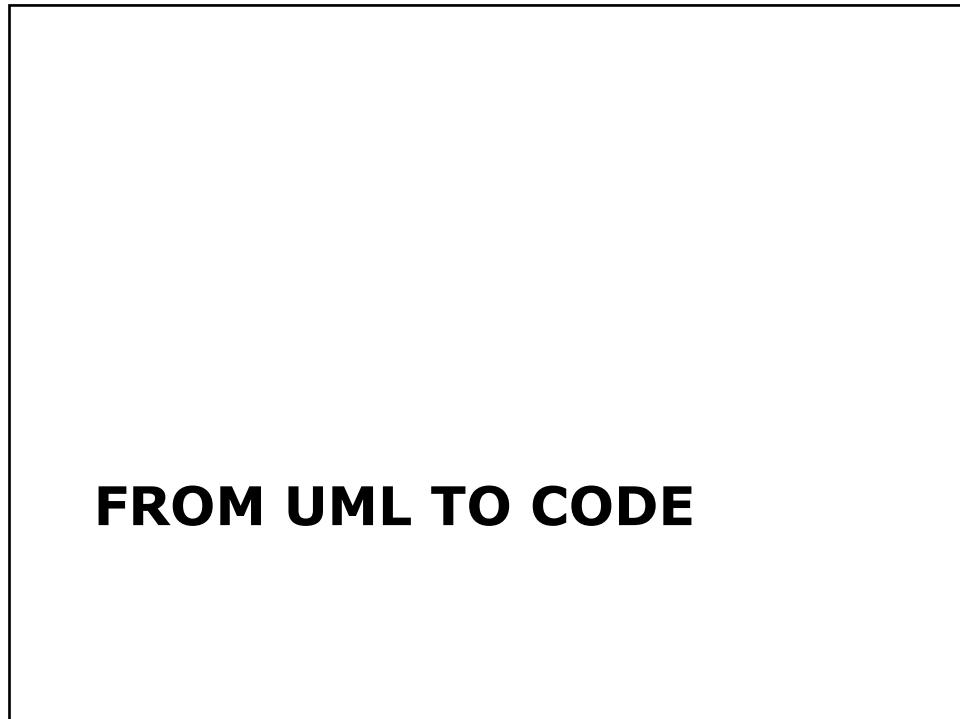




1



2

1

Chiaramento da lab 2 (1)

Come estrarre le classi? (3)

FASE 2: DIAGRAMMA DELLE CLASSI

- Identificare le classi
- Inserire gli attributi della classe indicando **nome, tipo e se:**
 - Pubblico +
 - Privato -
 - Protetto #
 - Ristretto al package ~
- Inserire i metodi indicando il **nome, i parametri e il tipo di ritorno**
- Aggiungere le associazioni indicando il **nome e la molteplicità.**
- Anche **ruolo, visibilità e navigabilità.**

3

Chiaramento da lab 2 (2)

Diagramma delle classi: associazione

- Le **associazioni tra classi** modellano possibili relazioni/connesioni tra oggetti (istanze) delle classi
 - Se gli attributi e i metodi hanno specificato la giusta visibilità, gli oggetti istanza delle classi in relazione possono accedere ai reciproci attributi o metodi

4

2

Chiarimento da lab 2 (3)

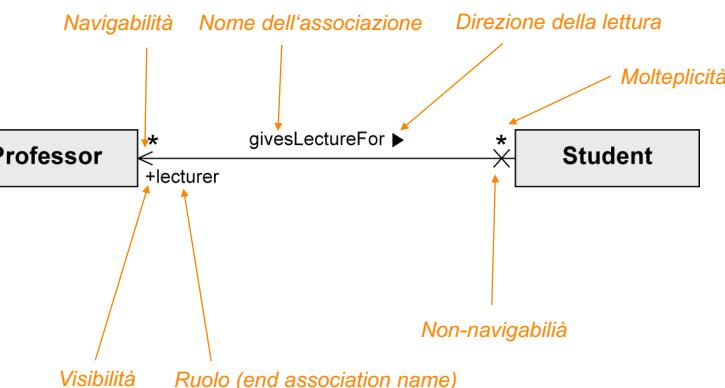
Proprietà di una associazione:

- **Nome**
 - in genere un verbo posto al centro della linea
- **Direzione dell'associazione**
 - un triangolino nero ▶ o ◀, per la "lettura" dell'associazione
- **Navigabilità**
 - se non espressa è bidirezionale
 - indica che l'oggetto di una classe "conosce" gli oggetti dell'altra classe (e se può accedere ai suoi attributi e metodi pubblici)
 - da non confondere con la direzione dell'associazione!
- **Molteplicità**
 - definisce il numero di istanze che prendono parte alla relazione
- **Ruolo (aka End association name, con visibilità)**
 - definisce il ruolo svolto dalla classe nell'associazione (utile in caso di associazioni multiple tra due classi)

5

Chiarimento da lab 2 (4)

Esempi:

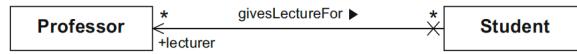


6

3

Chiaramento da lab 2 (5)

Esempi:

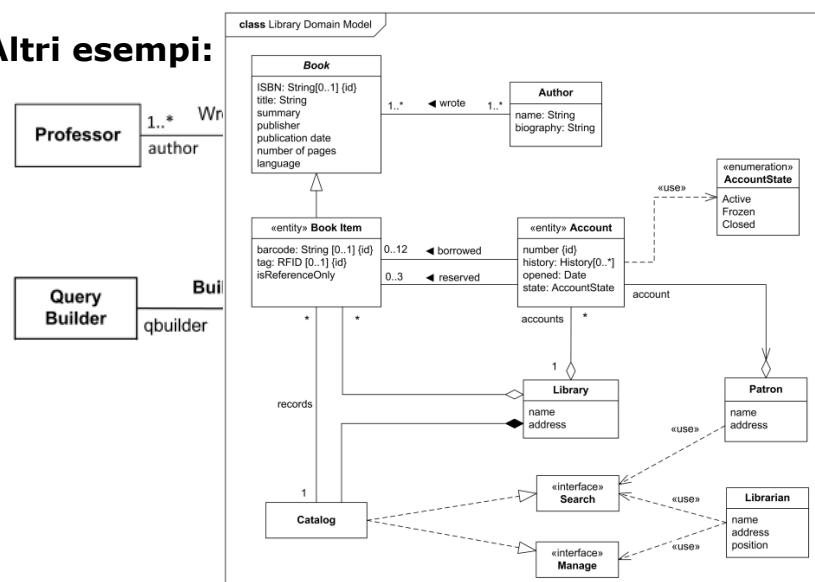


- Il professore fa lezione per gli studenti (e non viceversa)
- Il ruolo del professore è quello di *lecturer*
- Gli studenti possono accedere agli attributi e ai metodi (pubblici) del professore, ma non viceversa
- Più professori fanno lezione a più studenti, più studenti seguono le lezioni di più studenti

