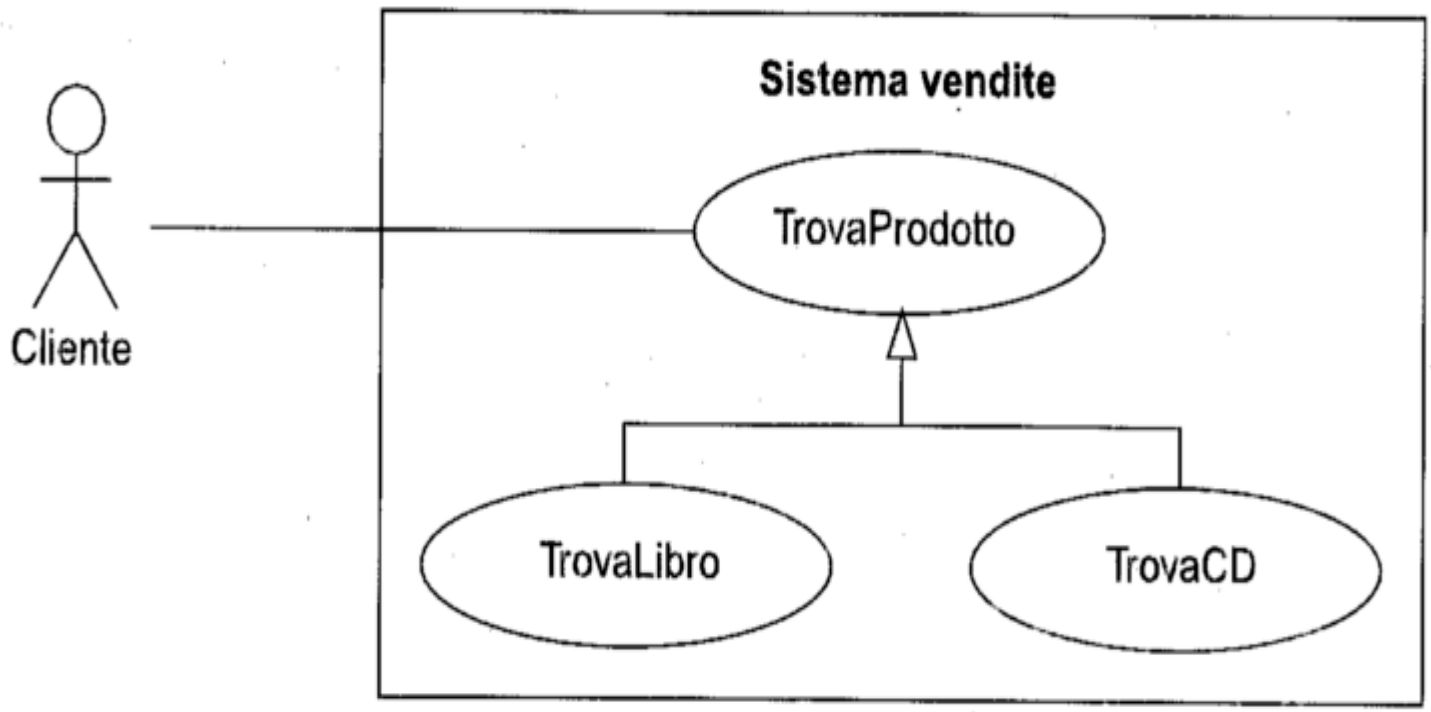

Nome del sistema
Relazione di comunicazione
Sistema di vendita per corrispondenza
Confine del sistema
InserisciOrdine
AnnullaOrdine
SpedisciProdotto
VerificaStatoOrdine
RichiediCatalogo
Caso d'uso
Cliente
Attore
Società di spedizioni
Spedizioniere

Diagramma

- La linea che unisce un attore a un caso d'uso costituisce una *comunicazione.*
- La comunicazione è l'unica relazione tra attori e casi d'uso.
- Sono possibili le relazioni seguenti:
  - *estensione* <<*extend*>>
  - *inclusione* <<*include*>>
  - *generalizzazione* 


<<include>>
Prenotazione
Verifica disponibilità
IlFrequentatore
Pagamento
Cancellazione
IlSocio
<<include>>
<<extend>>
IlTempo
Controllo giornaliero
Controllo 10

Generalizzazione



Metodo

- Identificare gli attori
- Definire cosa un attore vuol fare

  => caso d'uso
- Per ogni caso d'uso una descrizione testuale la cui essenza è:
  - L'attore fa <xx> il sistema fa <yy>
  - L'attore fa <aa> il sistema fa <bb>

Descrizione testuale

- Numero e titolo
- Breve descrizione
- Attori
- Precondizioni per l'esecuzione del CU
- Flusso (scenario) principale
- Flussi (scenari) alternativi
- Postcondizioni

Esempio di caso d'uso testuale

Caso d'uso: Acquisto di un prodotto

- Il cliente naviga nel catalogo e seleziona gli articoli
- il cliente "va alla cassa"
- il cliente indica le informazioni relative alla spedizione
- il sistema prospetta la spesa complessiva
- il cliente fornisce dati circa la sua carta di credito
- il sistema autorizza l'acquisto
- il sistema conferma la vendita
- il sistema invia al cliente una e-mail di conferma

- Alternativa: *Carta di credito non valida*
  - Il sistema non autorizza l'acquisto *(seguono azioni corrispondenti)*
- Alternativa: *Cliente abituale*
  - Al terzo passo il sistema chiede conferma della carta di credito (fornendo le ultime 8 cifre)
  - il quinto passo viene saltato

Caso d'uso: TrovaProdotto
ID: UC12
Attori: Cliente
Precondizioni:
Sequenza degli eventi: 1. Il caso d'uso inizia quando il Cliente seleziona "trova prodotto". 2. Il sistema chiede al Cliente i criteri di ricerca. 3. Il Cliente inserisce i criteri di ricerca richiesti. 4. Il sistema ricerca i prodotti che soddisfano i criteri specificati dal Cliente. 5. Se il sistema trova uno o più prodotti 5.1. Il sistema mostra un elenco dei prodotti che soddisfano i criteri di ricerca. 6. Altrimenti 6.1. Il sistema comunica al cliente che non sono stati trovati prodotti che soddisfano i criteri specificati.
Postcondizioni:
Sequenza alternativa: 1. In qualunque momento il Cliente può spostarsi su una pagina diversa.
Postcondizioni:

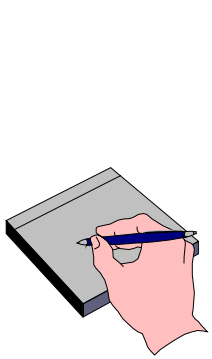
Scenari

Caso d'uso: OperazioniDiCassa
ID: UC14
Attori: Cliente
Precondizioni:
Scenario principale: 1. Il caso d'uso inizia quando il Cliente seleziona "procedi alla cassa". 2. Il sistema visualizza l'ordine del Cliente. 3. Il sistema richiede l'identificatore del cliente. 4. Il Cliente inserisce un identificatore di cliente valido. 5. Il sistema recupera e visualizza i dati del Cliente. 6. Il sistema richiede le informazioni della carta di credito – intestatario, numero e scadenza della carta. 7. Il Cliente inserisce le informazioni della carta di credito. 8. Il sistema richiede la conferma dell'ordine. 9. Il Cliente conferma l'ordine. 10. Il sistema addebita la carta di credito. 11. Il sistema visualizza la ricevuta.
Scenari secondari: IdentificatoreClienteNonValido InformazioniCartaCreditoNonValide SuperatoLimiteUtilizzoCartaCredito CartaCreditoScaduta
Postcondizioni:

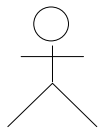
Scenario secondario

Caso d'uso: OperazioniDiCassa Scenario secondario: IdentificatoreClienteNonValido
ID: UC15
Attori: Cliente
Precondizioni:
Scenario secondario: 1. Il caso d'uso inizia al passo 3 del caso d'uso OperazioniDiCassa quando il Cliente inserisce un identificatore cliente non valido 2. Per tre tentativi di inserimento 2.1. Il sistema richiede al Cliente di inserire nuovamente il proprio identificatore. 3. Il sistema informa il Cliente che l'identificatore cliente inserito non è stato riconosciuto.
Postcondizioni:

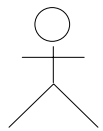
ruolo dei casi d'uso



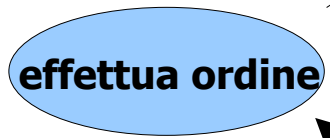
requisiti

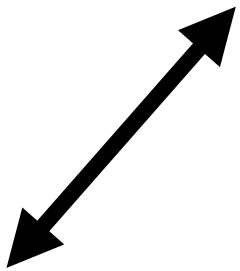


Acquirente

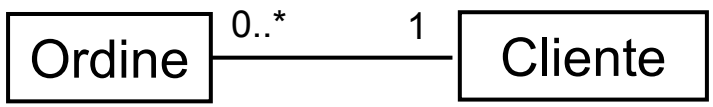


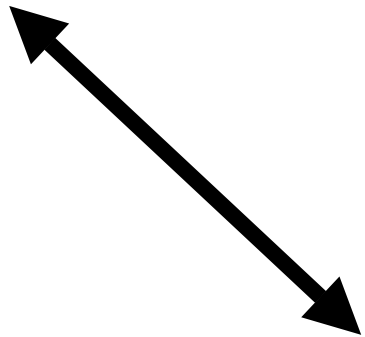
Venditore





modelli di analisi e progetto





casi prova

Diagramma delle classi

- E' il nucleo fondamentale di UML
- Mostra gli oggetti/classi contenuti nel sistema e le loro relazioni reciproche
- E' una rappresentazione strutturale
- Corrisponde a informazioni che non evolvono nel tempo => *vista statica*
- La vista dinamica viene catturata da altri formalismi

Modellazione/diagr Classi

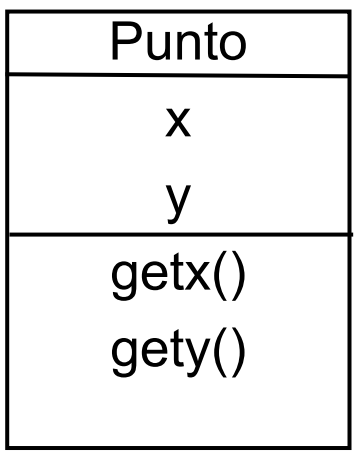
- Prospettive
  - Livello concettuale: *relazioni tra le classi*
  - Livello di specifica: *responsabilità delle classi (interfacce)*
  - Livello di implementazione: *dettagli realizzativi (puntatori, liste, etc..)*

Classe

- Rappresenta un *concetto* dell'applicazione che viene modellata
  - un'entità fisica, esempio: automobile
  - un'entità del settore applicativo: fattura
  - un'entità logica: un gestore di ordini
  - un'entità applicativa: un pulsante sull'interfaccia
  - un'entità "informatica": una tabella hash
  - un'entità comportamentale: un task
- Una classe rappresenta collettivamente tutti gli oggetti che hanno lo stesso comportamento

