# 10-Architecture and Design

Dopo la definizione dei Requisiti (cosa dovrebbe fare il software) avviene il design

**Architecture, design**: come il sistema dovrebbe essere costruito → **come progettare il sw per soddisfare i requisiti.**

* **Architettura**: componenti, framework di controllo e comunicazione (questo blocco sta con quello e fa questo)
* **Design**: interno di ogni componente (come i quadratini vengono definiti e implementati)

Molti difetti nelle implementazioni avvengono perchè:

* sw progettato male
* requisiti non erano chiari e quindi programmatore non sapeva cosa fare

Quando faccio il design, devo **specificaare il linguaggio che uso** → con una inspection, ci si accorge di eventuali problemi e si prendono contromisure prima di scrivere codice.

Dato un insieme di requisiti, posso fare diverse scelte:

* Scelte di alto livello
  + Diesel o gpl?
* Scelte di basso livello
  + Colore
  + Con ABS, ESP o no

*Passare dai requisiiti ai design è un processo creativo guidato da skill e da esperienza della persona → se ho già effettuato delle esperienze simili, sono guidato verso alcune scelte di design.*

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente**Guidelines**:

* Di stile architetturale
* Design pattern

In base a cosa devo processare, devo fare 2 cose diverse:

* **Processo solo sw** → requisiti sw, design sw (ramo dx)
* **System process** (RVC) → requisiti di sistema, design di sistema, requisiti sw, design sw (tutto)

**Esempio di allocazione hw-sw**

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamenteSe faccio una webApp → Server+browser   
Se faccio un’applicazione mobile → Applicazione su PC

### **Software design**

Dopo che si definisce il contesto hw, sw design riguarda:

* Definire moduli sw(functions, classes, packages, modules, ..) e le iterazioni che devono essere soddisfatte
  + Functional
  + non- functional requirements

## **Design del processo:**

* **Analisi**
  + Architettura
  + Design di alto livello
  + Design di basso livello
* **Formalizazione**
  + Text, diagram UML
  + Documentazione
* **Verifica**
  + Valutare problematiche

Ha come **input** → documento dei requisiti (funzionali e non)

Ha come **output** → documento di design, formato di componenti e collegamenti tra i vari componenti in grado di spiegare il tutto (strettamente collegato a qurllo che hai scritto nei requisiti)

1. **Architecture: intero sitema** → **definire** i componenti di alto livello e le loro iterazioni
   1. Dopo averli definiti, bisogna capire come questi componenti **comunicano**:
      1. Processi, thread
      2. Messaggi, chiamata e procedure, broadcast
   2. Usare **pattern** già usato e collaudati
2. **Design**:
   1. **Alto livello** → **classi** e loro iterazione
      1. Es: Utente che può essere manager o cliente → ora bisogna mettere per bene le variabili che andrò a scrivere nel mio sistema
   2. **Basso livello** → descrivere ad uno ad uno ciò che ho scritto nel class diagram → codice

### Proprietà di un design:

* **Funzionali**
  + **Immagine che contiene testo, Carattere, schermata, bianco

    Descrizione generata automaticamente**Il design che ho fatto, supporta i requisiti funzionali?
* **Non funzionali**: Il design che ho fatto, supporta i requisiti non funzionali?
  + Generali:
    - Affidabilità, efficienza, performance, usabilità, manutenibilità, portabilità, safety, security
  + Specifiche di design:
    - Testabillità (Osservabilità + controllabilità)
    - Monitorabilità
    - Interoperabilità
    - Scalabilità
    - Distribuibilità
    - Mobilità
    - Complessità:
      * Numero di compoennti
      * Numero di iterazioni

# KISS

* Immagine che contiene testo, schermata, linea, Carattere

  Descrizione generata automaticamenteCoupling:
  + Grado di dipendenza tra due componenti (esistono tool che mostrano i vari collegamenti tra componenti, dipendenze tra classi)
    - Classe che dipende dall’altra, se cambio una classe o la sposto, quale altra viene impattata (es: tubi e muri)
* Coesione:
  + Immagine che contiene testo, schermata, Carattere, linea

    Descrizione generata automaticamenteGrado di consistenza delle funzioni di un componente
  + Motore non è strettamente correlato a rilevare ostacoli → tenere separate le cose
* Costi
* Schedule
* Staff