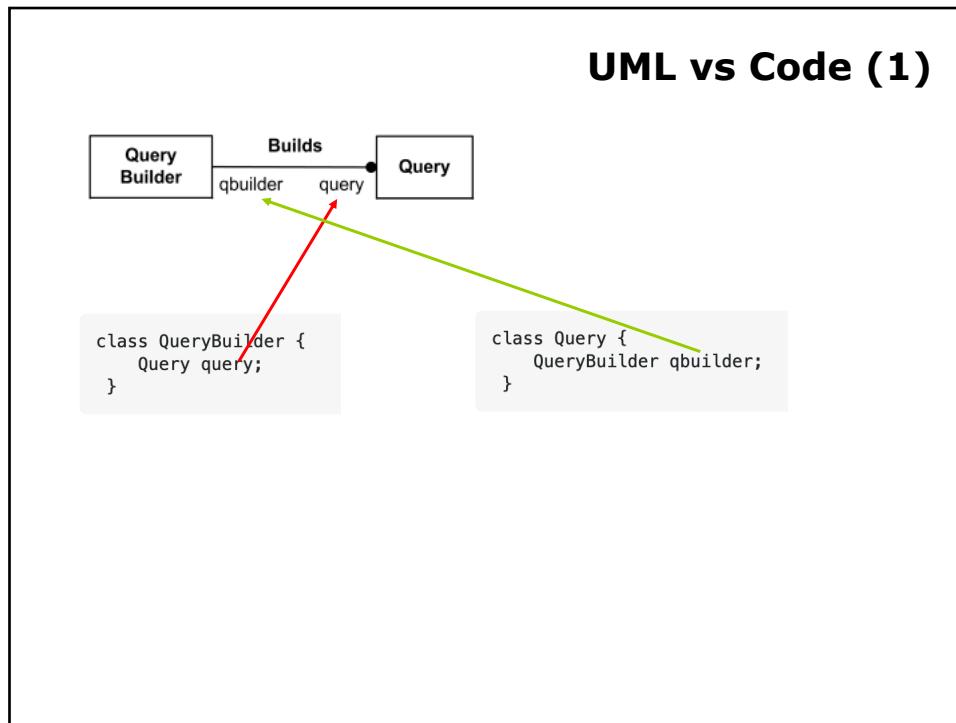
7

Chiaramento da lab 2 (6)

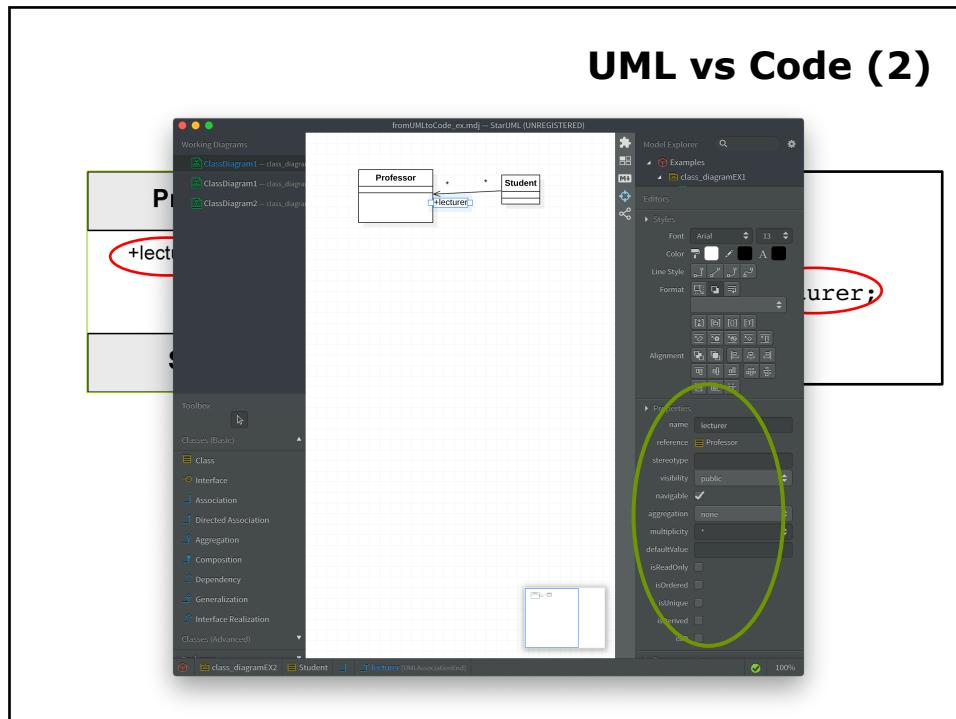
Altri esempi:



