

# INTRODUZIONE AI DESIGN PATTERN

## INGEGNERIA DEL SOFTWARE

Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica, A.A. 2015–2016

[alfredo.milani@unipg.it](mailto:alfredo.milani@unipg.it)



# SOMMARIO



- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- Utilizzare i design pattern

# SOMMARIO

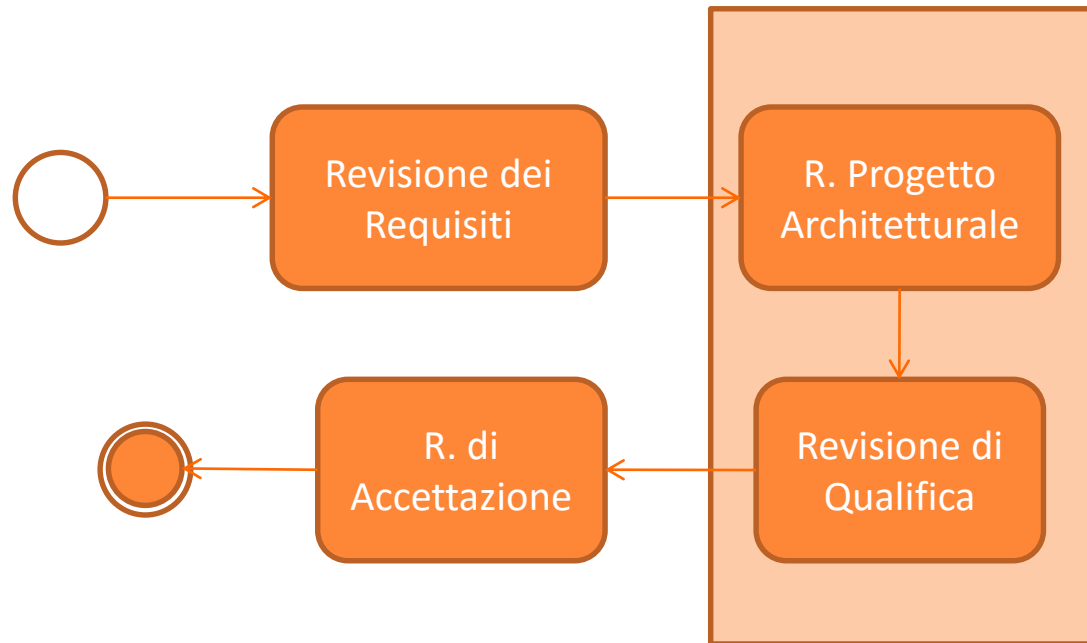


- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- Utilizzare i design pattern

# DESIGN PATTERN



- Specifica Tecnica, Definizione di Prodotto, Piano di Qualifica



# INTRODUZIONE



## ○ Progettare software *object oriented* è difficile

- Riusabilità
  - Ci vuole molta **esperienza**
- Soluzioni comuni e ricorrenti (*pattern*)
  - Flessibili, eleganti, riusabili

## ○ Non solo progettazione di *software*

- Macbeth, novelle romantiche

## ○ Altruismo ...

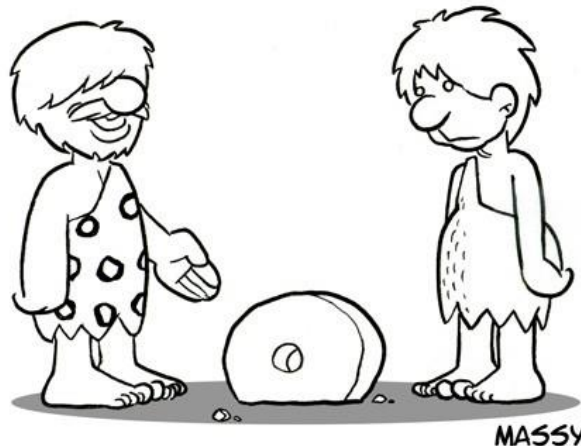
- Si tende a tenere per se le soluzioni vincenti!



## ○ Design Pattern

- Raccolta dell'**esperienza** dei progettisti, presentata in un modo effettivamente **utilizzabile**
  - Riutilizzo di soluzioni architetture vincenti
- **Nessuna** soluzione **nuova** (o non documentata)
- Non vogliamo riscoprire la ruota?!

- HO INVENTATO LA RUOTA SGONFIA.



# SOMMARIO



- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- Utilizzare i design pattern

# COS'È UN DESIGN PATTERN



## Definizione

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”

*Christopher Alexander*

## ○ Quattro elementi essenziali

- **Nome** del *pattern*
  - Vocabolario di progettazione
- Il **problema** che il *pattern* risolve
  - Descrizione del contesto
- La **soluzione**
  - Elementi, relazioni, responsabilità e collaborazioni
- Le **conseguenze**
  - Risultati e limiti della soluzione