**Proprietà** → Cosa fare:

* **Performance**
  + Localizare operazioni critiche che minimizzano la comunicazione
    - Es: la parte di analisi immagini la faccio con una libreria c++

Il resto dell’app in kotlin

* **Sicurezza**
  + Fare in modo che le parti critiche siano nascoste
* **Safety**
  + Dove è necessaria (es: automotive → usare un sottoinsieme del linguaggio C per le cose che non possono fallire)
    - * Se devo comunicare in maniera siocura con un componente devo fare rindondanza e aumentare disponibilità ma perdo performance
* **Availability**
  + Uso componenti rindondanti per aunentare tolleranza
    - es: inserisco due schede di rete di cui ne uso solo una
      * se inserisco dati rindondanti, devono mettere in sicurezza più dati → maggior effort in security
* **Maintenability**
  + Cerco componenti che si possano modificare
    - Componente che dialoga con il bluetooth non funziona → cerco uno analogo che possa essere funzionante nella mia architettura
    - Se posso uso componenti standard
      * Se personalizzo troppo, faccio una soluzione molto performante ma riduco mantenibilita

TRADE OFF

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

### **FORMALIZZARE ACHITETTURA**: da slide 35 a 51

* Informale
  + Immagine che contiene testo, schermata, diagramma, linea

    Descrizione generata automaticamenteBox e linee
    - Descrizione, separando funzionalità e responsabilità
    - Utili per comunicazioni con stakeholders
* **Semiformale**
  + **UML diagrams**
    - Structural: aiuta a rappresentare la struttura in modo organizzato e gerarchico attraverso:
      * Component, package diagrams
      * Class diagrams
      * Deployment diagrams
    - Dynamic views
      * Sequence diagrams
      * State charts
* **Formal ADL** → per sistemi complessi con alta safety e etc.