8

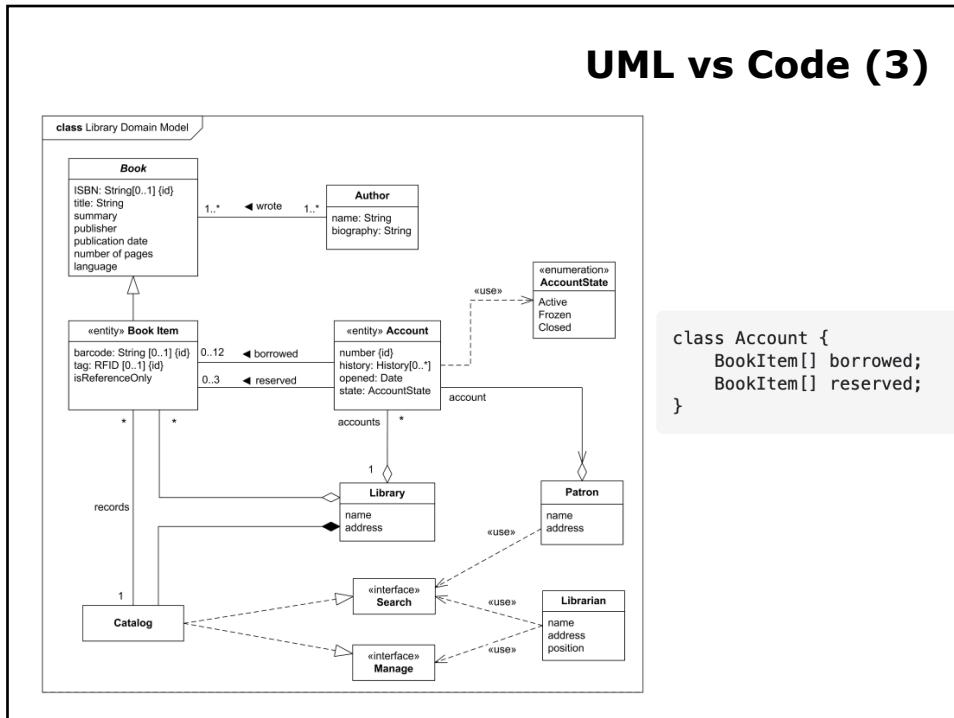


9

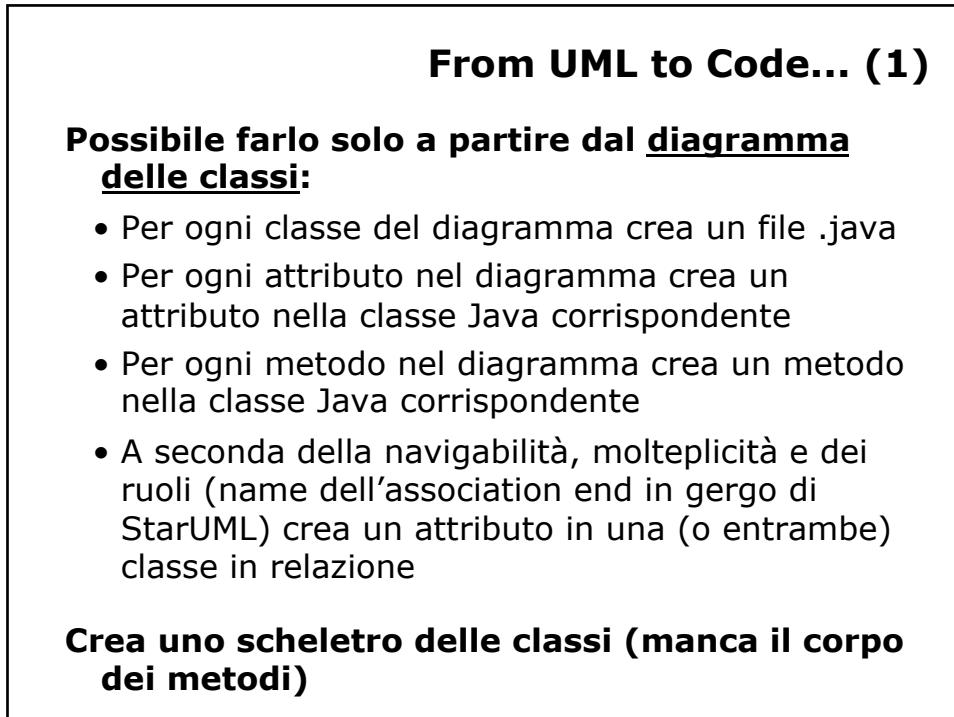


10

5



11



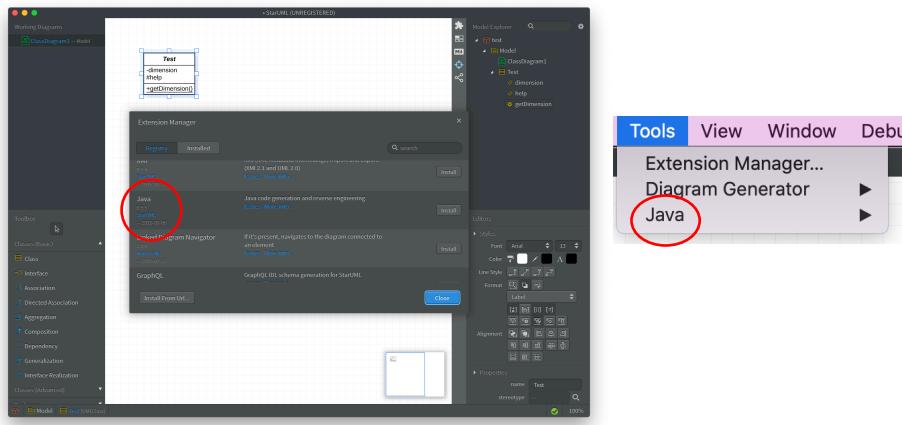
12

From UML to Code... (2)

In StarUML è necessario un plug-in aggiuntivo che va installato:

<https://staruml.io/extensions>

<https://github.com/staruml/staruml-java>



13

From UML to Code... (3)

Esempi pratici usando StarUML...

Lesson learned:

- più sono i dettagli nel diagramma delle classi, più le classi sono dettagliate (tipo di ritorno di un metodo, tipo degli attributi, quali attributi...)

14

...e viceversa

Dal codice a UML (usando Eclipse)

- è necessario installare un plug-in aggiuntivo:
 - Nome: ObjectAid UML Explorer
 - Link: <http://www.objectaid.com/update/current>
- Installazione:

Menu: Help > Install New Software...

15

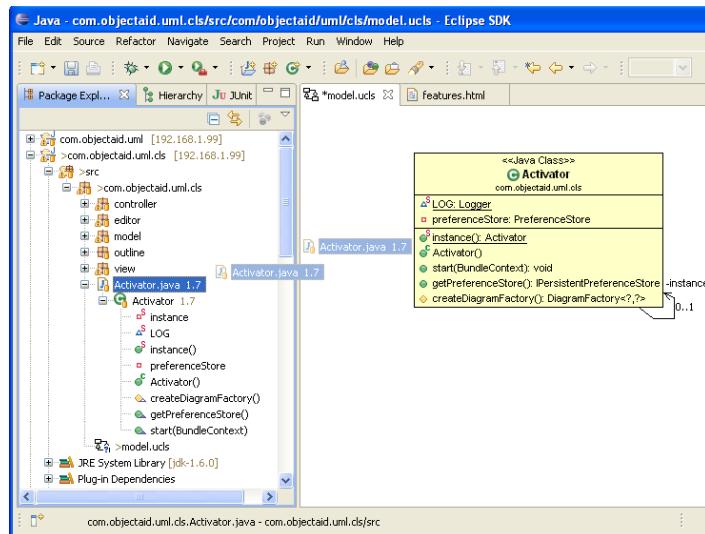
From Code to UML (1)

1. Creare un nuovo diagramma ObjectAid UML Diagram con il Wizard

16

From Code to UML (2)

2. Trascinare una o più classi dal Package Explorer



17

ALTRI DIAGRAMMI?

18

9

Macchine di stato (1)

- Una macchina di stato rappresenta il ciclo di vita di un oggetto
 - Gli **eventi** a cui è soggetto
 - Le **transizioni** che esegue al verificarsi di tali eventi
 - Quali **stati** raggiunge
 - stato = configurazione dell'oggetto tra il verificarsi di due eventi

19

Macchine di stato (2)

- In UML gli oggetti si distinguono in:
 - Oggetti dipendenti dallo stato (**statefull**)
 - Rispondono agli eventi in modo diverso a seconda dello stato in cui si trovano
 - Oggetti indipendenti dallo stato (**stateless**)
 - Rispondono agli eventi sempre nello stesso modo, indipendentemente dallo stato in cui si trovano
 - Le machine di stato vanno utilizzate per il comportamento di oggetti statefull