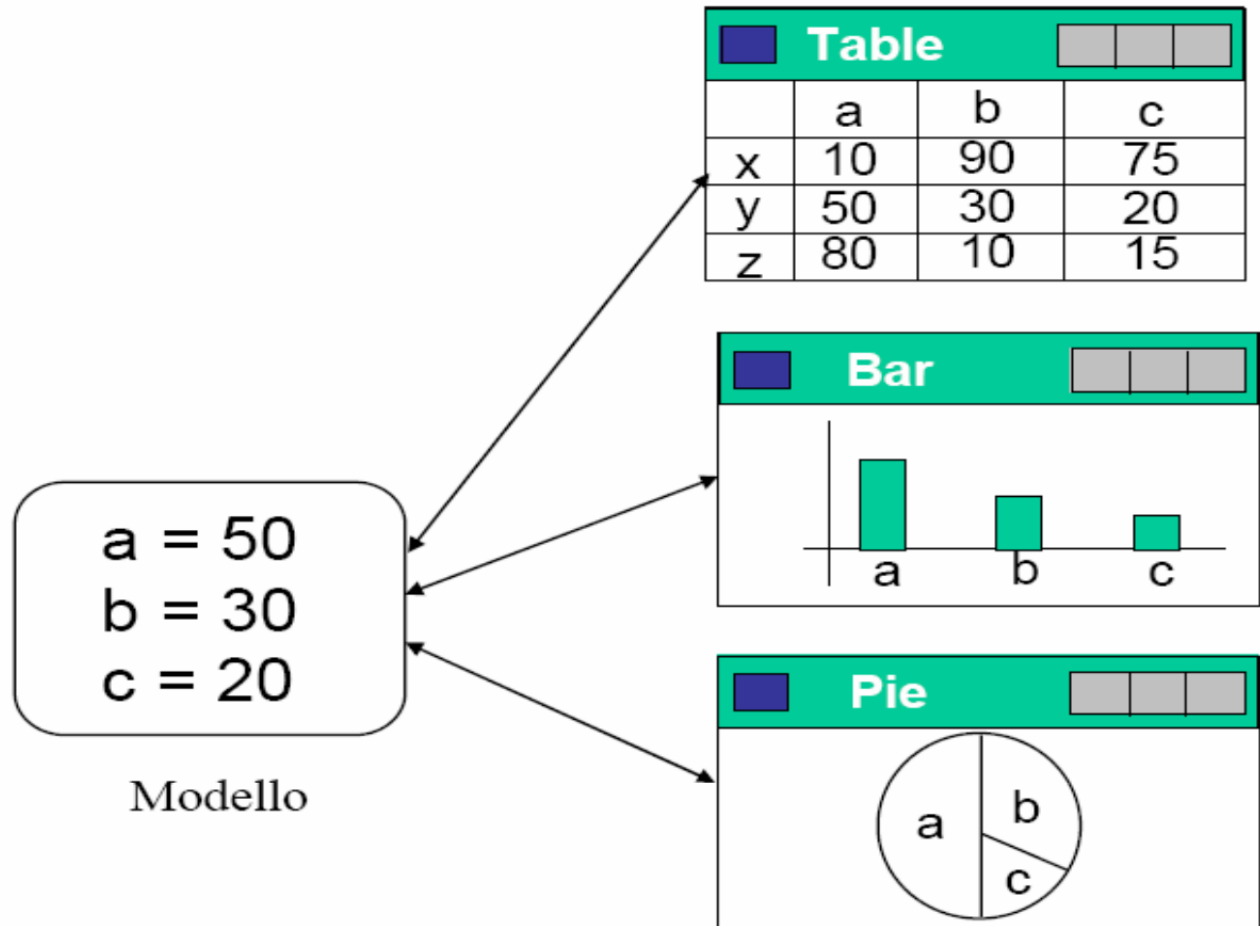


# DESIGN PATTERN: ESEMPIO MVC



## ○ Problema: sincronizzazione tra le viste di un modello

- Esistono più viste su uno stesso insieme di informazioni
- Agendo su ciascuna vista è possibile modificare il modello
- Come mantenere il sincronismo fra le viste e il modello quando vengono effettuate modifiche?



# DESIGN PATTERN: ESEMPIO MVC



- ❑ Per capire in pratica di che si tratta, consideriamo le classi Model, View e Controller (MVC) introdotte nel 1988 per costruire interfacce in Smalltalk-80.
- ❑ Model
  - ▶ rappresenta l'oggetto (il dato) vero e proprio, funzionale all'applicazione
  - ▶ è indipendente dalle specifiche rappresentazioni su schermo
  - ▶ è indipendente dalle modalità di input dei dati da parte dell'utente
- ❑ View
  - ▶ la rappresentazione dell'oggetto sullo schermo dell'utente
  - ▶ Una view ottiene i dati per la presentazione dal Model. Possono esserci (e tipicamente esistono) viste multiple dello stesso modello

# DESIGN PATTERN: ESEMPIO MVC



## □ Controller

- ▶ insieme di regole che stabiliscono le reazioni della presentazione sullo schermo in relazione all'input dei dati da parte dell'utente
- ▶ Ciascuna view ha associato un componente controller, il cui compito è
  - ◆ ricevere gli input dell'utente (tipicamente come eventi a seguito di operazioni con mouse o tastiera - Es. pressione di un pulsante)
  - ◆ traduce gli eventi in richieste di servizio per il modello o la vista cui è associato
- ▶ L'utente interagisce con il sistema solo ed esclusivamente attraverso il controller