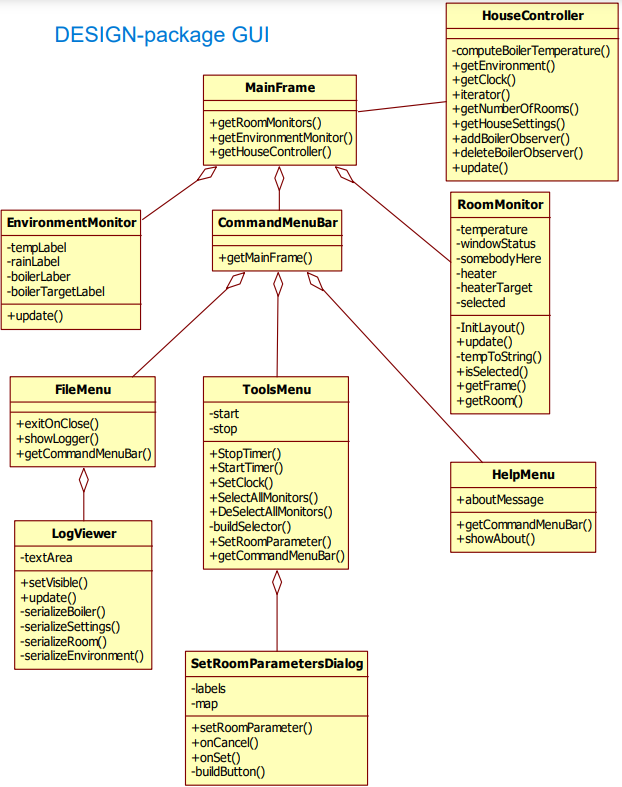
Esempio**:**

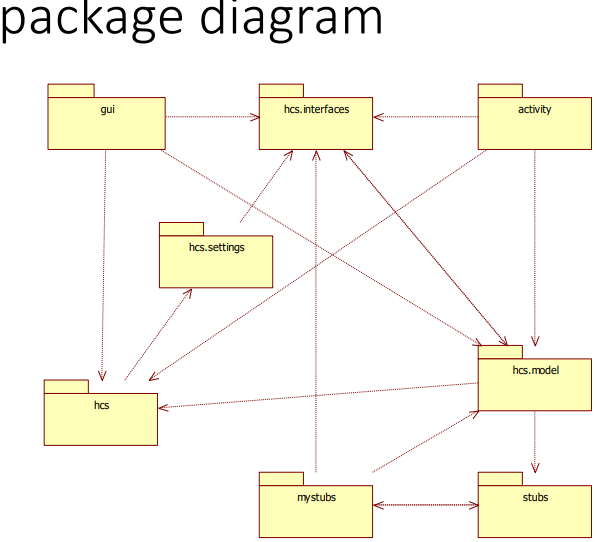
**Heating control system:**

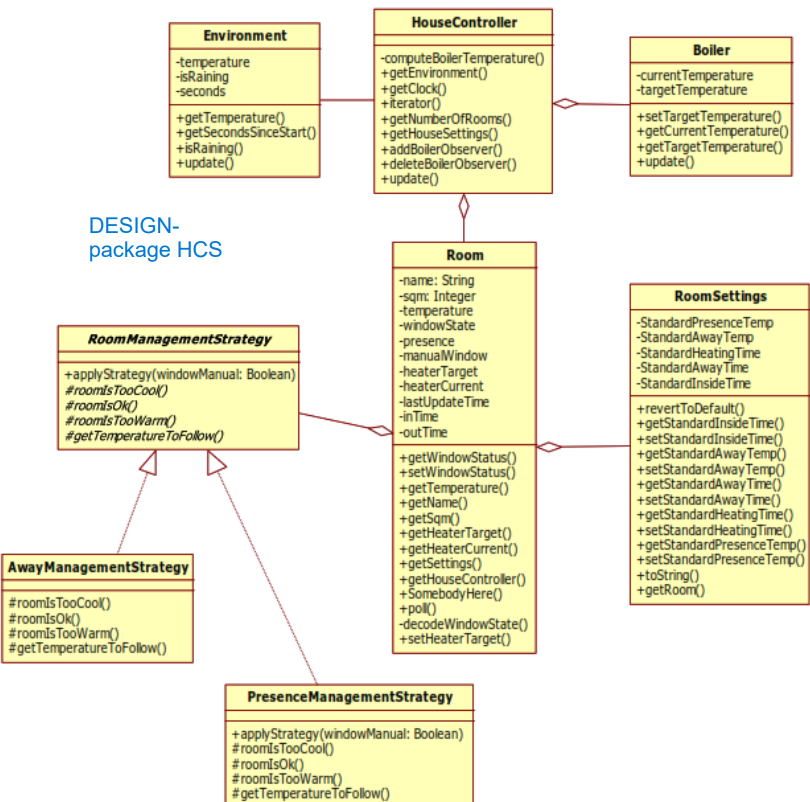
**obbiettivo: controllare la temperatura in una casa, usando sensori in ogni camera e attuatori**

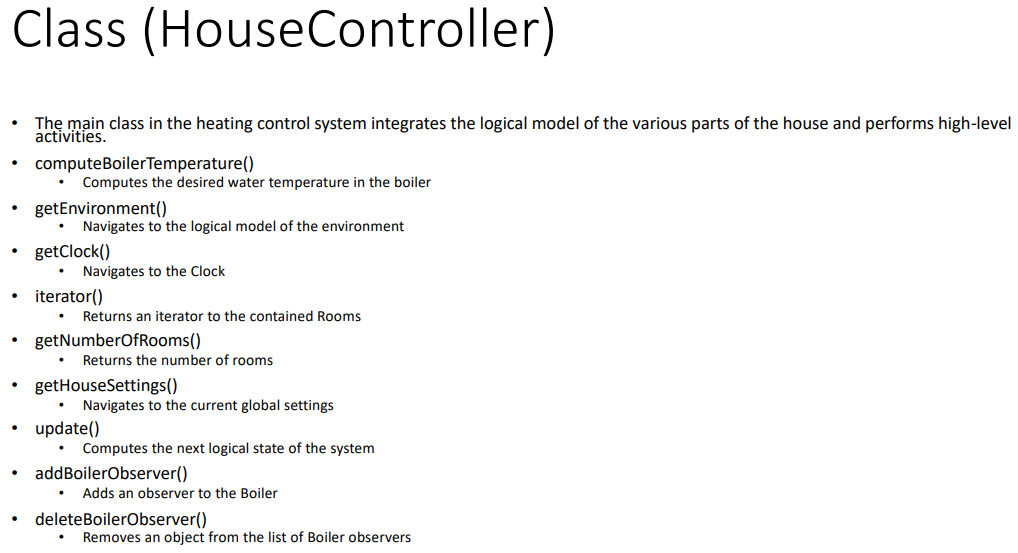
* Opt1 → alto livello
  + Una cpu, un processo
  + Comunicazione e controllo con chiamate a procedure
  + Stile a livelli
* Opt2 → alto livello
  + Metto una cpu per ogni casa e faccio chiamate http per comunicazione e controllo
  + concorrenza