Rappresentazione classi

- Differenti livelli di dettaglio
  - Rettangolo con nome
  - Rettangolo con nome, attributi e metodi
  - c.s. con in più la visibilità, tipi



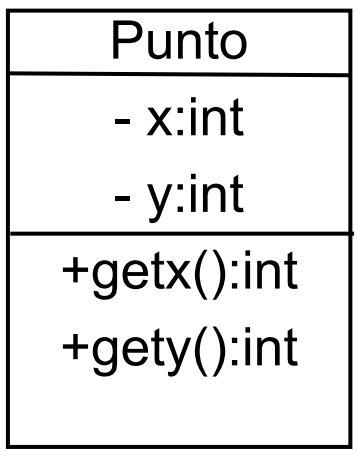
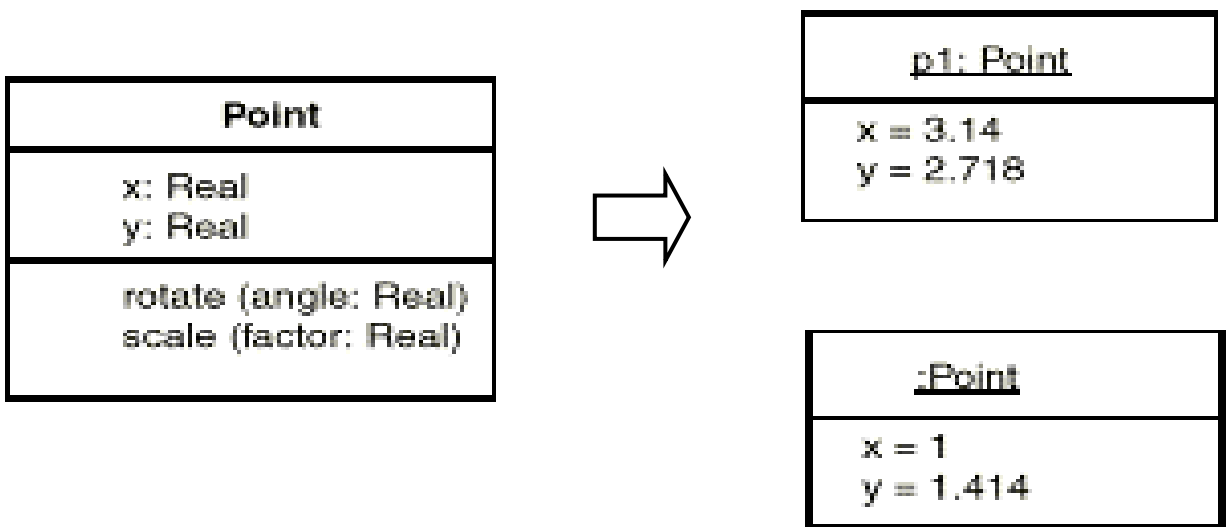


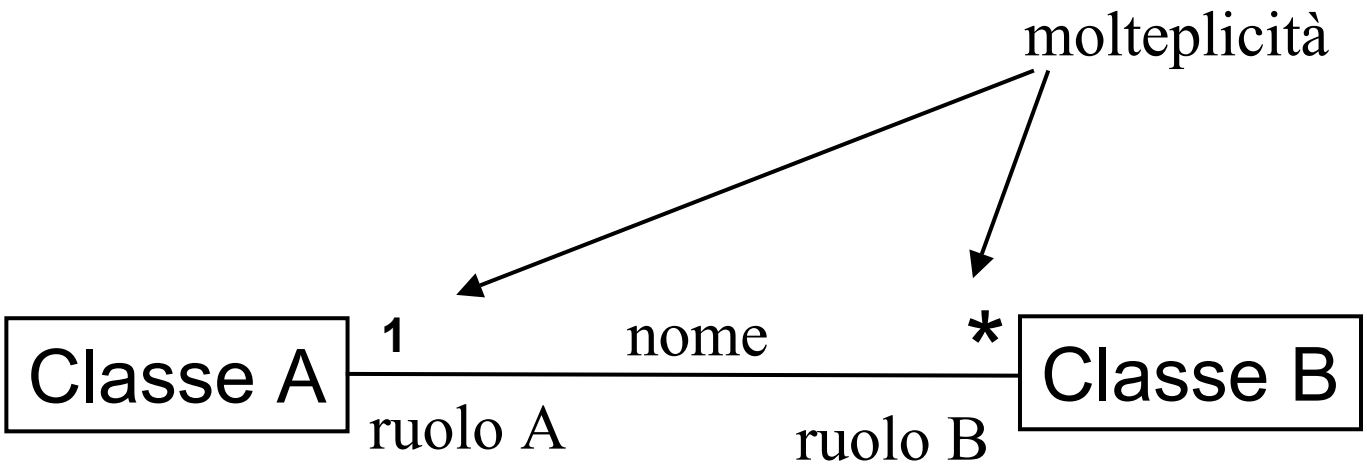
Diagramma degli oggetti

- La sintassi degli Object Diagram ricalca quella dei Class Diagram
- Descrive oggetti specifici, ottenuti istanziando classi



Associazione

- E' una qualunque relazione tra oggetti
  - rappresentata con una linea su cui si scrivono i ruoli e le molteplicità
  - spesso binaria (1 a 1)



Esempi di molteplicità

Una persona può possedere 0 o più case; una casa è posseduta da una o più persone

Una casa è situata in una o in nessuna città; una città contiene 1 e più case

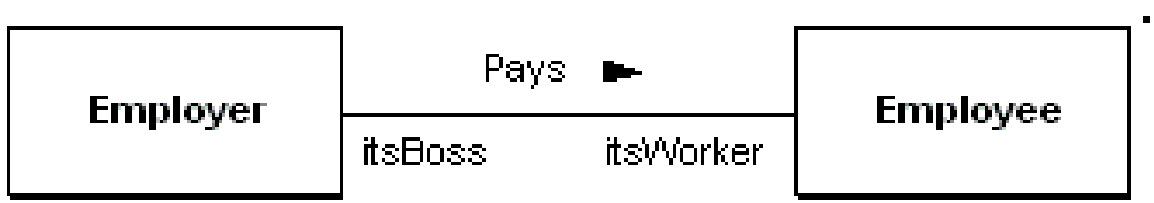
Un poligono ha 3 o più linee come lati; una linea può far parte di nessuno o più poligoni

Una linea ha estremi in due punti; un punto può essere l'estremo di 0 o più linee

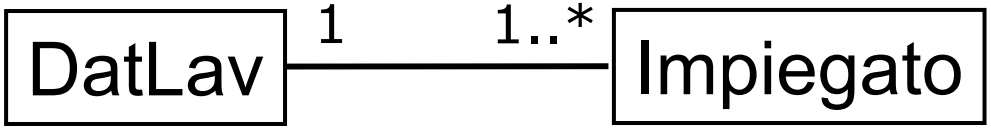
Associazione

- UML prevede la molteplicità
  - Se manca si interpreta come indefinita

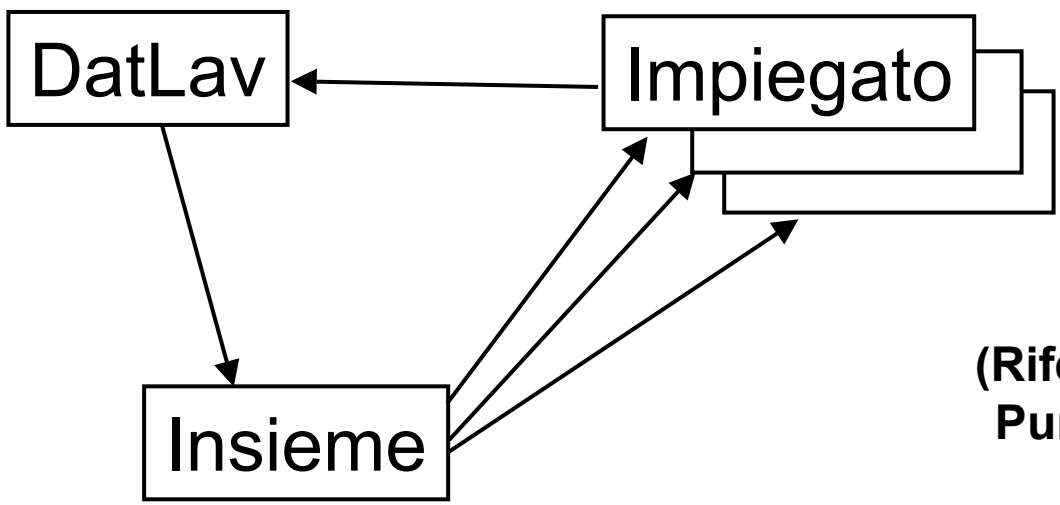
    (nessuno rispetta questa regola: viene interpretata come 1)
- La linea senza frecce indica navigabilità bidirezionale



Associazione



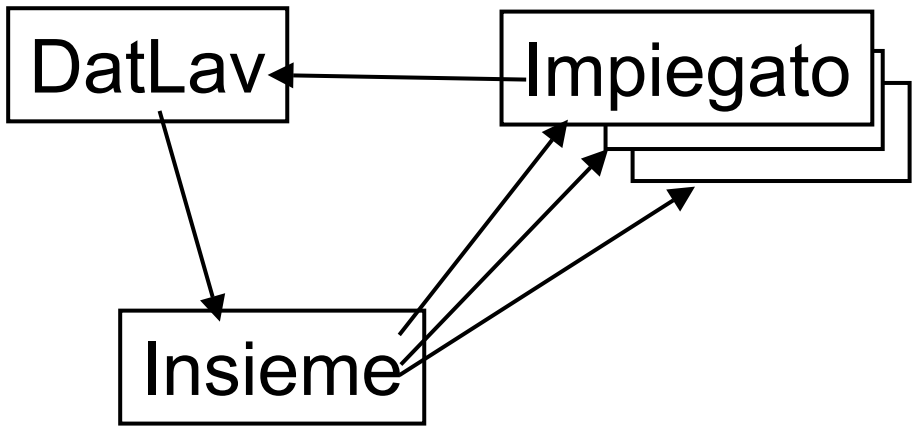
Possibile implementazione



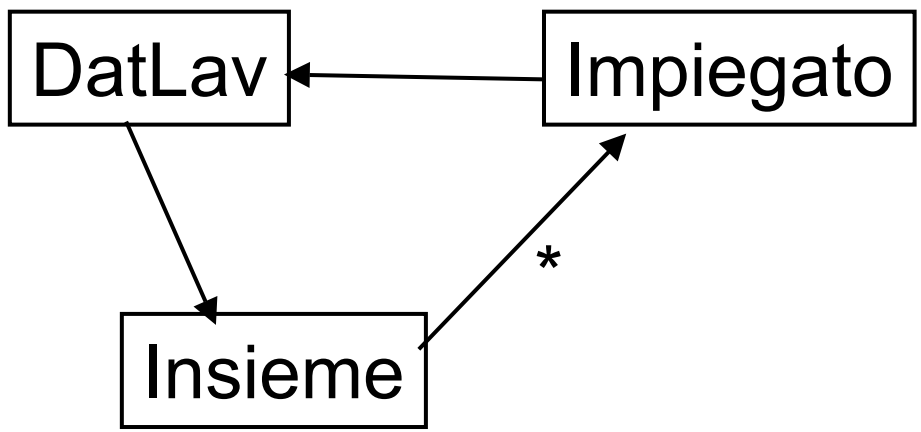
(Riferimenti, Puntatori)