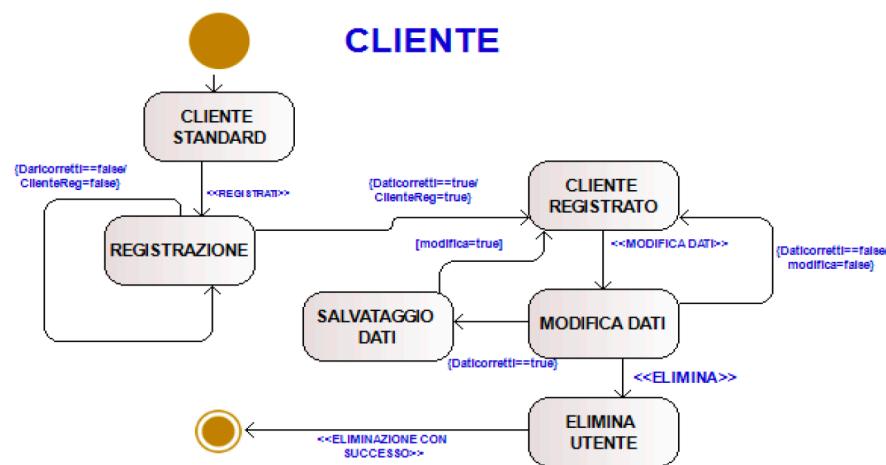
20

Descrizione del sistema di esempio

- Sistema per la *gestione di un'Agenzia di Autonoleggio* di medie dimensioni.
- L'applicazione dovrà servire sia il personale dell'Agenzia che gli utenti che desiderano noleggiare un'auto. Gli utenti possono essere sia utenti registrati che utenti non registrati
- Funzionalità di base:
 - Prenotare un'automobile, scegliendo la macchina a seconda del numero di posti e/o della cilindrata dalla lista delle macchine disponibili e selezionando la durata del noleggio.
 - Disdire una prenotazione.

21

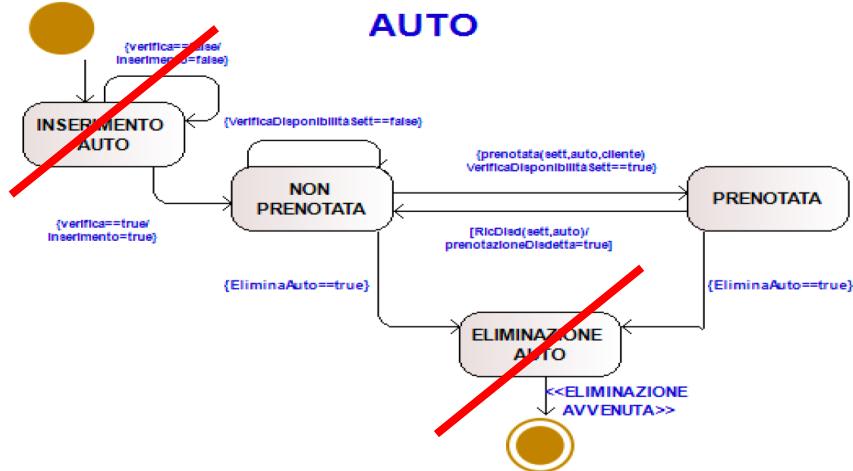
Macchine di stato by example (1)



Esempio errato!

22

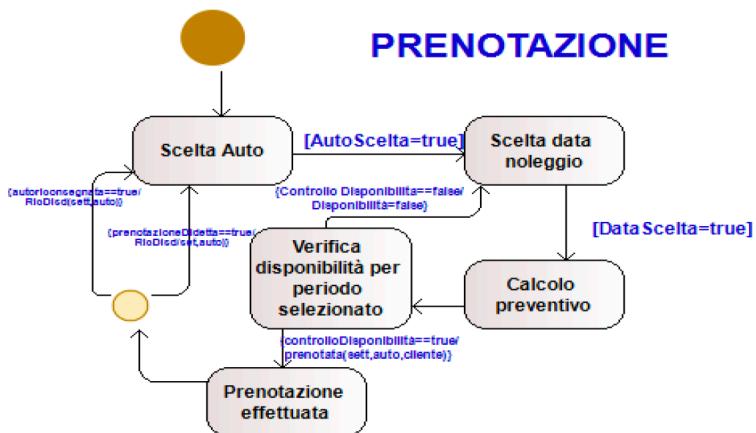
Macchine di stato by example (2)



Esempio errato!

23

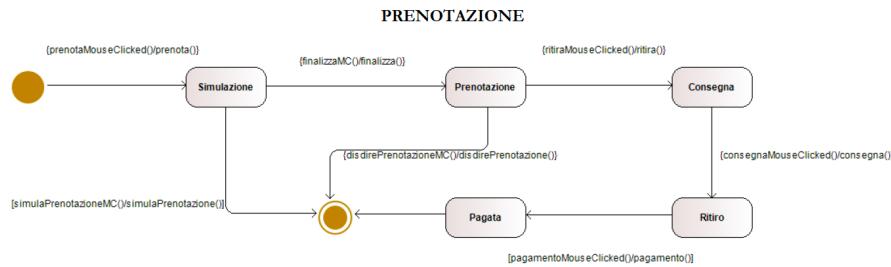
Macchine di stato by example (3)



Esempio errato! Macchina di stato confusa con il diagramma delle attività.

24

Macchine di stato by example (4)

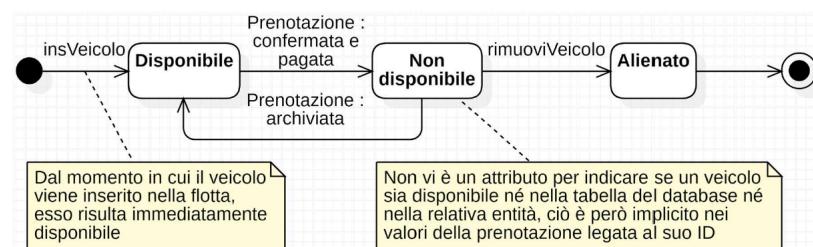


Meglio, ma migliorabile...

25

Macchine di stato by example (5)

2.5.3 Stati di un veicolo

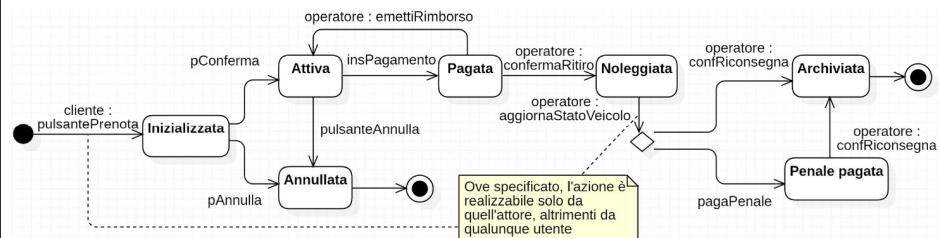


26

13

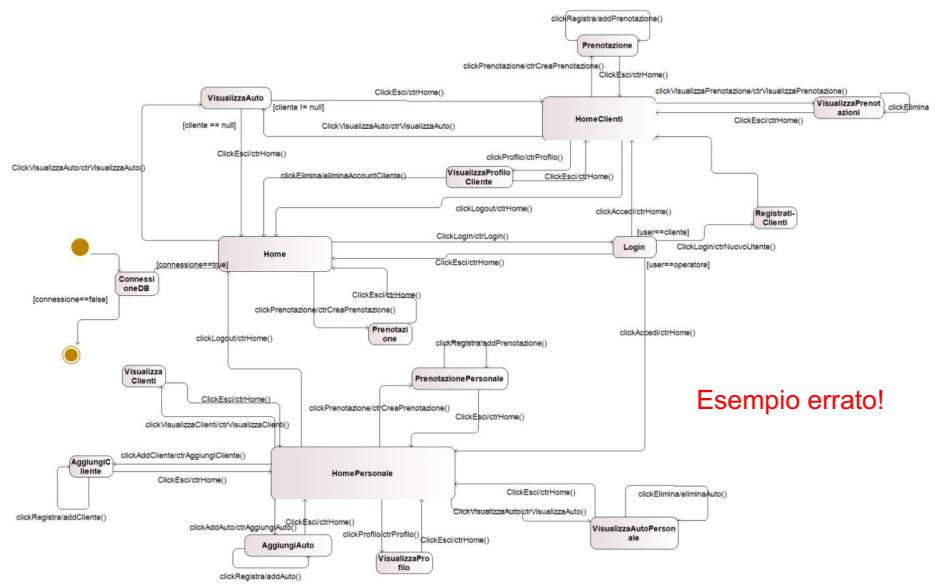
Macchine di stato by example (5)