*Man mano che si scende, molti più dettagli*



UML-struttural





Parte più **dinamica**, prende in considerazione anche il tempo:

**State chart:**

Immagine che contiene testo, diagramma, linea, Parallelo

Descrizione generata automaticamente

Immagine che contiene testo, linea, diagramma, schermata

Descrizione generata automaticamente

## PATTERN:

Soluzioni riutilizzabili per problemi ricorrenti in un contesto ben definito.

Tipi di pattern:

1. Immagine che contiene testo, schermata, Carattere, numero

   Descrizione generata automaticamente**Architetturali** →Potrebbe interessarmi il tipo di linguaggio (ad oggetti/no)F

Danno un funzionamento generale del sistema, applicabili in domini specifici, bisogna conoscerli tutti e riconoscere quale serve

* + **Layers**: organizzare il sistema in diversi livelli, ognuno dei quali fornisce un servizio differente
    - Vantaggi: ogni layer risolve un particolare problema
      * Immagine che contiene testo, Carattere, schermata, numero

        Descrizione generata automaticamenteQuando cambio un componente, impatto solo il layer vicino, cambio comunicazione solo con lui
  + Immagine che contiene testo, linea, diagramma, Carattere

    Descrizione generata automaticamente**Pipes and filter**: per fare un insieme di operazioni sequenziali, passando i risultati
    - Data source passato in ingeresso
    - Filtri passati in ingresso
    - Immagine che contiene testo, Carattere, diagramma, schermata

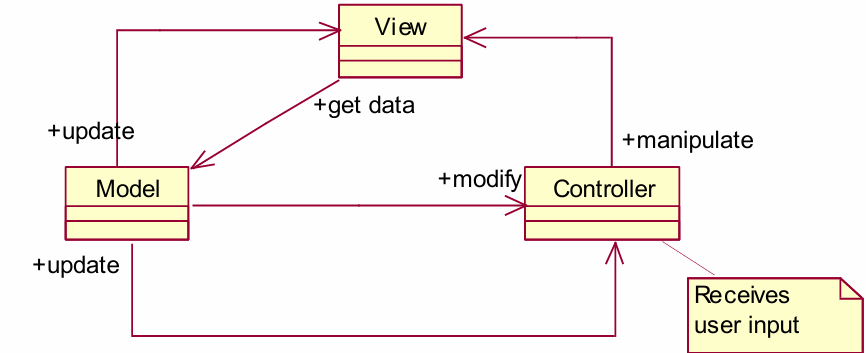
      Descrizione generata automaticamenteDatasynk che consuma il dato → risultato finale
  + **Repository**: se ho un’architettura complessa posso:
    - Immagine che contiene testo, schermata, Carattere, diagramma

      Descrizione generata automaticamentedb centrale → dati condivisi sono mantenuti in un db centrale o repository. Possono essere acceduti da tutti I sottosistemi.
    - Usare proprio db per ogni sottosistema/modulo e usare il repository per dialogare cong li altri (mando dati al repository e tuti i componenti leggono da lì)
      * Utile se ho bisogno di mandare tanti dati in diversi posti
      * Immagine che contiene testo, schermata, Carattere, algebra

        Descrizione generata automaticamenteNon uso API dirette verso i componenti, mando solo a repository
  + **Immagine che contiene testo, schermata, Carattere, diagramma

    Descrizione generata automaticamenteClient server**:
    - Server in giro
    - Immagine che contiene testo, schermata, Carattere, algebra

      Descrizione generata automaticamenteClient accedono
  + Immagine che contiene testo, linea, diagramma, schermata

    Descrizione generata automaticamente**Broker**: sistema complesso con componenti eteorogenei che deovno essere in grado di comunicare tra loro
    - Il broker nasconde tutto esponendo dei metodi e poi gestisce lui
      * Gestisce tutto
    - Componenti IOT
  + **MVC**: mostra i dati all’utente e gestisce il cambiamento tra dati
    - Se ho la stess ainformazione conservata in stesso modo e finiestre devono gestire stati, deov avere:
      * Model: responsabile di geestire lo stato → si interfaccia con DB o filesystem)
      * View: mi mostragraficamente quello xche ho a sistema, renderizza UI
      * Immagine che contiene testo, Carattere, schermata, bianco