Notazione

Implementazione



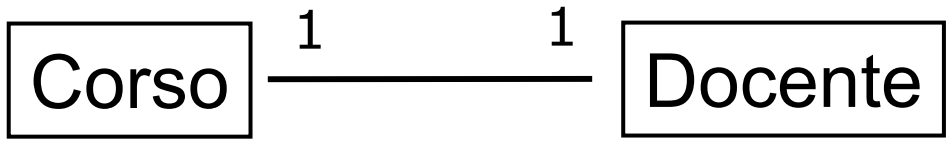
Notazione corrispondente



Associazione (implement.)


```
class DatLav {
    private Impegato[] dip;
}

class Impegato {
  private DatLav capo;
  private Progetto p;
}

class Progetto {
  // nessun rif a impiegato;
}
```

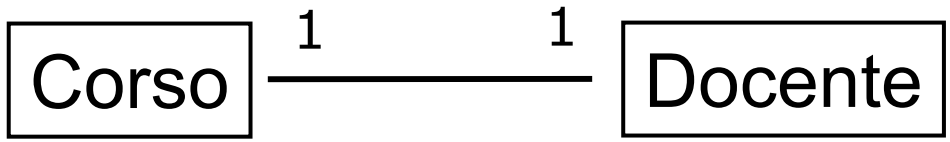
Associazione (implem.1)




```
class Corso {
  private Docente Titolare;
  void setT(Docente d) {Titolare= d;}
}
class Docente {
  private Corso Titolarita;
  void setT(Corso c) {Titolarita= c;}
}
```

Titolare e **Titolarità** sono *riferimenti*

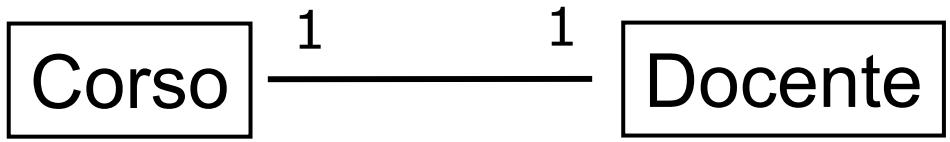
Associazione (implem.1) segue



Da qualche parte

```
Corso IngSW = new Corso()
…
Docente Bucci = new Docente();
Bucci.setT(IngSW);
IngSW.setT(Bucci);
```

Associazione (implem.1) segue



Da qualche parte

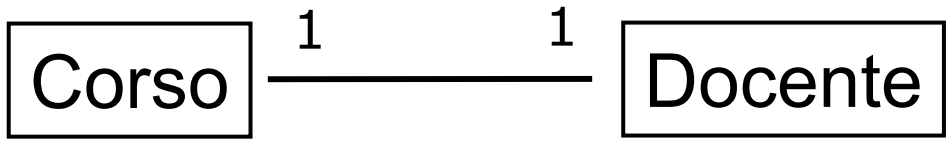
```
Corso IngSW = new Corso()
…
```

Alternativamente

```
IngSW.setT(Bucci);
```

```
Docente Bucci = new Docente(IngSW)
```

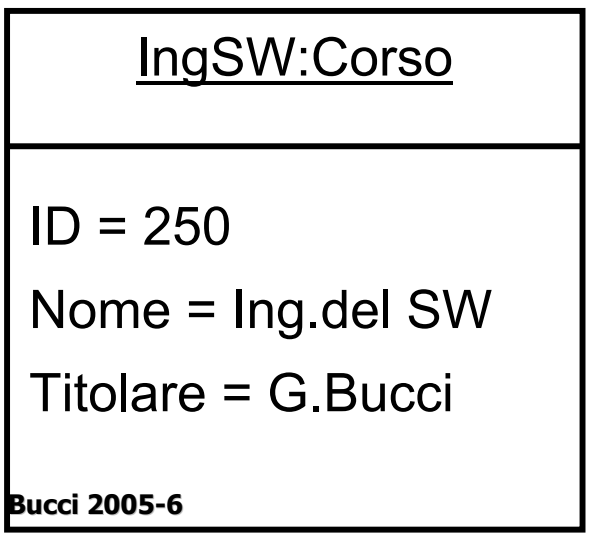
Associazione (implem.2)

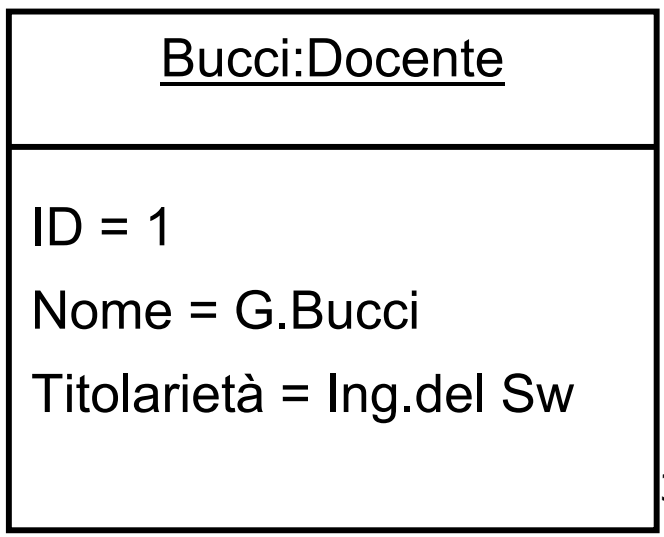



```
class Corso {
      private int ID;
      private String Nome;
      private String Titolare;

}

class Docente {
      private int ID;
      private String Nome;
      private String Titolarità
   }
```

Associazione (implem.2) segue

```
Corso IngSW =
      new Corso(250,”Ing.del SW”,”G.Bucci”);

Docente Bucci =
      new Docente(1,”G.Bucci”,”Ing.del SW”);
```





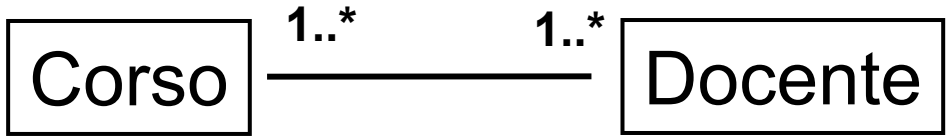
Associazione (implem.2) segue

- Anche questa è una associazione.
- Occorre che ci sia un campo (attributo) univoco in ciascun oggetto
  - Ad esempio ID o Nome
- La navigazione è più macchinosa

Associazione (implem.2) segue

```
String getName();  //in Corso

Corso unCorso;

if (unCorso.getName()==“Ing.del SW”){
    // è ingegneria del SW
    }
```

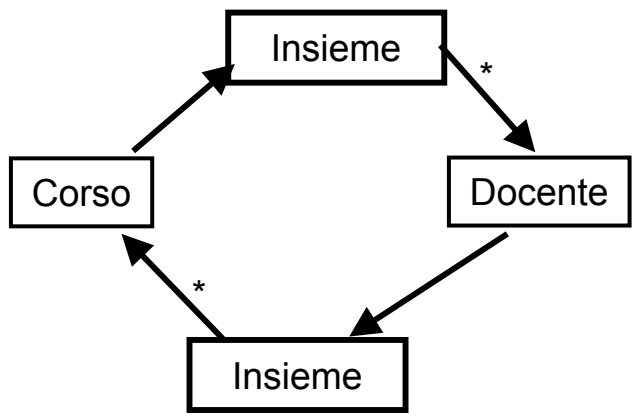
Implementazione



Java

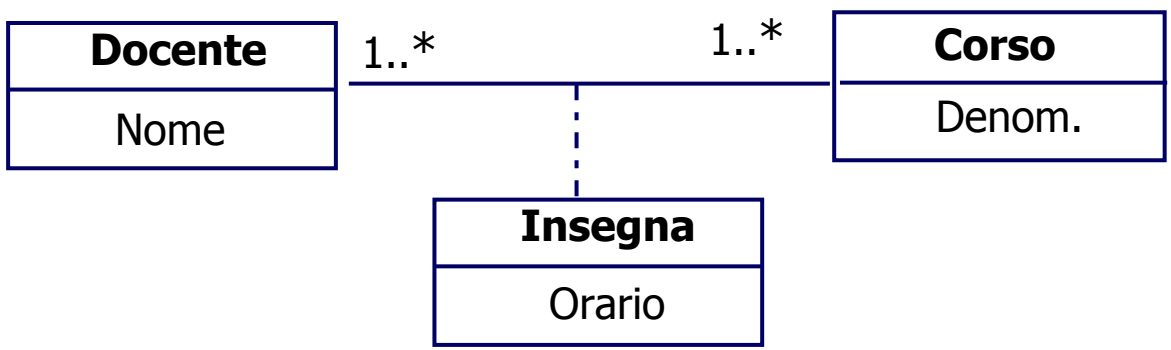
```
class Corso {
    private Docente [] prof;
}

class Docente {
    private Corso [] discipl;
}
```

Schema con puntatori

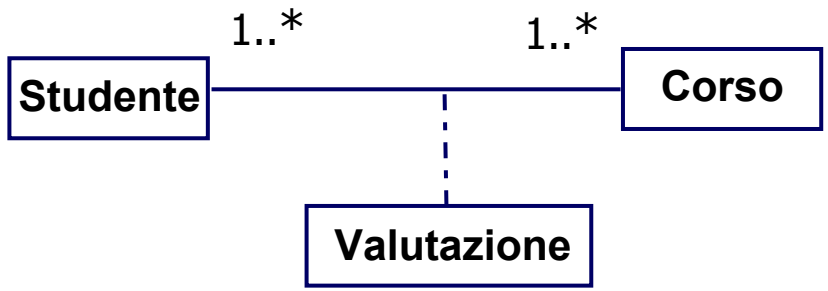


Classe di associazione

- Docente tiene 1 o più corsi, Corso è tenuto da 1 o più docenti
- Se si vuole anche esprimere l'orario in cui un docente tiene un corso? L'orario non può stare né in Docente né in Corso !
- E' necessario che l'associazione abbia attributi, cioè che sia una classe