# DESIGN PATTERN: ESEMPIO MVC

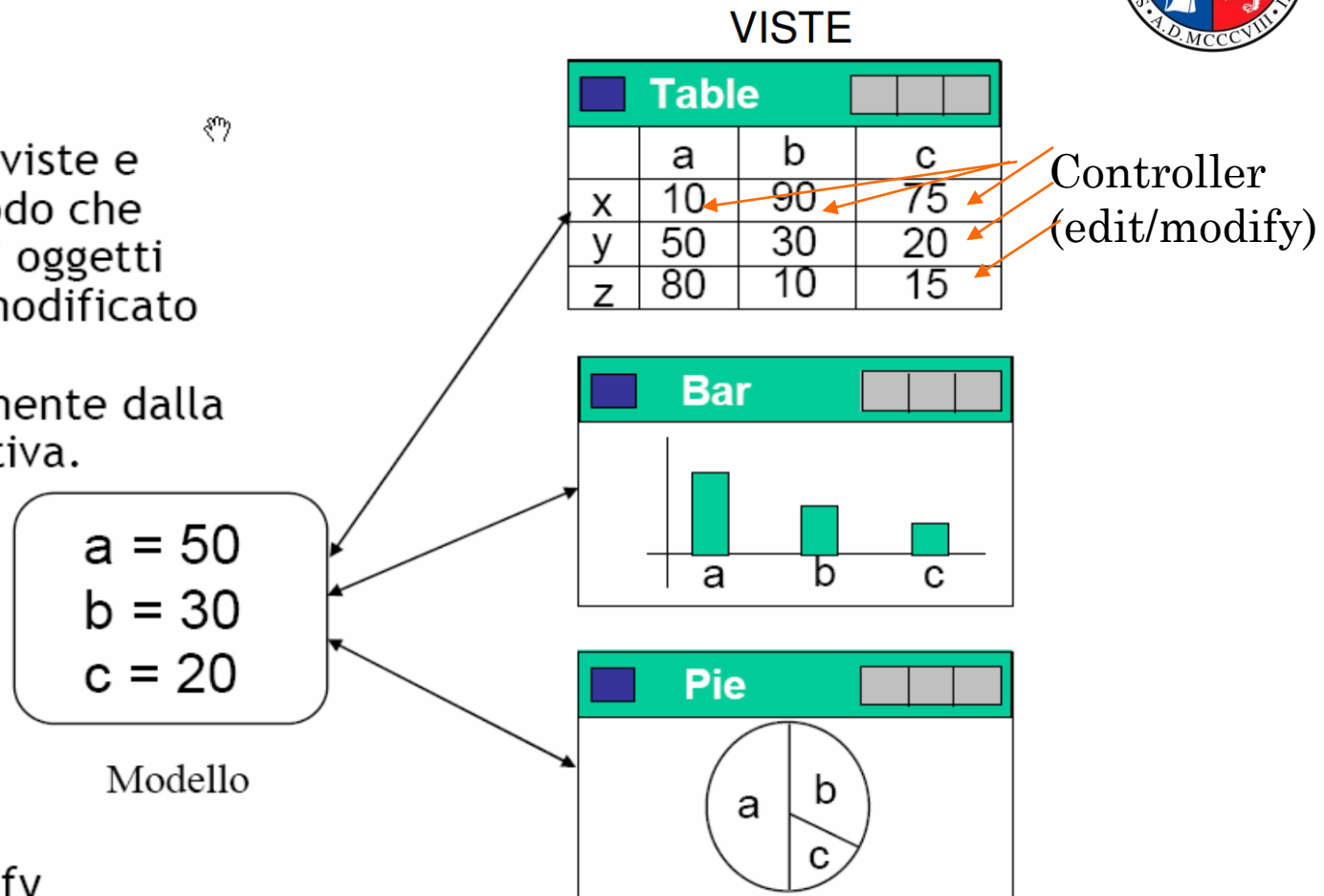


## □ Strategia:

- ▶ disaccoppiare viste e modelli, in modo che l'aspetto degli oggetti possa essere modificato o arricchito indipendentemente dalla logica applicativa.