2.5.2 Stati di una prenotazione



27

Macchine di stato by example (6)



28

Diagramma dei componenti (1)

- Diagramma dei componenti
 - Rappresentano un insieme di componenti, le loro interfacce (implementate e richieste) e le relazioni tra i componenti
- Un componente è una parte «significativa» di SW (spesso composti da più classi)
 - Una componente è una unità SW che può essere eseguita in modo indipendente dal resto del sistema e che fornisce servizi ad altre componenti ed usa i servizi di altre componenti
- Mostra i componenti (moduli) software e le loro interdipendenze
 - (eseguibili, librerie, ecc..)
- E' un grafo di componenti connessi da relazioni di dipendenza o di contenimento

29

Diagramma dei componenti (2)

Elementi grafici (1)

Elemento	Simbolo/Notazione	Spiegazione
Componenti ("component")		Simbolo per i moduli di un sistema (interazione e comunicazione avvengono tramite interfaccia)
Pacchetto		Un pacchetto riassume diversi elementi del sistema (per es. classi, componenti, interfacce) in un gruppo.
Artefatto		Gli artefatti sono unità d'informazione fisiche (per es. codice sorgente, file .exe, script, documenti) che vengono creati o di cui si ha bisogno nel processo di sviluppo o durante l'esecuzione di un sistema.

30

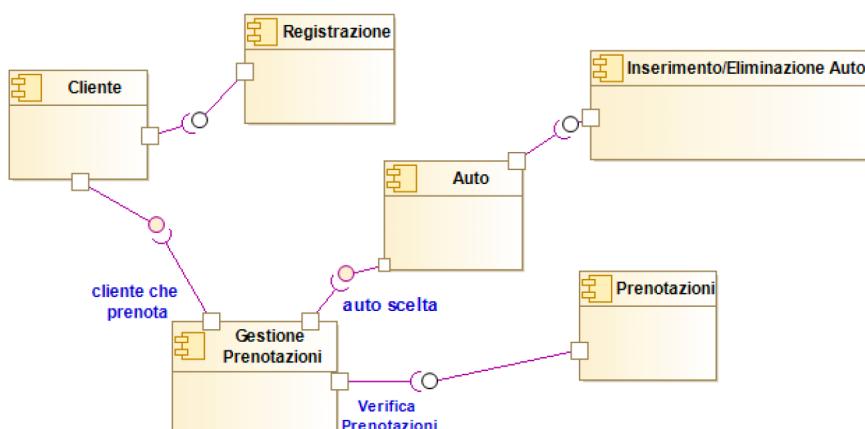
Diagramma dei componenti (2)

Elementi grafici (2)

Interfaccia offerta	○—	Simbolo di una o più interfacce definite in modo chiaro che mettono a disposizione funzioni, servizi o dati verso l'esterno (il semicerchio viene anche chiamato socket).
Interfaccia necessaria	—○	Simbolo di un'interfaccia necessaria che riceve funzioni, servizi o dati dall'esterno (il cerchio con notazione con bastoncino viene chiamato anche notazione lollipop).
Porta	□	Il simbolo rappresenta un punto di interazione separato tra un componente e il proprio ambiente.
Relazione	——	Le linee fungono da connettori e indicano le relazioni tra componenti.
Relazioni di dipendenza	----->	Connettori speciali per una relazione di dipendenza tra componenti del sistema (non viene sempre mostrata esplicitamente).

31

Diagramma dei componenti by ex. (1)

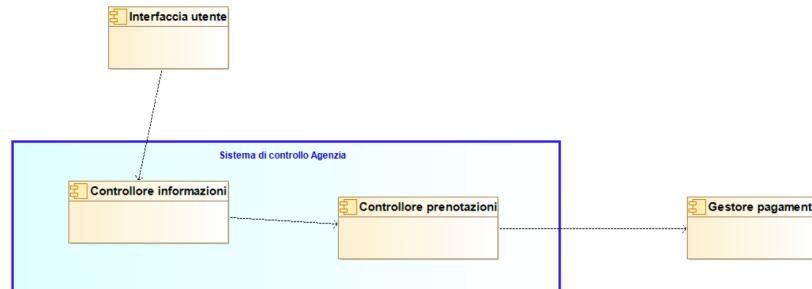


Esempio errato!

32

16

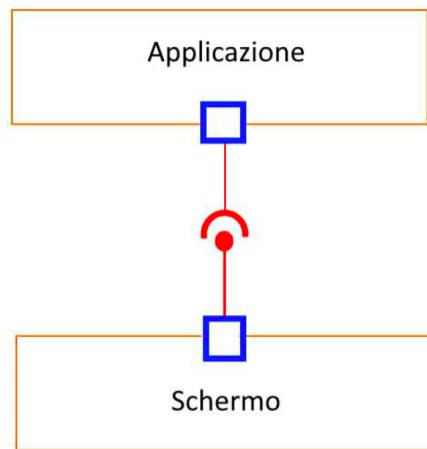
Diagramma dei componenti by ex. (2)



Migliorabile....

33

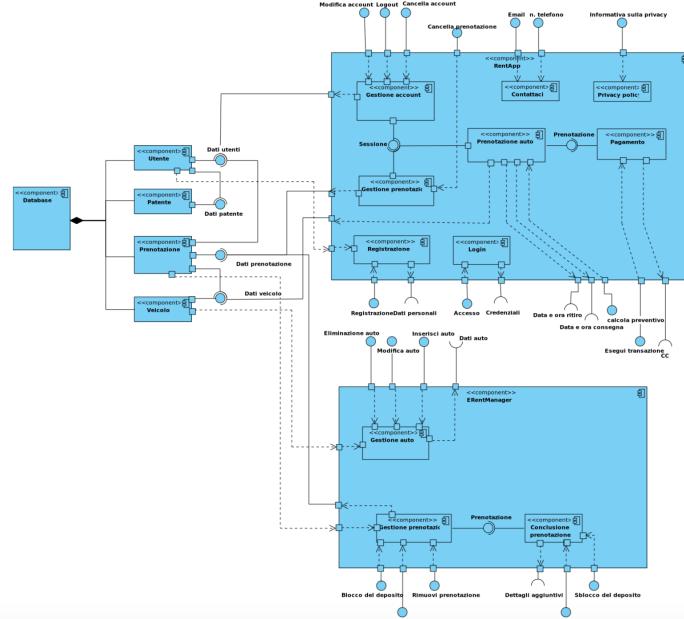
Diagramma dei componenti by ex. (3)



Esempio errato!