Associazione con attributi

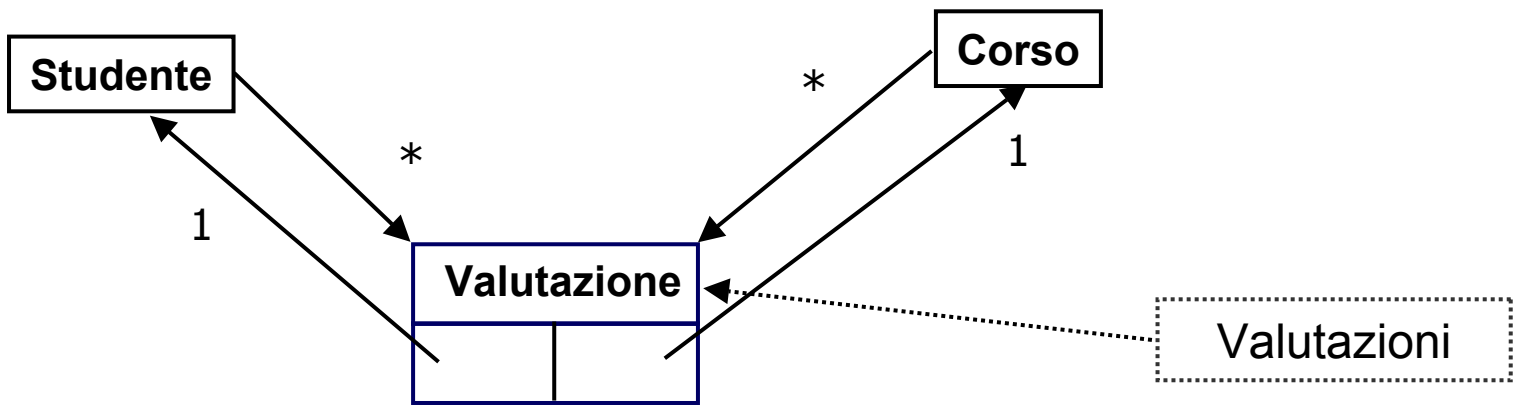


Dato un certo oggetto Studente e un certo oggetto Corso esiste un solo oggetto Valutazione

Reificazione

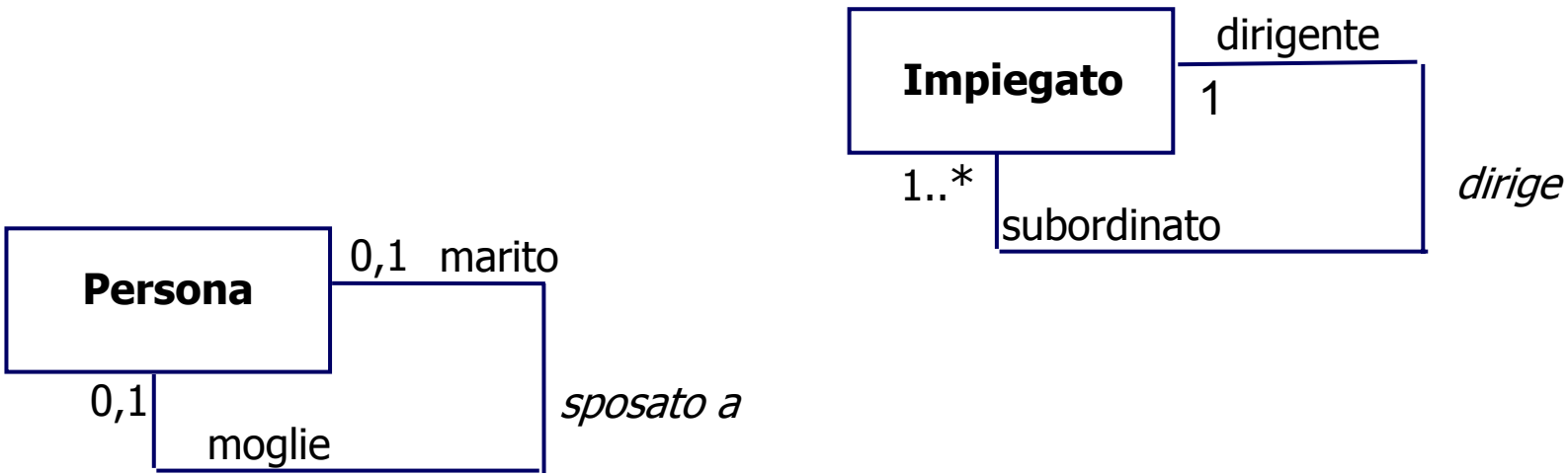
Associazione con attributi

- In fase di progetto/codifica occorre stabilire come vengono realizzate le associazioni tra gli oggetti (puntatore in un verso, puntatori nei due versi, creazione di un oggetto associazione).



Ass. unaria (ricorsiva)/ruoli

- Ruoli nell'associazione



Approccio (elementare) OO

- Si modellano le entità del mondo reale con oggetti (un oggetto per ciascuna entità del mondo reale)
- Si identificano le relazioni di associazione
- Si cercano le comunalità e le si raccolgono attraverso il concetto di classe
- Si costruiscono gerarchie di classi, specializzando, ridefinendo il comportamento nelle classi subordinate

Modellazione (concettuale)

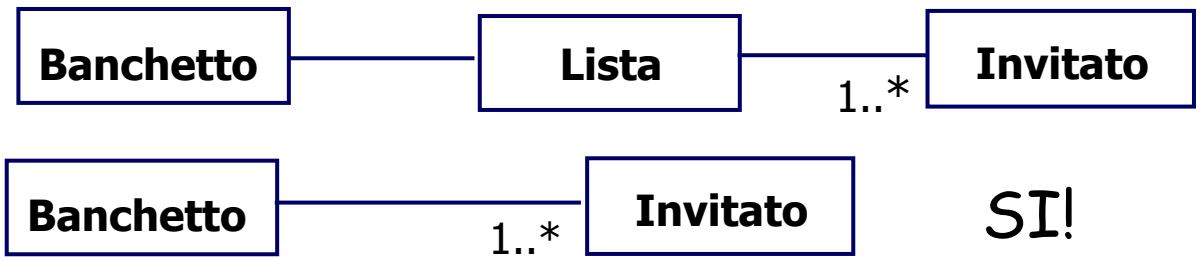
- Costruire il modello dell'applicazione
  - In esso entrano solo le entità che fanno parte del dominio applicativo
    - dette *classi di analisi*
    - ovvero classi ***entity***
  - Non entrano gli oggetti che rappresentano soluzioni progettuali/implementative (liste, puntatori, ecc) o altro (interfacce utente, oggetti con funzione di controllo

Identific. delle associazioni

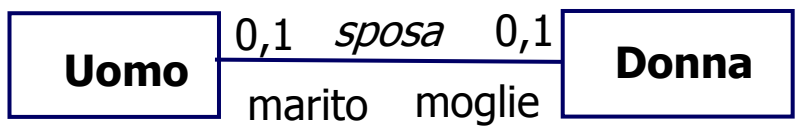
- Le associazioni sono tipicamente espresse da verbi che esprimono:
  - collocazione fisica (*contenuto in*); azioni (*gestisce, corre, ..*); proprietà (*possiede, ha*); soddisfacimento di condizioni (*sposato a*)

- Suggerimenti
  - Ogni riferimento da una classe a un'altra è un'associazione
  - Una associazione deve esprimere una proprietà strutturale del dominio, non un evento transitorio

Identificazione delle classi

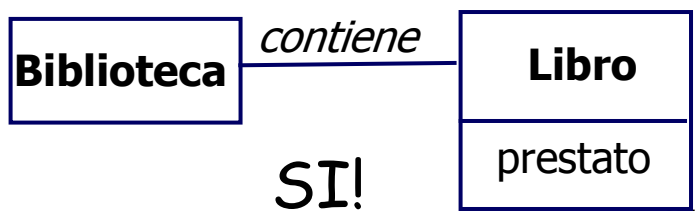


NO!



SI!

NO!



SI!

Esempio: Modellazione di un museo

- Un museo si compone di diverse sezioni, ciascuna comprende un certo numero di sale.
- Ogni sezione ha un orario di apertura giornaliero ed è custodita durante l'orario da un solo custode secondo un turno settimanale; il turno definisce quale sezione un custode deve sorvegliare ogni giorno della settimana.
- Ciascuna sala comprende diverse opere d'arte: dipinti, sculture (divise in bassorilievi, altorilievi, statue), arazzi e ceramiche.
- Ogni opera ha un autore, ma ci sono opere di autore sconosciuto.
- Ogni opera appartiene a un periodo storico (Rinascimento, Medio Evo, Novecento, ..); alcuni autori appartengono a un movimento artistico (impressionismo, cubismo, futurismo,…)

Esempio (continuazione)

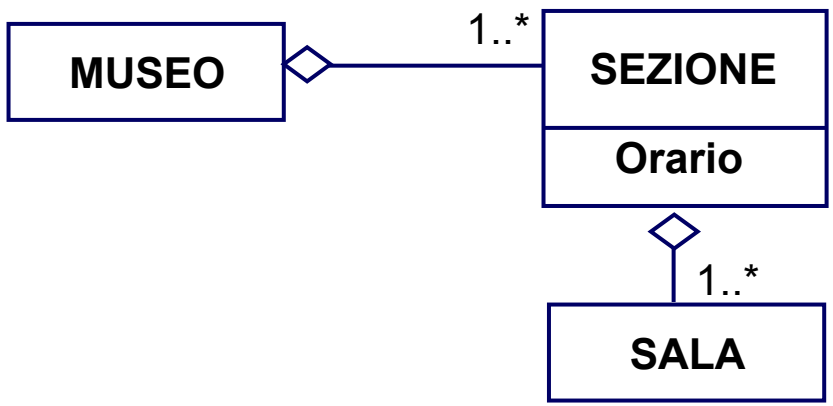
- Si analizza attentamente il testo alla ricerca dei sostantivi che identificano le classi e i loro eventuali attributi
  - Un **museo** si compone di diverse **sezioni**, ciascuna comprende un certo numero di **sale**.
  - Ogni **sezione** ha un *orario* di apertura giornaliero ed è custodita durante l'orario da un solo **custode** secondo un turno settimanale; il turno definisce quale **sezione** un **custode** deve sorvegliare ogni **giorno della settimana**.

Esempio (continuazione)

- Ciascuna **sala** comprende diverse **opere d'arte**: **dipinti**, **sculture** (divise in **bassorilievi**, **altorilievi**, **statue**), **arazzi** e **ceramiche**.
- Ogni **opera** ha un **autore**, ma ci sono opere di autore sconosciuto.
- Ogni **opera** appartiene a un **periodo storico** (*Rinascimento, Medio Evo, Novecento*, ..); alcuni autori appartengono a un **movimento artistico** (*impressionismo, cubismo, futurismo*,…)

Esempio (continuazione)

- Si analizza attentamente il testo alla ricerca dei sostantivi che identificano le classi e i loro eventuali attributi
  - Un **museo** si compone di diverse **sezioni**, ciascuna comprende un certo numero di **sale**. Ogni **sezione** ha un *orario* di apertura giornaliero…..