## □ Come:

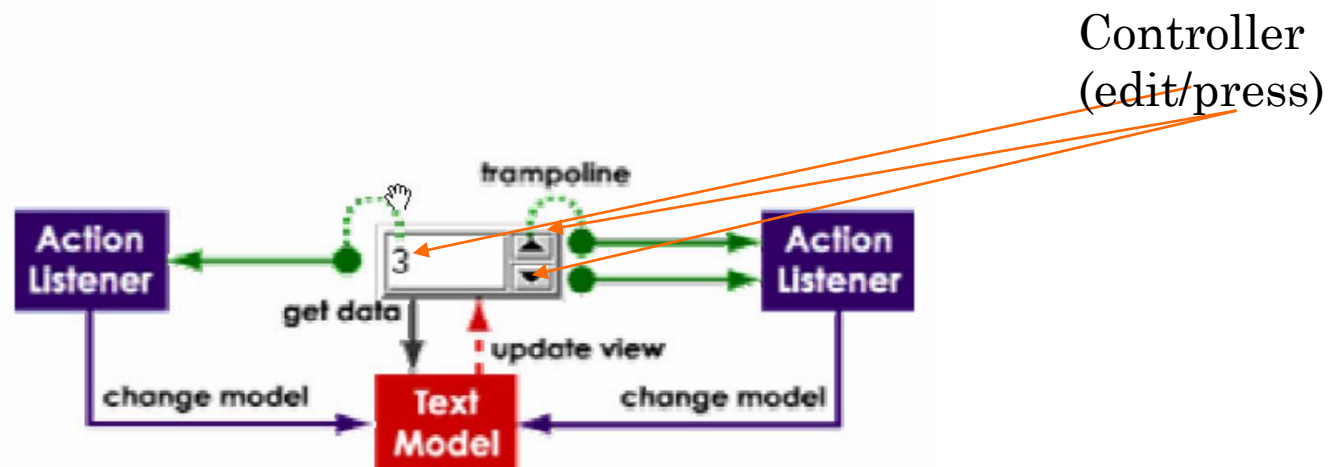
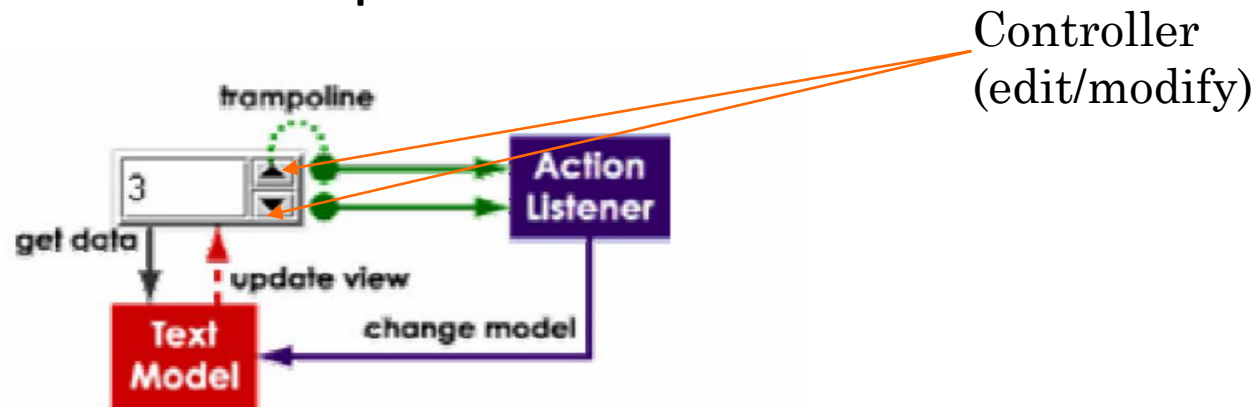
- ▶ meccanismo subscribe/notify



# DESIGN PATTERN: ESEMPIO MVC



## ○ Altro esempio di MVC: lo spin button

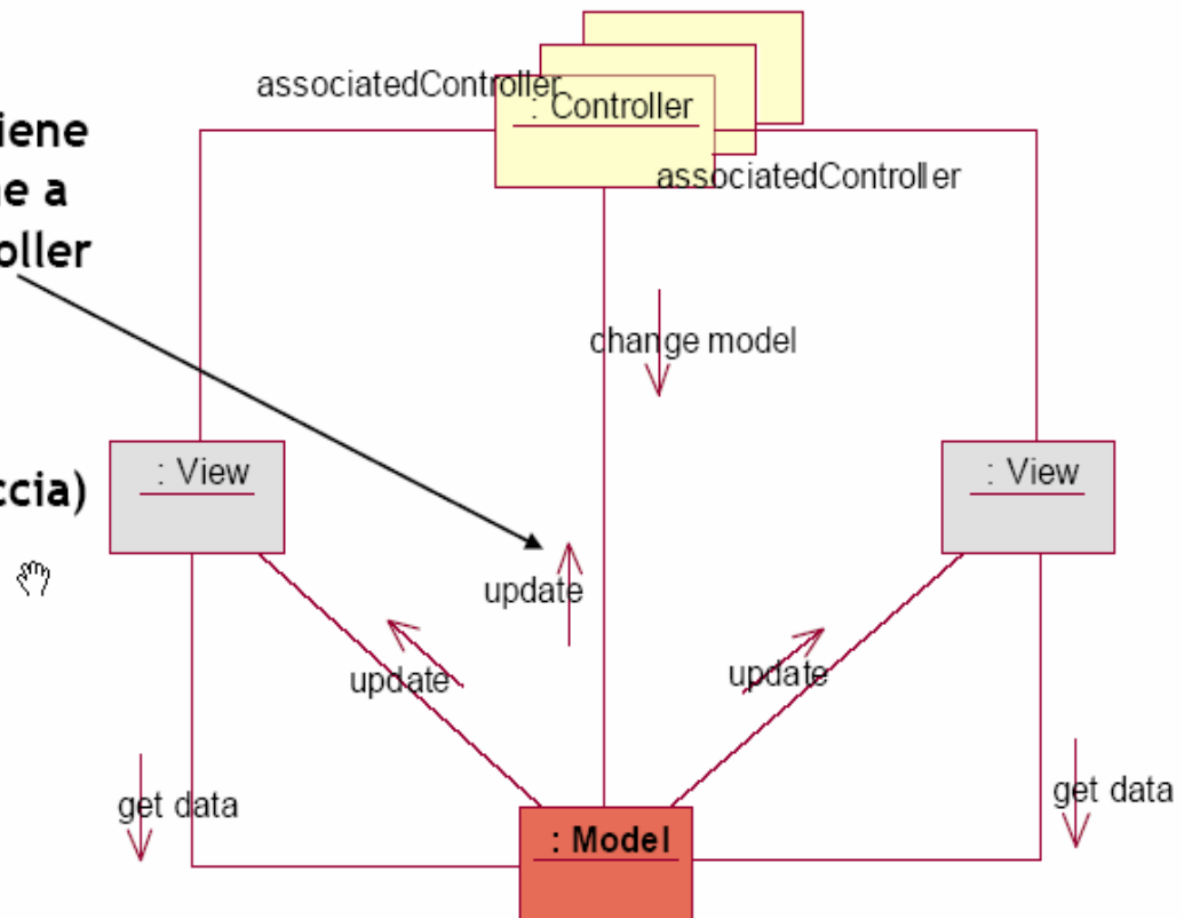


# DESIGN PATTERN: ESEMPIO MVC



## Lo schema UML

la notifica viene inviata anche a tutti i controller (Es. per disabilitare elementi dell'interfaccia)