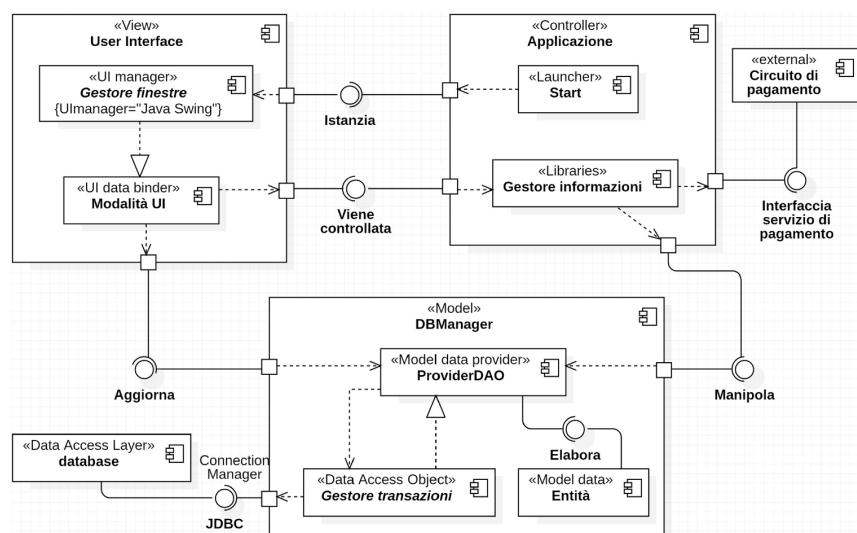
34

Diagramma dei componenti by ex. (4)



35

Diagramma dei componenti by ex. (6)



36

18

Diagramma di deployment (1)

- Permette di rappresentare, a diversi livelli di dettaglio, l'**architettura fisica** del sistema
- E' un grafo di **nodi elaborativi** in ambiente di esecuzione, connessi da associazioni di *comunicazione*
- Può mostrare i *componenti, processi e oggetti* ubicati in questi nodi

37

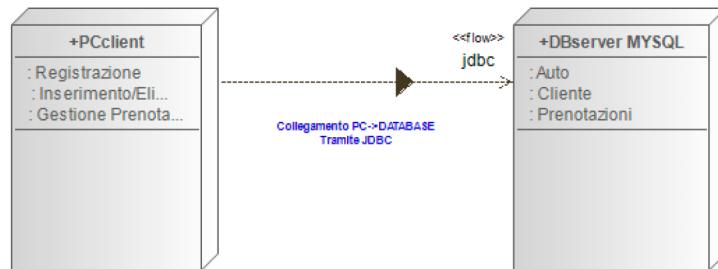
Diagramma di deployment (2)

- Un **nodo** è un **oggetto fisico** presente a *run-time* che rappresenta una risorsa di elaborazione avente generalmente almeno una *memoria* e spesso anche *capacità elaborative*
- Sono nodi i dispositivi di elaborazione ma anche le risorse umane e le risorse meccaniche di elaborazione
- I nodi possono contenere componenti e oggetti, ad indicare che essi vivono e vengono eseguiti su quel particolare nodo
- I **componenti**
 - non esistono come entità a run-time
 - non compaiono come nodi nei diagrammi di distribuzione ma vanno rappresentati in quelli dei componenti

38

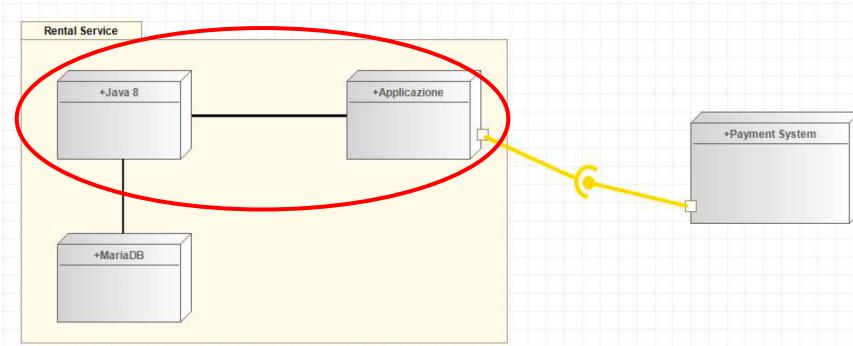
19

Diagramma di deployment by ex. (1)



39

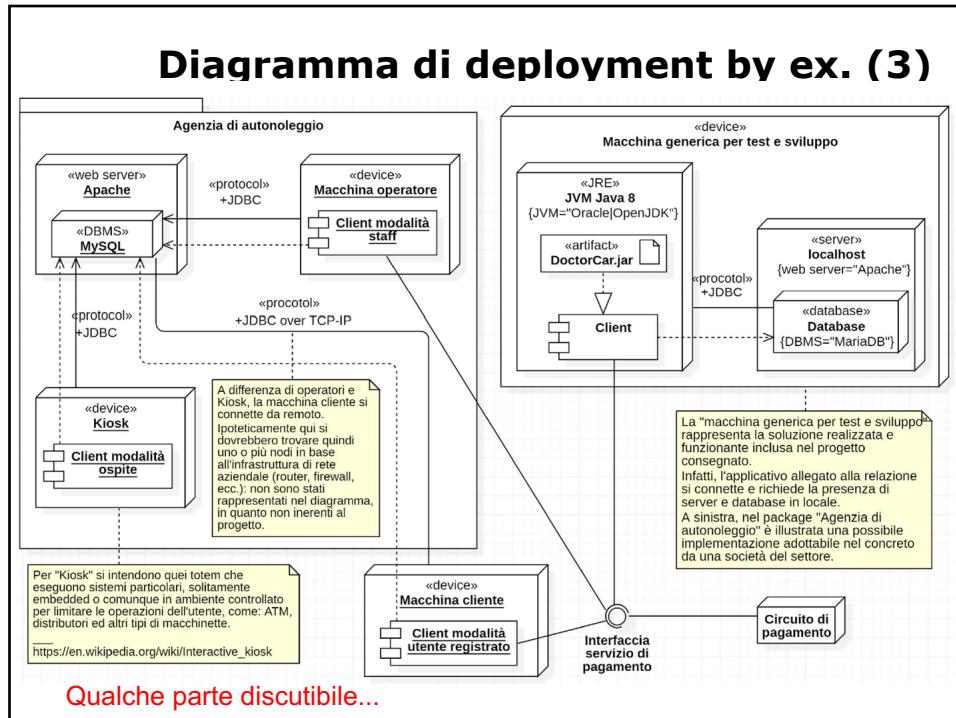
Diagramma di deployment by ex. (2)