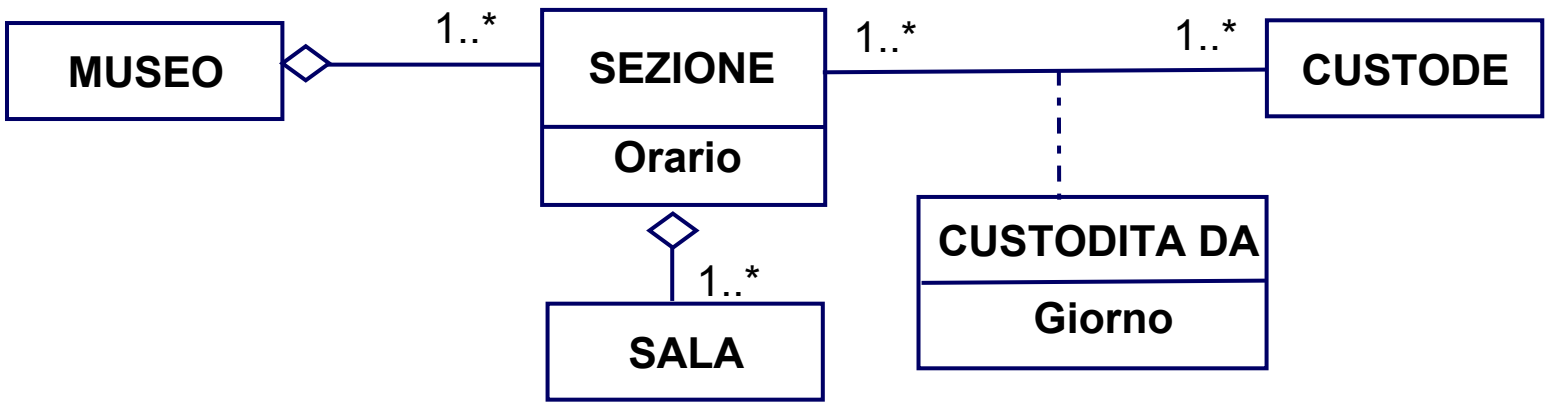


Esempio (continuazione)

- Ogni **sezione** ha un *orario* di apertura giornaliero ed è custodita durante l'orario da un solo **custode** secondo un turno settimanale; il turno definisce quale **sezione** un **custode** deve sorvegliare ogni **giorno della settimana**.

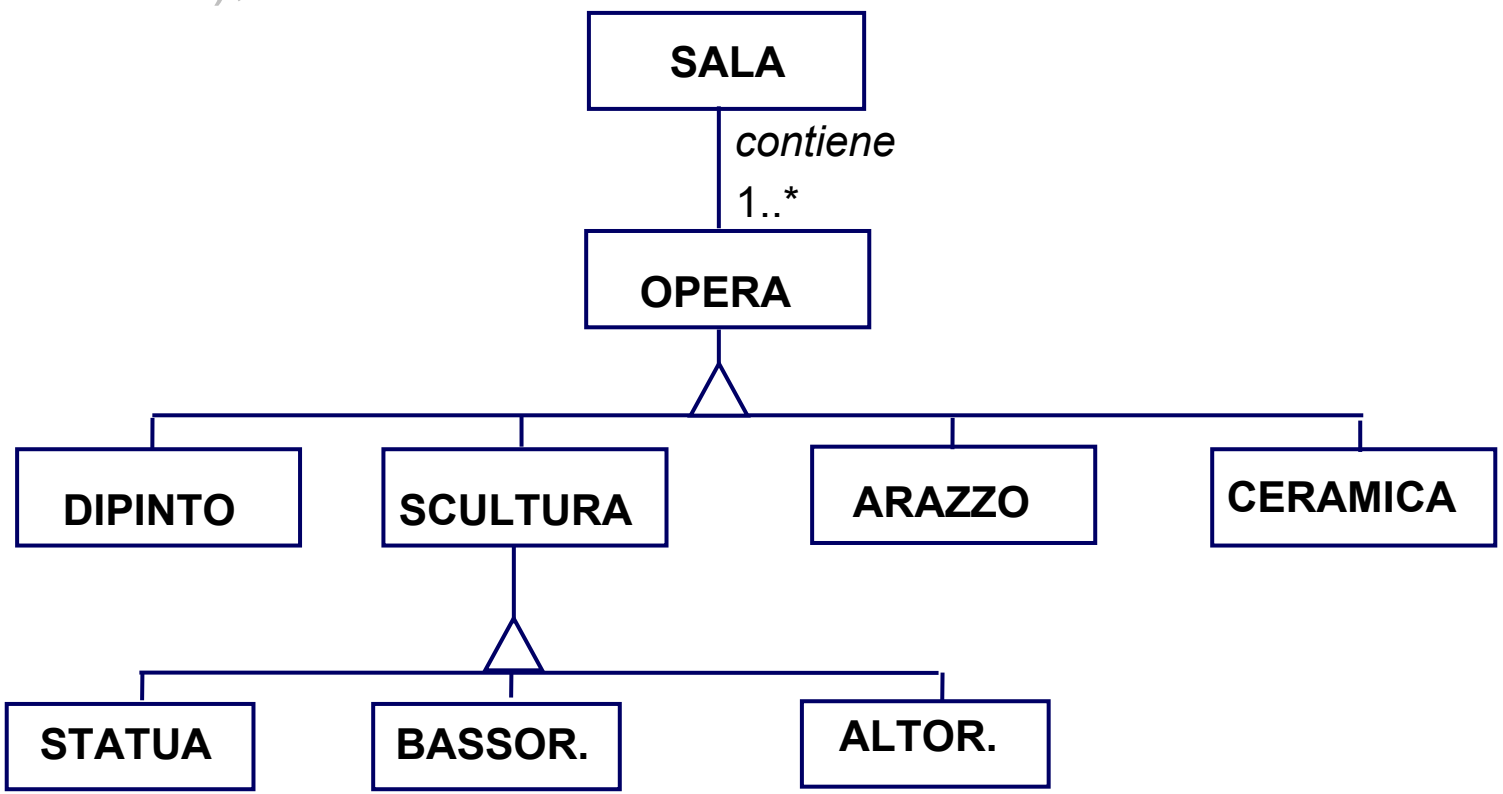
- Qui c'è una associazione ternaria: un custode è associato a una sala ma in base al turno. L'associazione stessa è una classe che lega sezioni a custodi in base al giorno della settimana

Esempio (continuazione)



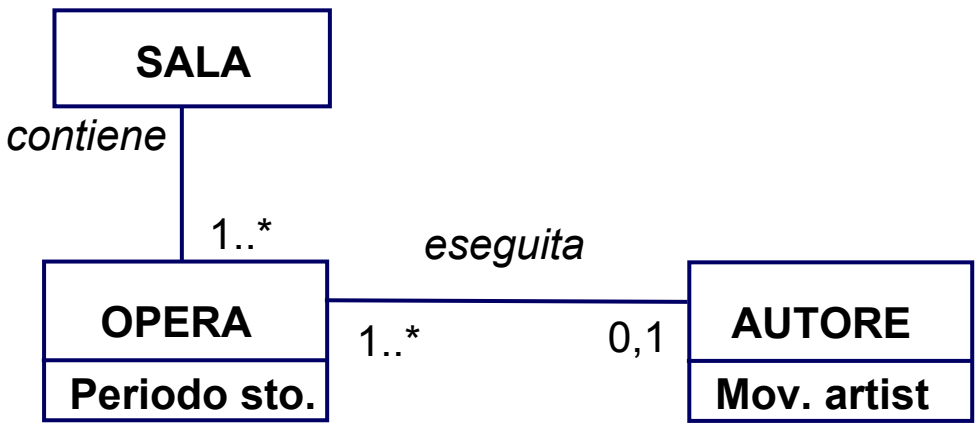
Esempio (continuazione)

- Ciascuna **sala** comprende diverse **opere d'arte**: **dipinti**, **sculture** (divise in **bassorilievi**, **altorilievi**, **statue**), **arazzi** e **ceramiche**.



Esempio (continuazione)

- Ogni **opera** ha un **autore**, ma ci sono opere di autore sconosciuto.
- Ogni **opera** appartiene a un **periodo storico** (*Rinascimento, Medio Evo, Novecento*, ..); alcuni autori appartengono a un **movimento artistico** (*impressionismo, cubismo, futurismo*,…)