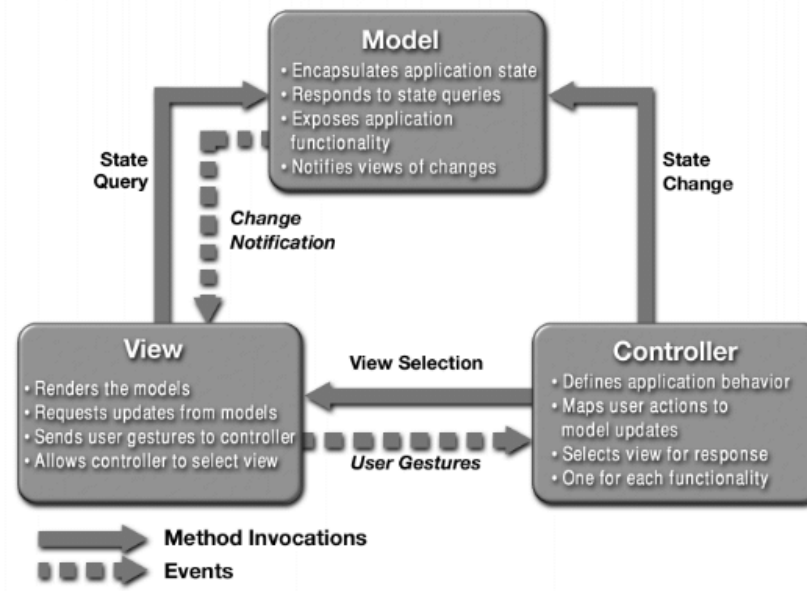


# DESIGN PATTERN: ESEMPIO



## ○ Model-View-Controller (MVC)

- **Disaccoppia** le tre componenti, rendendole **riusabili**
  - *Model*: dati di *business* e regole di accesso
  - *View*: rappresentazione grafica
  - *Controller*: reazione della UI agli *input* utente

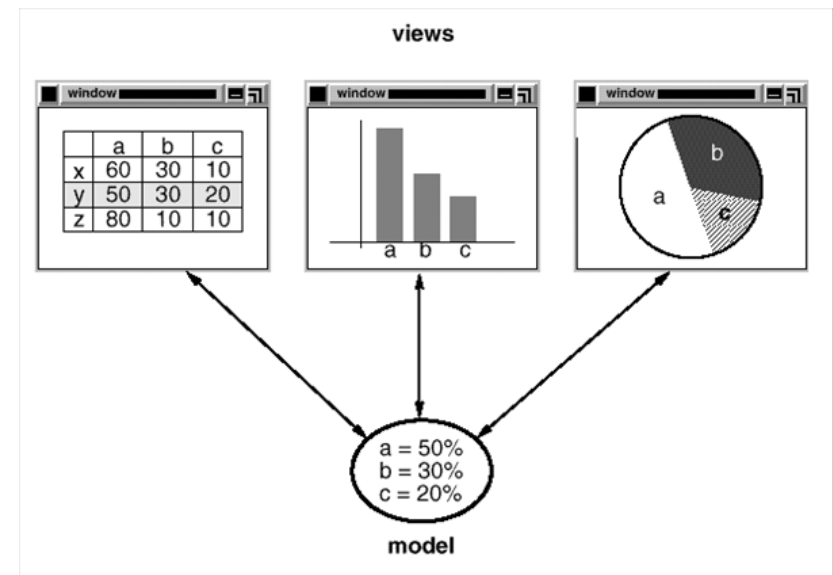


# DESIGN PATTERN: ESEMPIO



## ○ MVC: Model – View

- Disaccoppiamento → protocollo di notifica
  - View deve garantire una visualizzazione **consistente**
- Diversi *design pattern*
  - **Observer** design pattern
  - **Composite** design pattern





# DESIGN PATTERN: ESEMPIO



## ○ MVC: View – Controller

- Permette la modifica del comportamento in risposta all'*input*
  - Nessuna modifica di visualizzazione
- **Strategy/Algorithm** design pattern

## ○ Altri pattern...

- **Factory Method / Template** design pattern
- **Decorator** design pattern


# DESCRIZIONE DI UN DESIGN PATTERN



- Convenzioni per la specifica
  - **Nome** e classificazione
  - **Sinonimi**: altri nomi noti del pattern
  - **Motivazione**: problema progettuale
  - **Applicabilità**: contesti in cui il pattern può essere applicato
  - **Struttura**: rappresentazione grafica delle classi
  - **Partecipanti**: classi e/o oggetti partecipanti e responsabilità
  - **Collaborazioni** tra i partecipanti
  - **Conseguenze**: costi e benefici
  - **Implementazione**: suggerimenti, tecniche, errori comuni
  - **Esempio di codice sorgente**: possibile implementazione
  - **Utilizzo comune**: il pattern nei sistemi reali
  - **Pattern correlati**

# CLASSIFICAZIONE DESIGN PATTERN



Relazioni tra		Campo di applicazione		
		Creational (5)	Structural (7)	Behavioral (11)
	Class	Factory method	Adapter (Class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter(Object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