        Descrizione generata automaticamenteController: responsabile per la gestione degli eventi da UI

Immagine che contiene testo, schermata, memoria flash, Carattere

Descrizione generata automaticamente

* + **Microkernel**: componenti he lavorano tra loro in maniera continua, architettura portabile
    - Coresystem → fa il grosso lavoro
    - Plug-in inseribili per ulteriori funzionalità
  + **Microservices**: fare tanti pezzetti indipendenti tra di loro ed eseguo comunicazione tramite json → ogni componente è indipendente
    - Ogni componente ha le proprie API
    - Tutte in http rest
      * Posso avere metà python e metà C, l’importante è che faccio comunicazione usando correttamente le API

Nota: un sistema reale è influenzato da diversi pattern architetturali, non uno solo.

1. **Design pattern** → Mi interessa lo specifico linguaggio

* Definire le classi, recuperandole dal glossario e dal context diagram
  + Attributi, tipo, privacy
  + Metodi, tipo di ritorno, numero e ripo di parametri, privacy
  + Getter e setter
  + Algoritmi per implementare i vari requisiti
  + Pe rogni relazione → scegliere implementazione
    - Chiave
    - Array, mappa, lista
    - Domanda di michele: 10:21 →52:00
* Dentro classi
* Decidere persistenza
  + No persistenza
  + Sì persistenza
    - Serializzazione (a file o rete)
    - A database
      * Framework
    - Su tutti gli oggetti
    - Su parte degli oggetti

**Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Descrizione generata automaticamente**Esempio:

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamenteImmagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente

## **Design pattern:**

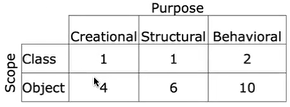
Descrizione di oggetti e classi che devono comunicare per risolvere un problema in un contesto preciso:

* Come creare queste classi
* Identificare aspetti principali e più comuni con l’idea di creare codice riutilizzabile ad oggetti

#### **Ogni pattern è caratterizzato da:**

* **Nome e classificazione**
* **Intento**: cosa fa il pattern, a cosa serve e che problema risolve
* **Motivazione**: scenario che spiega il problema e come possono classi e oggetti risolvere questo problema
* **Applicabilità**: descrive le situazioni in cui questo design pattern può essere usato
  + Esempi di design non ottimale, come riconoscerlo
* **Struttura**: una rappresentazione grafica delle classi, di come si usano
* **Partecipanti**: le classi e/o gli ofetti che partecipano ai design pattern e le loro responsabilità
* **Collaborazioni**: come i partecipanti comunicazion per risolvere i problemi
* **Conseguenze**: come il design pattern supporta gli obbiettivi, quali sono i trade-off da tenere in considerazioni, quali aspetti il sistema permette di variare
* **Implementazione:**
* **Codice di esempio**
* **Casi d’uso noti**
* Pattern **collegati** :

#### **Come selezionare un pattern:**

* Considerare come il pattern risolve i problemi
* Analizzare gli intenti
* Studiare come i pattern sono correlati
* Esaminare cause di ridesign
* Considerare come effettuare redesign se parto da altro