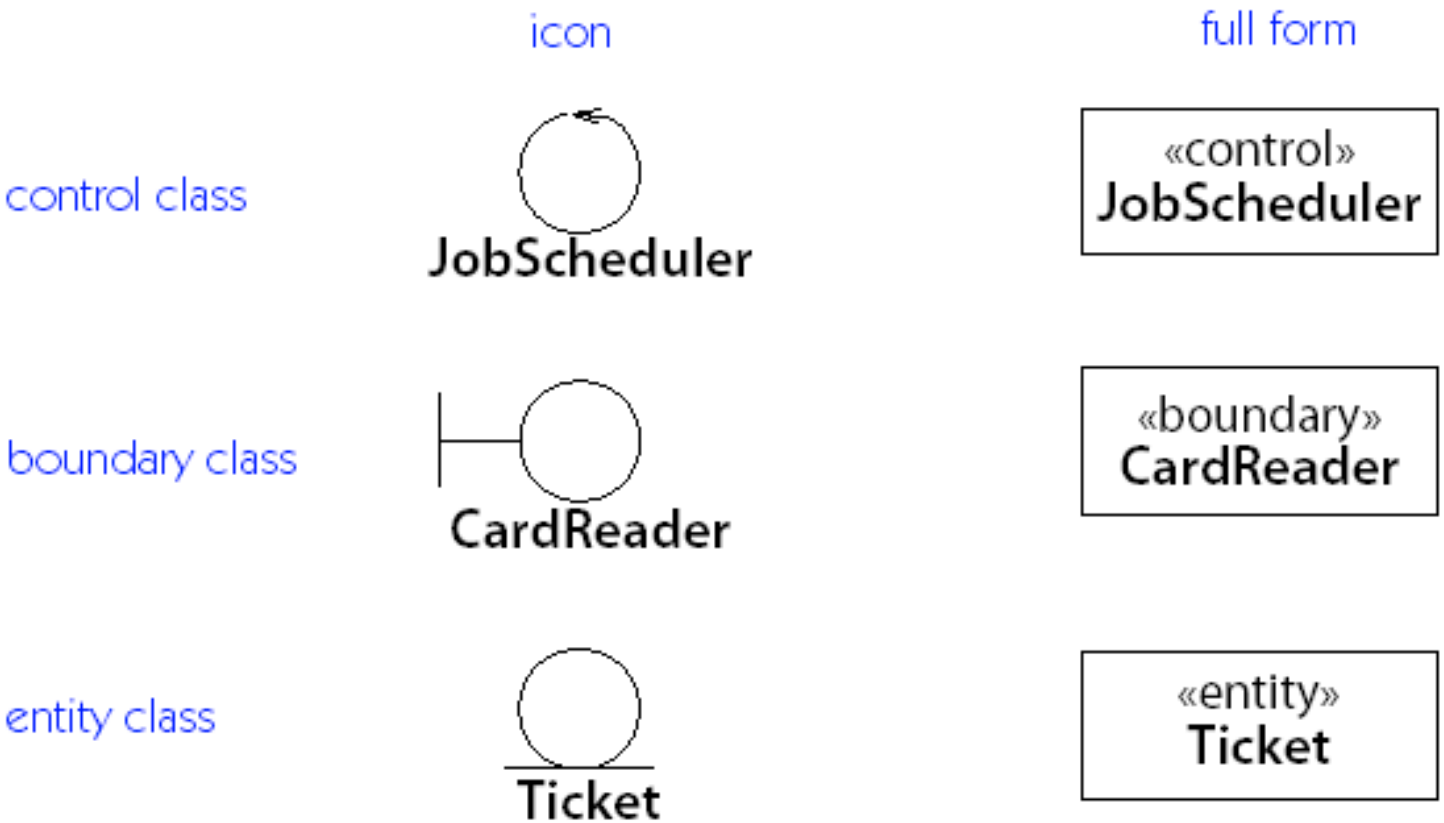
Commento

- L'esempio precedente è un caso applicativo molto statico, tipico problema da base di dati
  - è difficile immaginare un caso d'uso diverso da una interrogazione o un aggiornamento
- Interessano casi in cui gli oggetti interagiscano tra loro

Tre stereotipi (di classe)

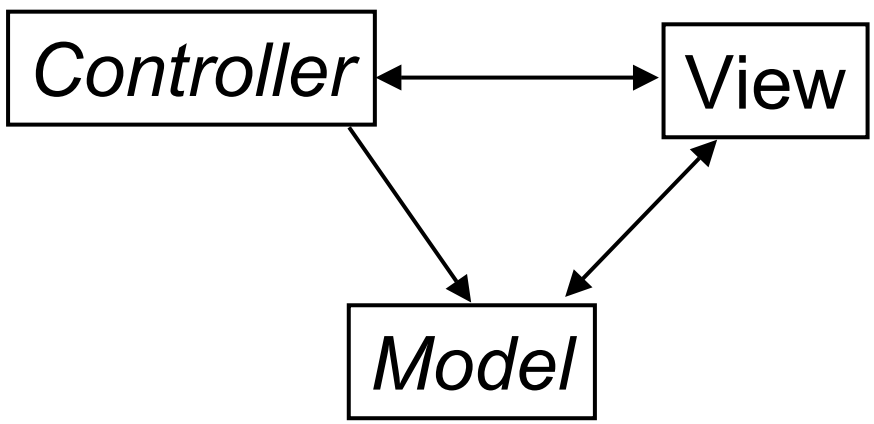
- Entity: classi di oggetti che rappresentano la semantica del dominio applicativo
- Boundary: classi di oggetti che rappresentano l'interfaccia tra gli attori e il sistema (il modello applicativo e il resto)
- Controller: classi di oggetti che determinano il modo in cui l'applicazione risponde agli input degli attori:
  - interpretazione delle richieste utente e loro passaggio alla logica applicativa

Boundary Control Entity

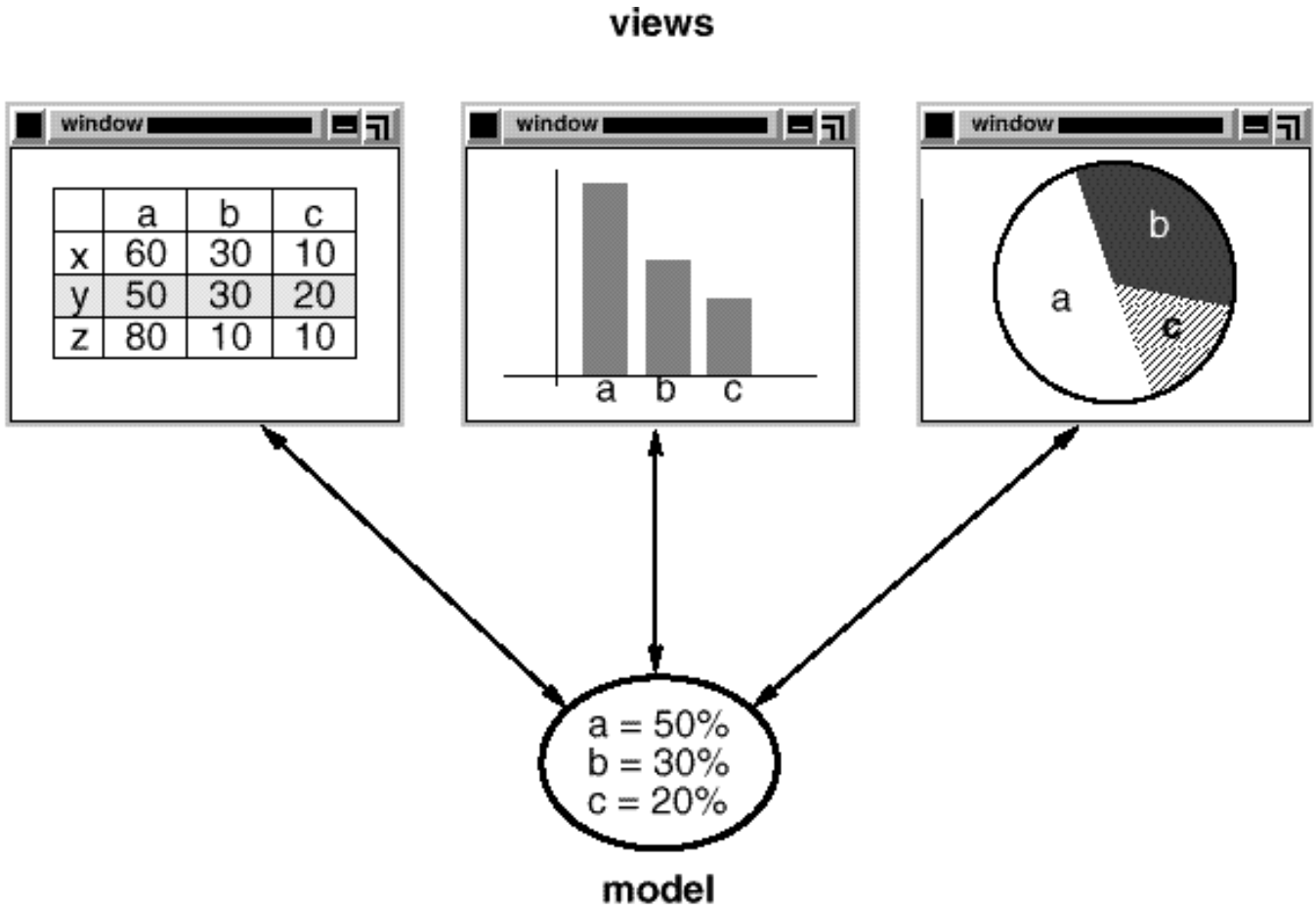


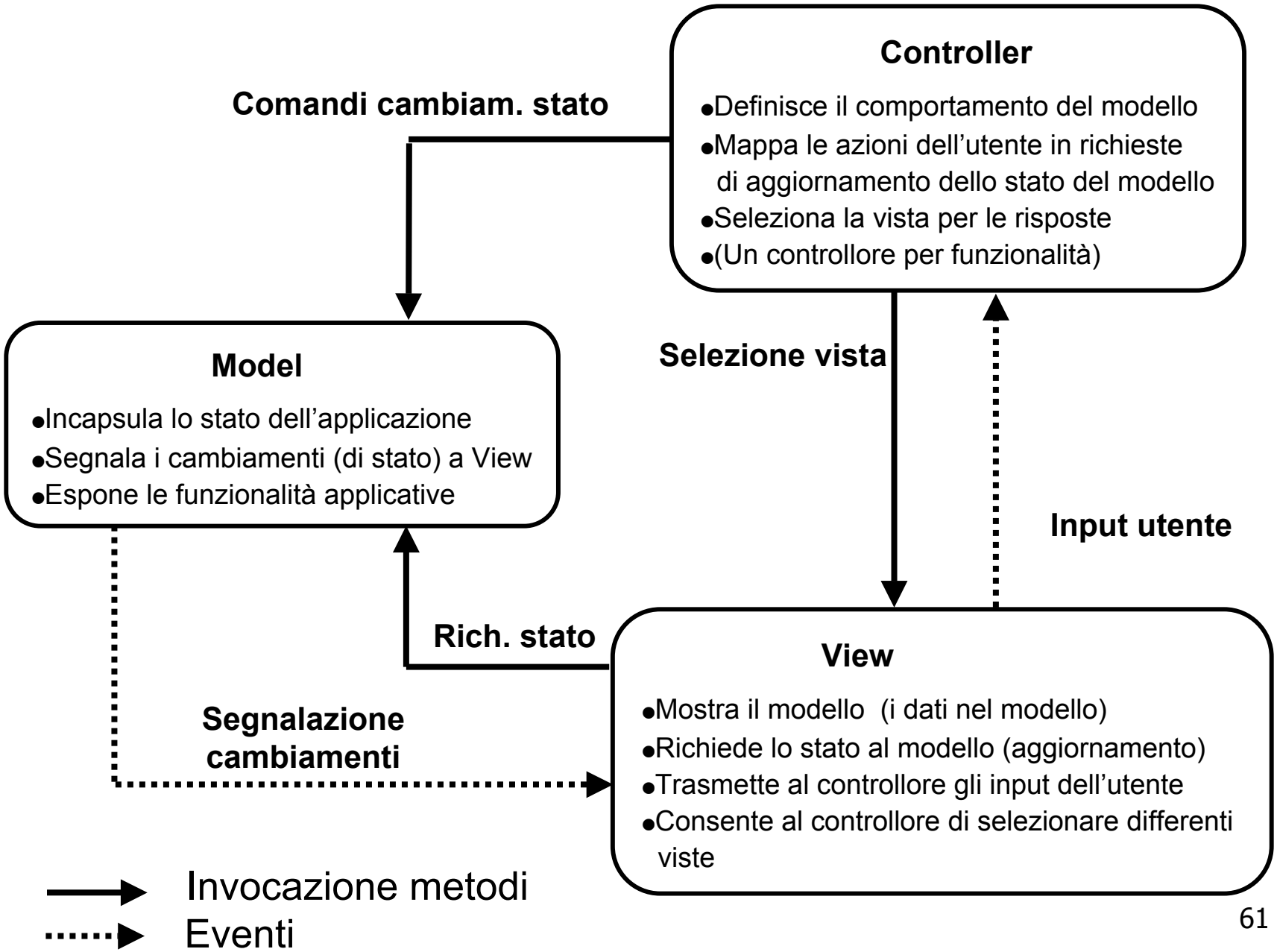
MVC Model-View-Controller

- Model: Oggetto applicativo
- View: Presentazione (viste)
- Controller: Determina il comportamento del modello e della presentazione (comandi)