# SOMMARIO



- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- Utilizzare i design pattern

# RISOLVERE PROBLEMI ARCHITETTURALI



- Utilizzare gli oggetti **appropriati**
  - È **difficile decomporre** il sistema in oggetti!
    - Oggetti “reali” ≠ oggetti “non reali”
  - DP permettono di identificare le astrazioni meno ovvie
- Utilizzare la giusta **granularità**
  - Come **decidere** cosa deve essere un oggetto?
  - DP forniscono la granularità più appropriata
- Utilizzare le giuste **interfacce**
  - Gli oggetti sono “conosciuti” solo per il proprio insieme di operazioni
  - DP definiscono cosa mettere/non nell’interfaccia

# PRINCIPI DI PROGETTAZIONE



*Program to an interface, not an implementation.*

*Favor object composition over class inheritance*

# PRINCIPI DI PROGETTAZIONE



- Progettare pensando al cambiamento
  - Anticipazione di nuovi requisiti e loro estensioni
    - Massimizza il riuso
  - DP permettono al sistema di evolvere
    - Creazione di oggetti indirettamente (interfacce)
    - Decidere quale operazione utilizzare solo a *runtime*
    - Limitare le dipendenze dall'ambiente di esecuzione
    - Nascondere l'implementazione degli oggetti
    - Isolare gli algoritmi che hanno possibilità di essere estesi
    - Utilizzare astrazione e livellamento delle classi per ottenere un sistema con un basso grado di accoppiamento
    - Aggiungere nuove funzionalità utilizzando la composizione

# SOMMARIO



- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- **Livelli di progettazione**
- Utilizzare i design pattern

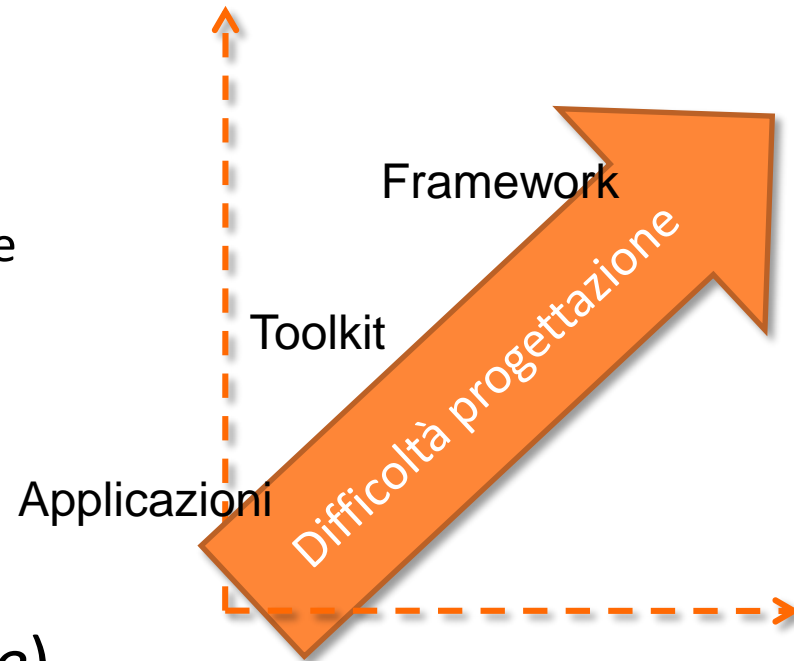


# LIVELLI DI PROGETTAZIONE



## ○ Progettare applicazioni

- Riutilizzo “interno”
  - Riduzione delle dipendenze



## ○ Toolkit (librerie *software*)

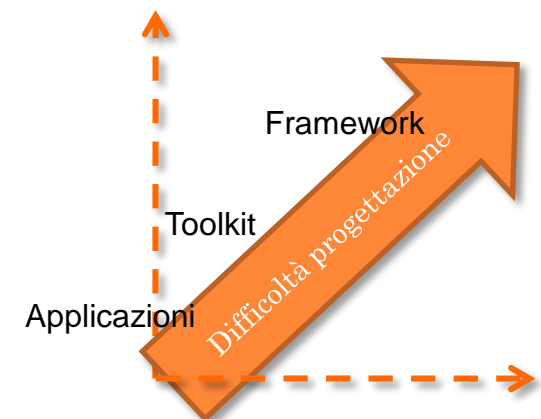
- Insieme di oggetti correlati e riutilizzabili progettati per fornire funzionalità generiche.
- Riutilizzo del codice (*code reuse*)
  - C++/Java I/O stream *library*

# LIVELLI DI PROGETTAZIONE



## ○ Framework

- Insieme di classi che cooperano per costruire **architetture riutilizzabili** per sviluppare un dominio di applicazioni
- Impone un **disegno architettuale**
  - Riutilizzo architeturale
- “*Inversion of control*”
- Minor grado di accoppiamento possibile
- NON sono *design pattern*
  - DP sono più astratti
  - DP disegnano architetture più piccole
  - DP sono meno specializzati