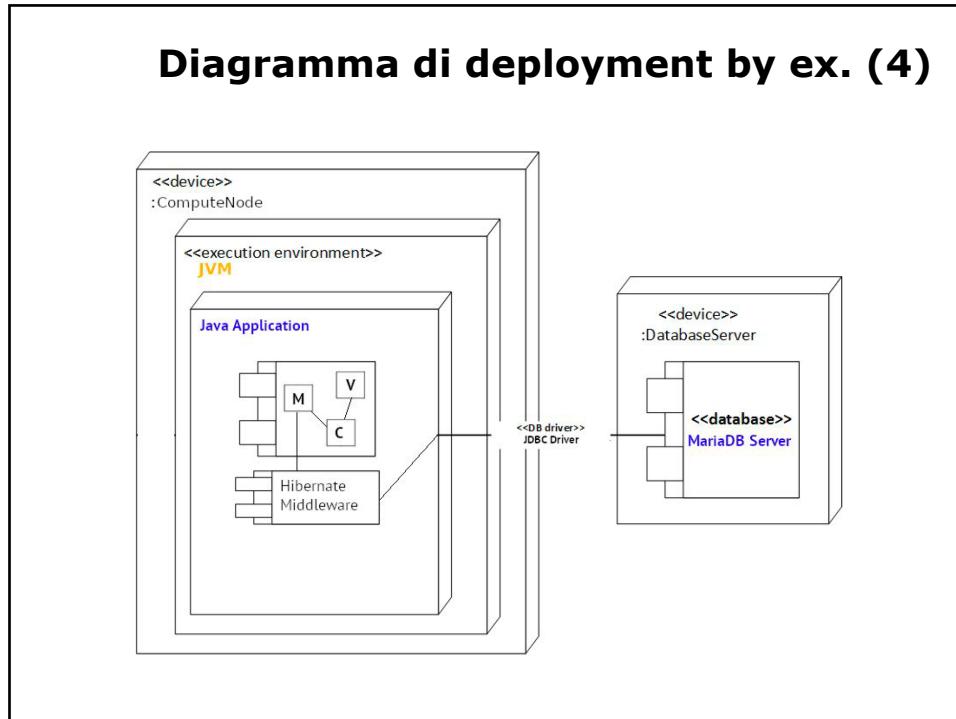
Esempio errato!

40

20



41



42

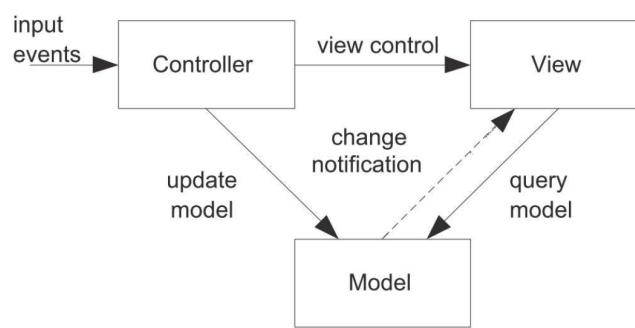
MVC PATTERN AGAIN

43

Design Pattern MVC (1)

Model-View-Controller (MVC) pattern

- Il pattern architettonale che aiuta a separare l'interfaccia utente (layer) dalla parte funzionale del sistema
 - Caso speciale di multi-layer

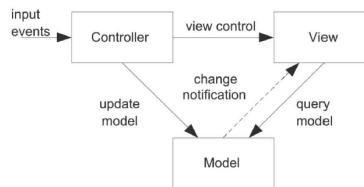


44

22

Design Pattern MVC (2)

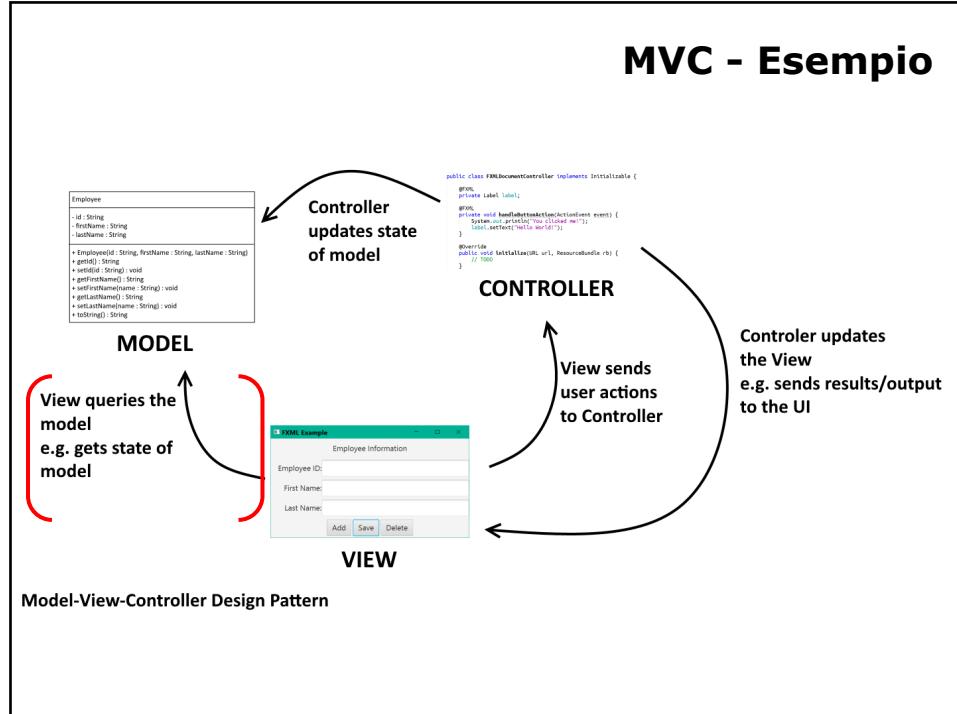
Model-View-Controller (MVC) pattern



- **Model** contiene le classi le cui istanze devono essere visualizzate e manipolate (layer funzionale)
- **View** contiene oggetti utilizzati per visualizzare i dati del model all' utente tramite interfaccia
- **Controller** contiene gli oggetti che gestiscono l'input dell'utente (a volte non distinta da view)

45

MVC - Esempio



46

23

Data-driven Software design (1)

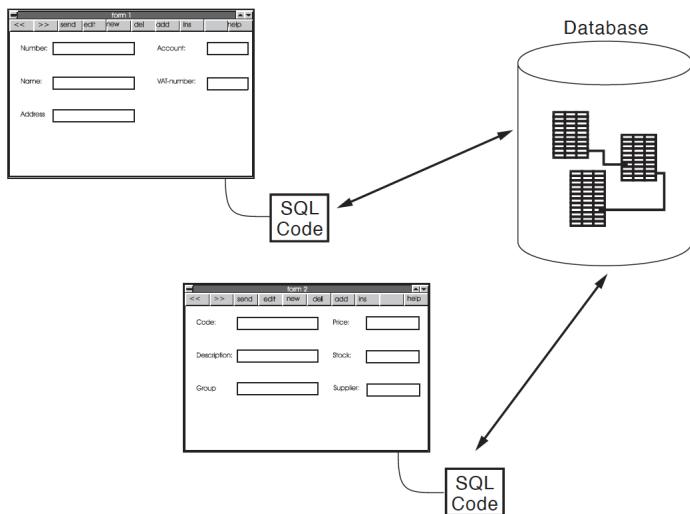


FIGURE 3.1 Data-driven software design. Many applications are designed as a collection of user interface forms with some SQL code attached to them. Because this approach depends heavily on the database layout, it can lead to programs that are very difficult to maintain.