Viste differenti

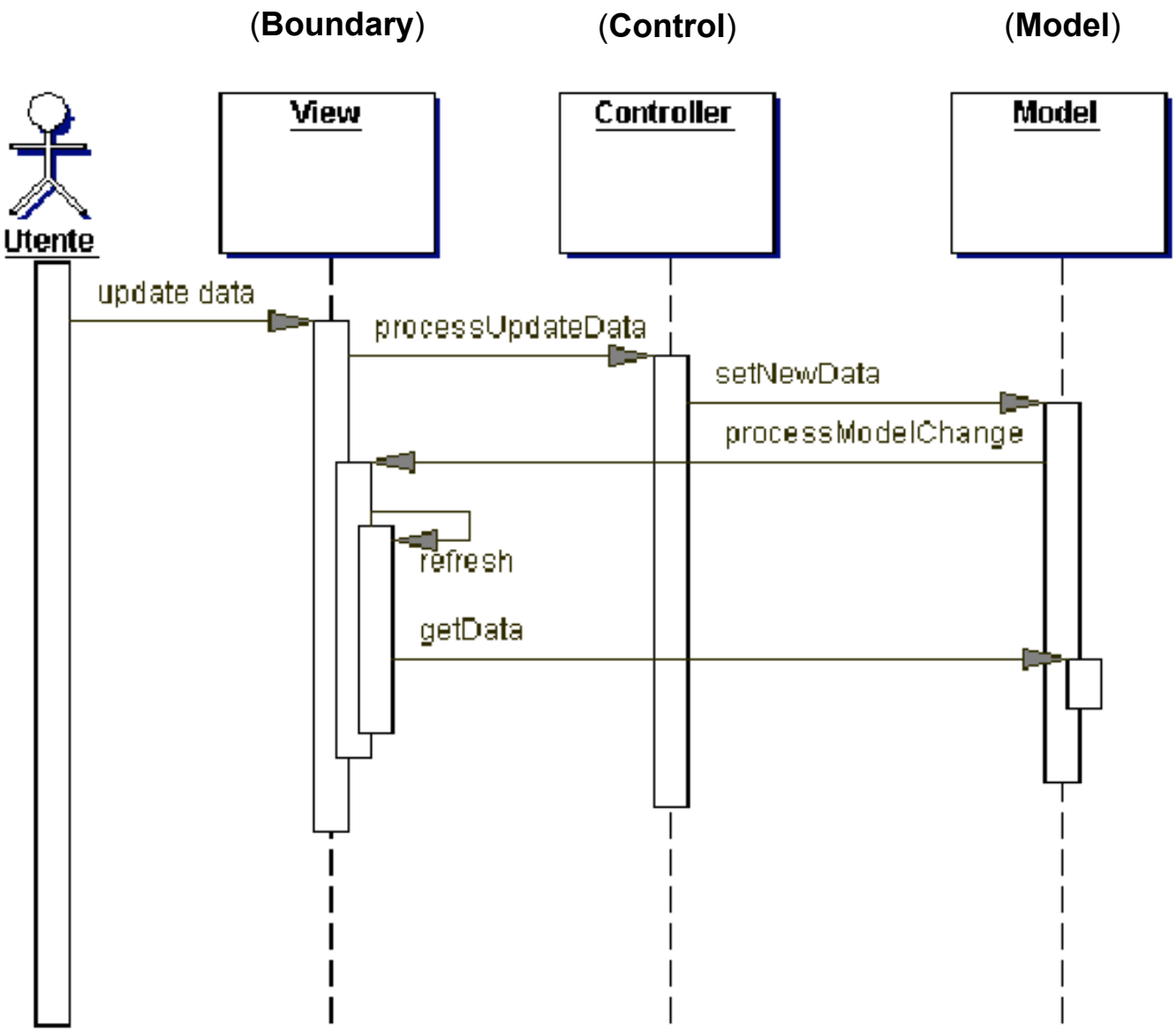



Controller
●Definisce il comportamento del modello
●Mappa le azioni dell'utente in richieste di aggiornamento dello stato del modello
●Seleziona la vista per le risposte
●(Un controllore per funzionalità)
Comandi cambiam. stato
Model
●Incapsula lo stato dell'applicazione
●Segnala i cambiamenti (di stato) a View
●Espone le funzionalità applicative
Selezione vista
Input utente
Rich. stato
Segnalazione cambiamenti
View
●Mostra il modello (i dati nel modello)
●Richiede lo stato al modello (aggiornamento)
●Trasmette al controllore gli input dell'utente
●Consente al controllore di selezionare differenti viste
Invocazione metodi
Eventi

MVC

- E' esso pure un ***Pattern*** di progettazione (di livello molto alto)
- Disaccoppia le tre categorie di oggetti
  - Evita i grovigli
- Tra View e Model c'è un protocollo *publish/subscribe:* il modello che ha un cambiamento informa le viste ad esso relative; queste hanno la possibilità di aggiornarsi
  - Più viste per uno stesso modello

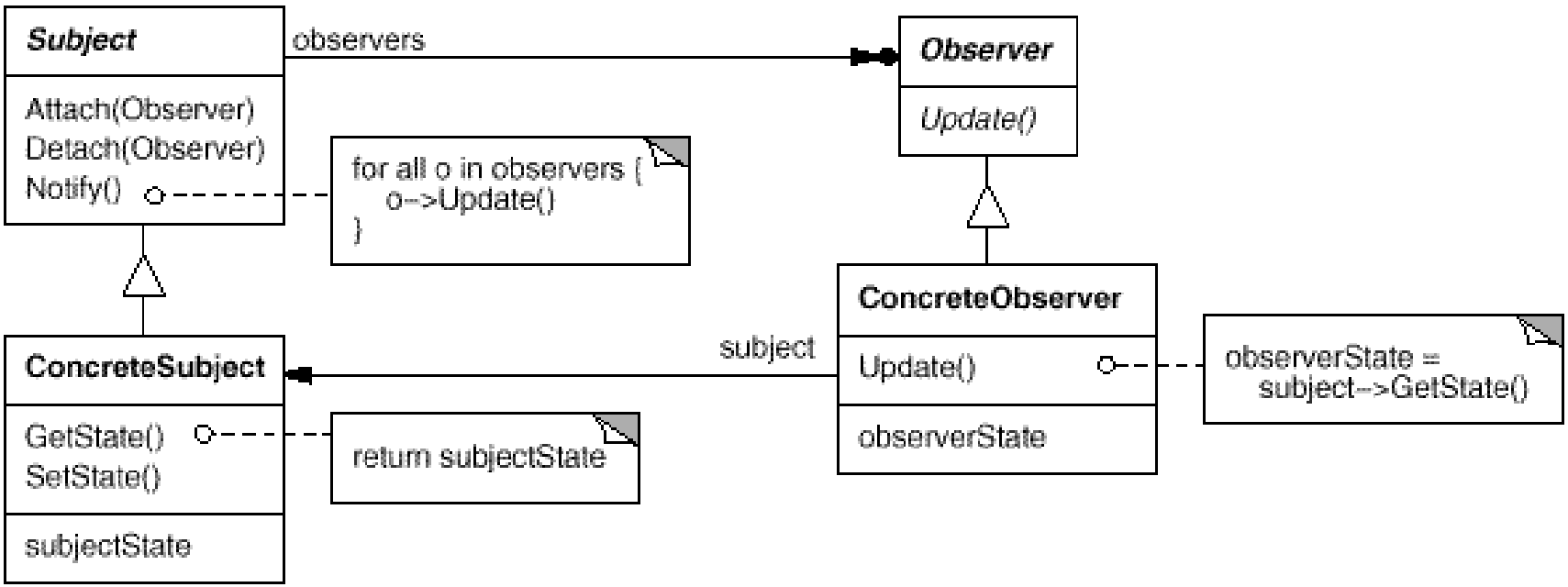
Dinamica


(Boundary)
(Control)
(Model)
View
Controller
Model
Utente
update data
processUpdateData
setNewData
processModelChange
refresh
getData

Pattern Observer

- Il Modello e la Vista implementano il pattern *publish & subscribe* detto anche *observer*
- Il pattern assume che l'oggetto (*Subject* o *Observable*) che contiene i dati sia separato rispetto agli oggetti (*Observer*) che presentano i dati
- Gli observer *osservano* i cambiamenti nel subject