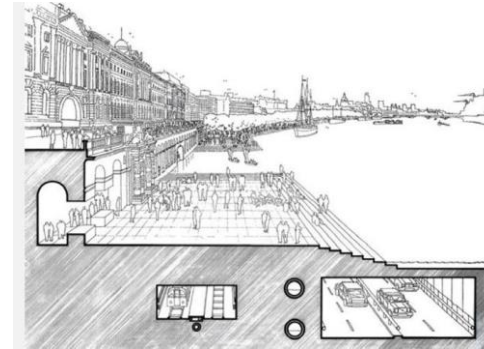


# TIPOLOGIE DI PATTERN



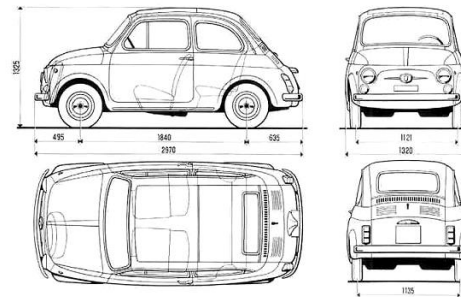
## ○ Architetturali

- Stili architetturali di alto livello



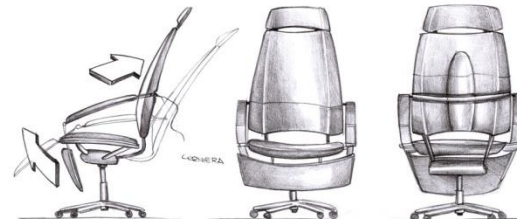
## ○ Progettuali

- Definiscono micro architetture



## ○ Idiomi

- Risolvono piccoli problemi
  - Legati al linguaggio



# DESIGN PATTERN ARCHITETTURALI



## ○ Pattern di alto livello

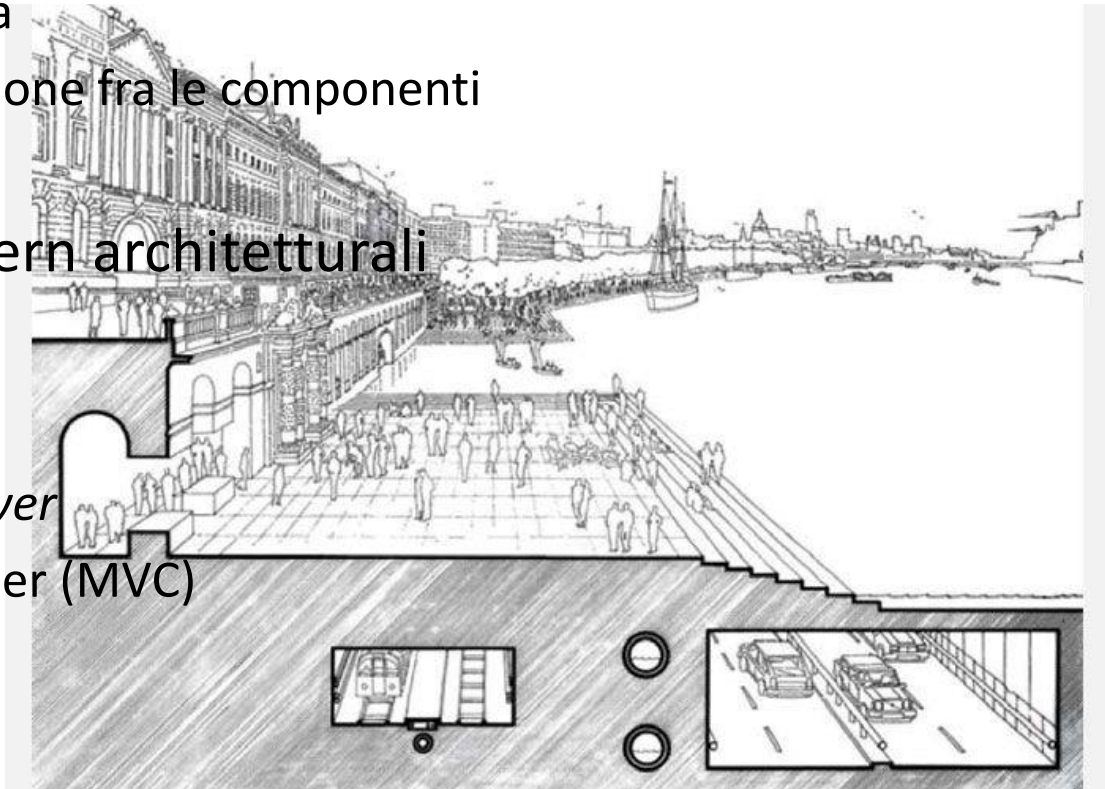
- Guida nella **scomposizione** in **sottosistemi**