47

Data-driven Software design (2)

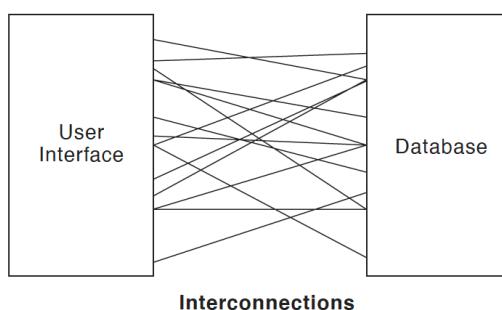


FIGURE 3.2 Data-driven programming. Direct mapping of the user interface on the database layout results in a web of interconnections. Every change in the database layout can result in drastic changes in the application code.

- Sviluppo sicuramente veloce
- Ma la manutenzione?

48

24

Data-driven Software design (3)

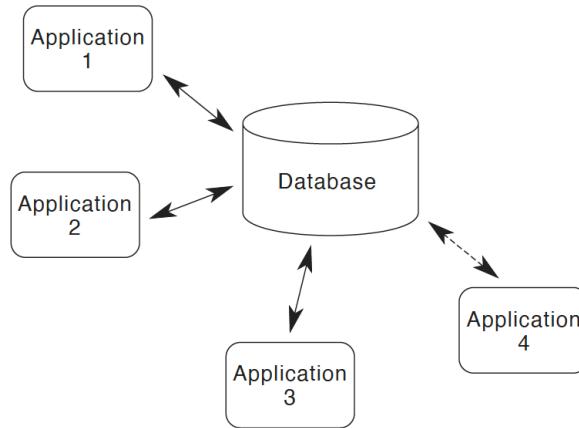


FIGURE 3.3 Supporting multiple applications. Adding a new application (*dotted line*) to the database often requires changes to the database schema. These changes may result in having to modify all of the other applications that operate on that database.

49

Object-Oriented Software design (1)

Application domain vs Problem/Business domain

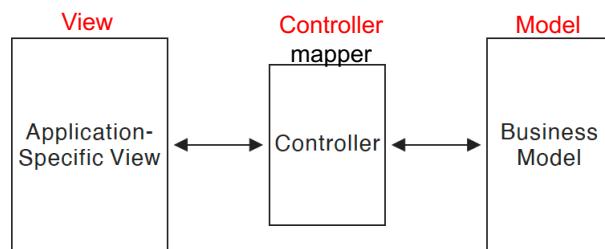


FIGURE 3.4 MVC architecture.

Business Model: what are the business entities or processes?
(Class diagram)

50

25

Object-Oriented Software design (2)

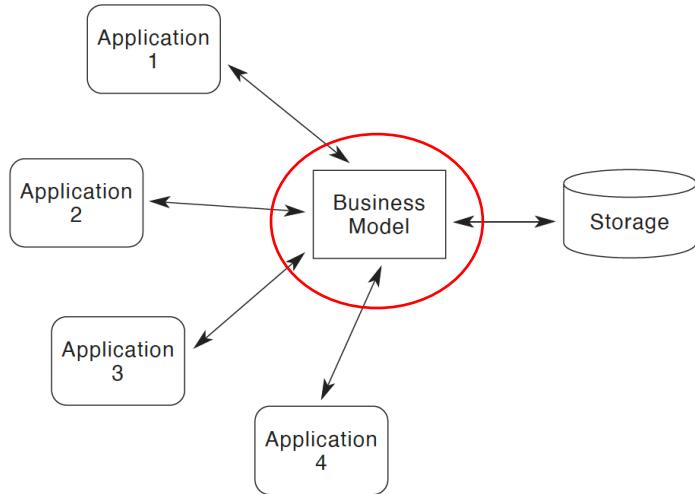


FIGURE 3.6 Business model-driven software design.

51

Object-Oriented Software design (3)

In presenza di Database:

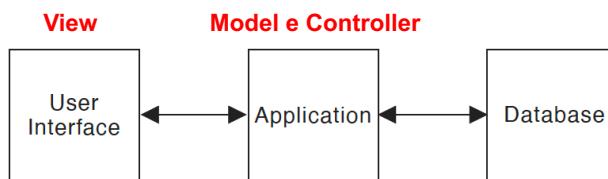


FIGURE 3.7 Applications consist of three main parts: the user interface, the actual application, and the data storage part.

52

26

Object-Oriented Software design (4)

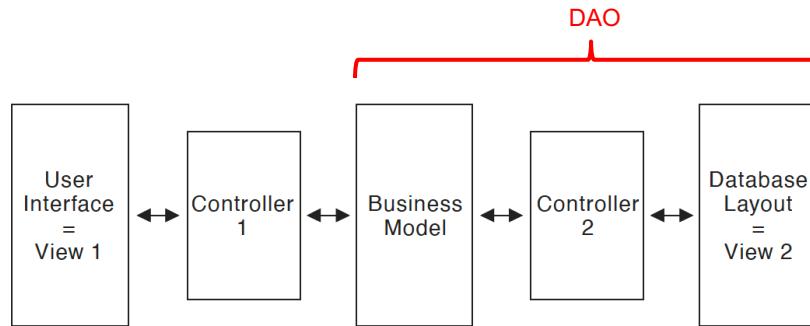
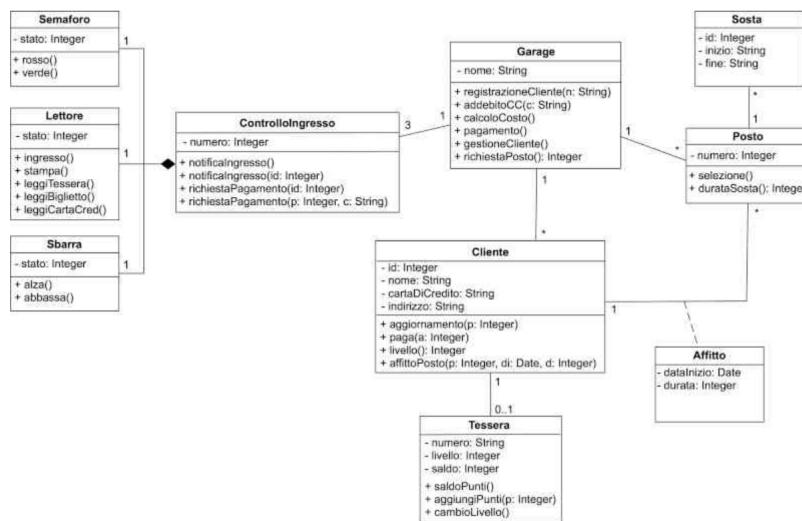


FIGURE 3.9 Analogy between user interface and database. Both the user interface and the database can be regarded as views on the business model. They both serve in presenting (storing) the information contained in the model.

53

Object-Oriented Software design (5)



54

27