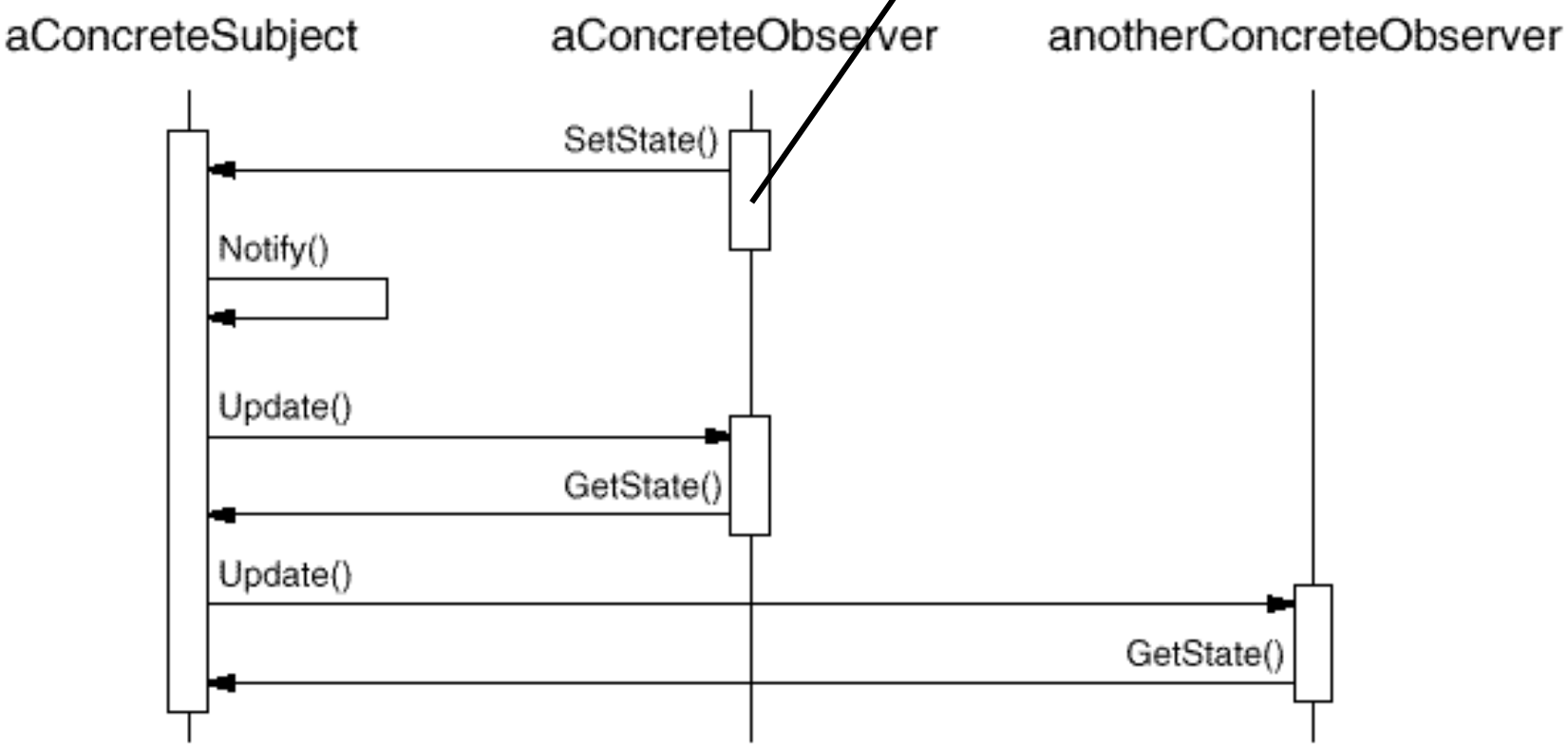
Struttura



GOF

Dinamica

Col modello MVC SetState() è chiamato presumibilmente dal controller



In Java

Osservatore

- Deve implementare l'interfaccia **`Observer`**
- L'interfaccia ha il solo metodo

```
update(Observable obj, Object arg)
```

esso viene chiamato quando l'oggetto osservato cambia

- I due parametri individuano l'oggetto che è cambiato e gli argomenti passati all'observer

In Java

Osservato

- Un oggetto “osservato” si ottiene subclassando **`java.util.Observable`**
- Eredita i metodi

  `addObserver(Observer obj)`
  - permette la registrazione degli osservatori

  `notifyObservers(Object arg)`
  - chiama il metodo **`update`** degli osservatori registrati passando l’identità del soggetto e gli argomenti
  - l’ordine di chiamata degli osservatori non è definito

Esempio

- **Osservatore:**

```
public class Osservatore implements Observer{
void update(Observable obj, Object arg) {
//implementazione del metodo
}
.......
.......
Osservato.addObserver(this); //registrazione
}
```

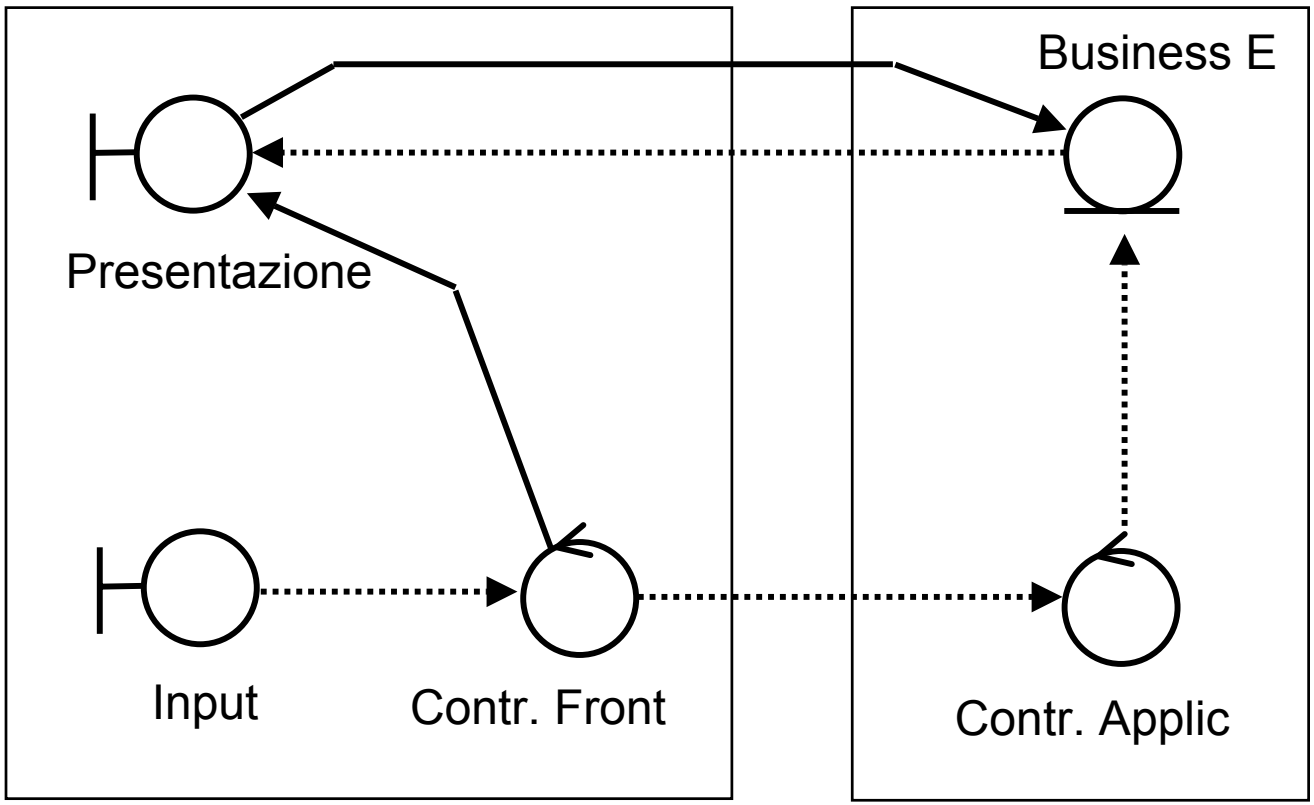
- **Osservato:**

```
public class Osservato extends Observable{
\\eredita addObserver e notifyObservers
}
```

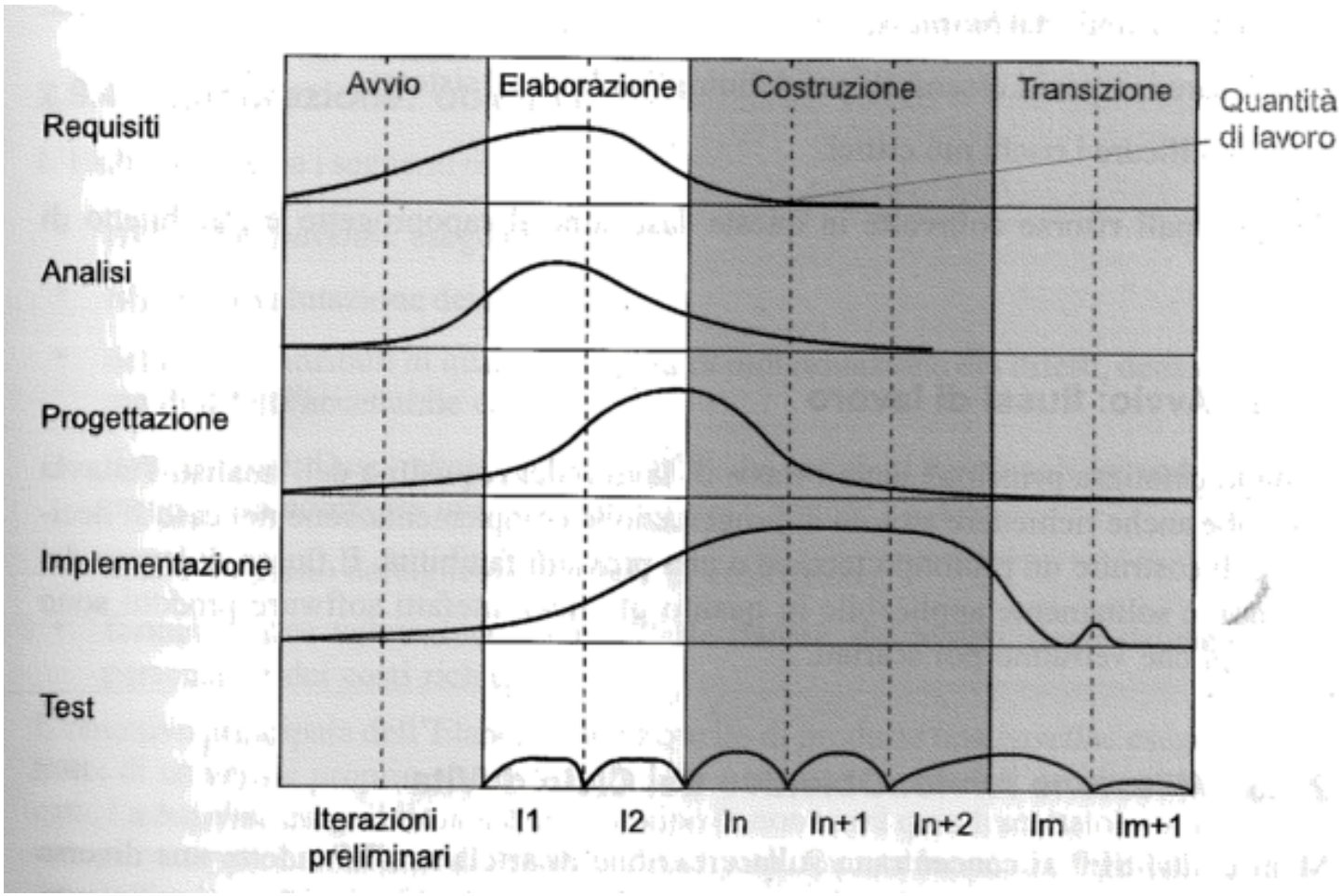
Lavoro individuale

- Esaminare la documentazione Java (le API)
- Verificare come si costruisce il pattern Observer
  - in particolare verificare i metodi della classe **`Observable`**
- Fare programmino di prova

Un sistema distribuito



UP


Avvio
Elaborazione
Costruzione
Transizione
Quantità di lavoro
Requisiti
Analisi
Progettazione
Implementazione
Test
Iterazioni preliminari
I1
I2
In
In+1
In+2
Im
Im+1

Processo UP

- Requisiti
  - Costruire i casi d'uso
  - Descrivere gli scenari (flussi) nei casi d'uso
- Analisi
  - Per ogni caso d'uso individuare le **classi di analisi** (class entity) e le loro relazioni
  - Per ogni caso d'uso individuare una/la realizzazione (le interazioni tra gli oggetti che danno luogo al comportamento previsto nel caso d'uso)

Processo Up (segue)

- Progettazione
  - Definire le **classi di progettazione**
    - Dettagliare le classi di dominio (*entity)*
    - Identificare eventuali classi non di dominio
    - Identificare le classi *boundary* e *control*
  - Definire una realizzazione di ogni caso d'uso con le classi di progettazione