- Ruoli e responsabilità
- Regole di comunicazione fra le componenti

- **Agglomerati** di pattern architetturali

- Esempi

- Paradigma *client-server*
- Model-View-Controller (MVC)
- *Peer to peer*

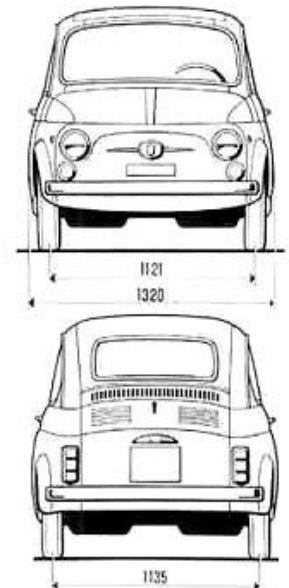
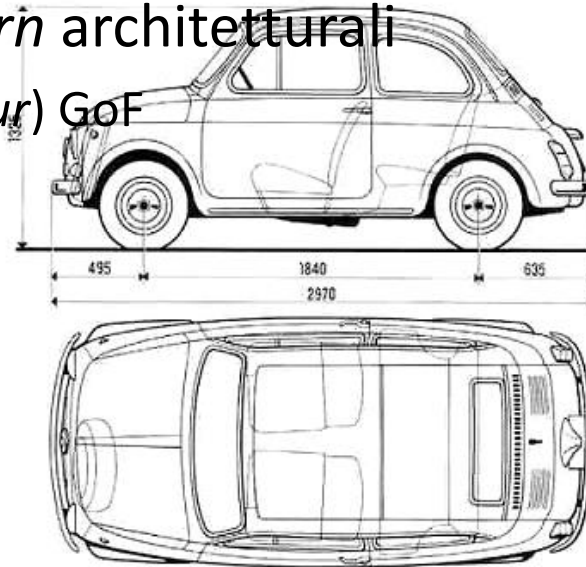


# DESIGN PATTERN PROGETTUALI



## ○ Progettazione di dettaglio di componenti

- Definiscono micro-architetture
  - Schema di comunicazione fra gli elementi di un sistema *software*
- Basi per costruire i *pattern* architetturali
  - Sono i *pattern* (*Gang of Four*) GoF
    - Factory
    - Command
    - Proxy
    - Observer
    - ...



# DESIGN PATTERN E IDIOMI

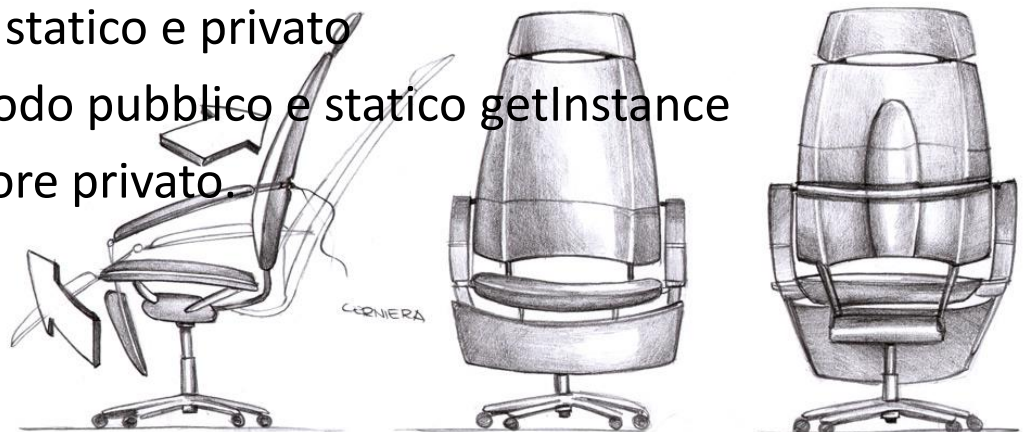


- Basso livello di astrazione

- Specifici del linguaggio di programmazione
  - Utilizzano direttamente direttive del linguaggio

- Esempio

- Come implementare un Singleton in Java?
  - Definire un attributo statico e privato
  - Implementare il metodo pubblico e statico getInstance
  - Impostare il costruttore privato.



# DESIGN PATTERN E IDIOMI



## ○ Design Pattern

- **Soluzione generica** ad una classe di problemi
- Propone un modello architetturale

## ○ Idioma

- **Soluzione specifica** ad un linguaggio
  - Legato alla tecnologia

Un design idiomatico rinuncia alla genericità della soluzione basata su un pattern a favore di una implementazione che poggia sulle **caratteristiche** (e le **potenzialità**) del **linguaggio**.

# SOMMARIO



- Introduzione
- Cos'è un design pattern
- Perché i design pattern
- Livelli di progettazione
- **Utilizzare i design pattern**



# UTILIZZARE I DESIGN PATTERN



## ○ Come **selezionare** i DP?

- Considerare come il DP risolve il **problema**
- Leggere la sezione delle **motivazioni**
- Conoscere le relazioni fra DP
- Considerare le possibili **cause di cambiamento** dell'architettura
- Considerare cosa può **evolvere** nell'architettura



# UTILIZZARE I DESIGN PATTERN



- Come **utilizzare** un *design pattern*
  - Leggere il DP **attentamente**
  - Rileggere le sezioni che descrivono **Struttura**, Partecipanti e Collaborazioni
  - **Analizzare** il **codice** di esempio fornito
  - Scegliere **nomi appropriati** per i partecipanti
  - Definire la struttura delle classi del *pattern* all'interno della propria applicazione
  - Scegliere nomi appropriati per le operazioni
  - Implementare le operazioni

# ANTI-PATTERN



Problemi ricorrenti che si incontrano durante lo sviluppo dei programmi e che dovrebbero essere evitati, non affrontati

- Evitare gli errori già commessi
- Evidenziare perché una soluzione sbagliata può sembrare in principio corretta per un problema
- Descrivere come passare dalla soluzione sbagliata a quella corretta

## ○ Esempi

- Reinventare la ruota (quadrata)
- Codice spaghetti
- Fede cieca
- Anomalia della sottoclasse vuota
- Programmazione “copia e incolla”
- ...



- Design Patterns, Elements of Reusable Object Oriented Software, GoF, 1995, Addison-Wesley
- MVC  
<http://www.claudiodesio.com/ooa&d/mvc.htm>
- Design Patterns  
[http://sourcemaking.com/design\\_patterns](http://sourcemaking.com/design_patterns)