## Esistono 3 tipi principali di pattern:

## **Creational Patterns:**

#### Factory Method

#### **Abstact Factory**

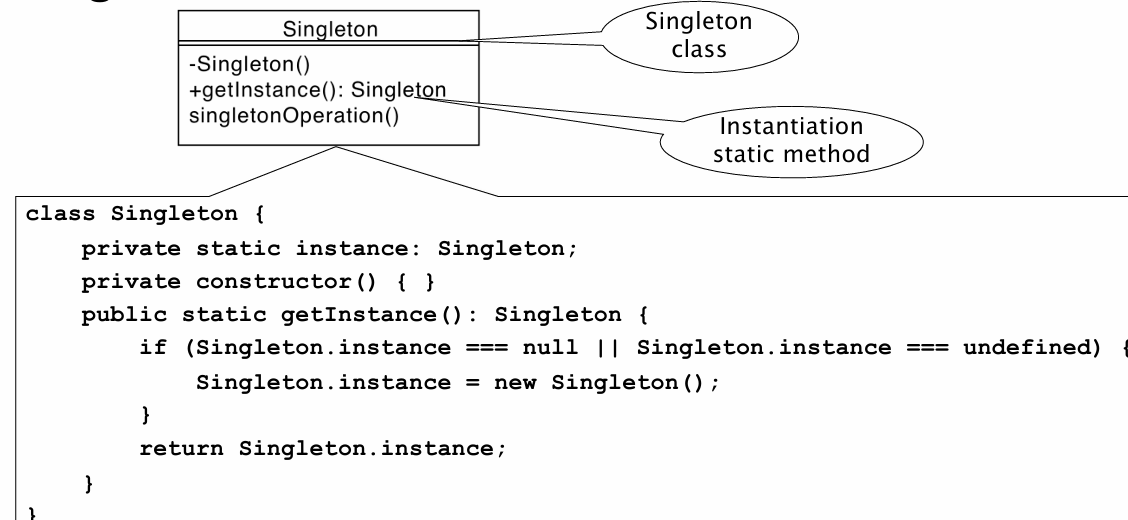
Ho una famiglia di classi simili tra di loro ma con dettagli di implementazione diversi.

* Posso dare una classe astratta che descrive l’operazione generale e poi questa viene implementata in diverse forme
  + Metodi che ogni classe definisce nella classe specifica, sono quelli riportati nella classe astratta

#### Builder

#### Prototype

#### **Singleton**

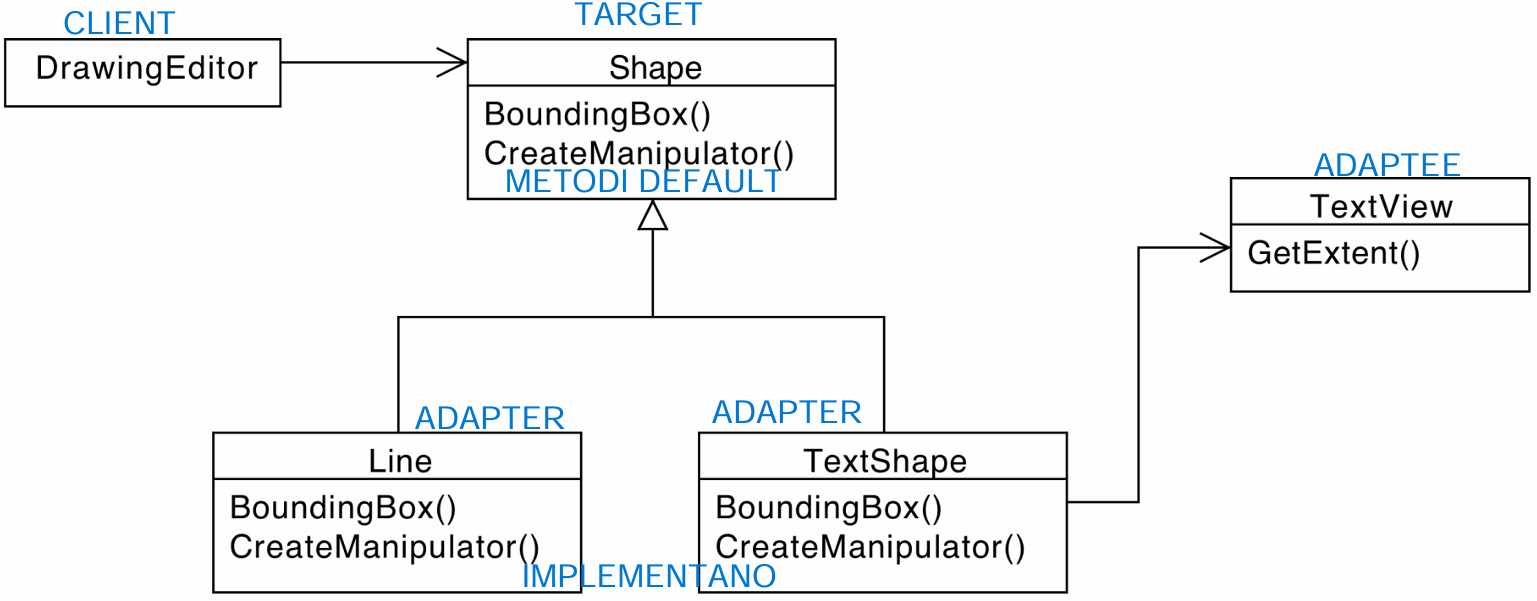
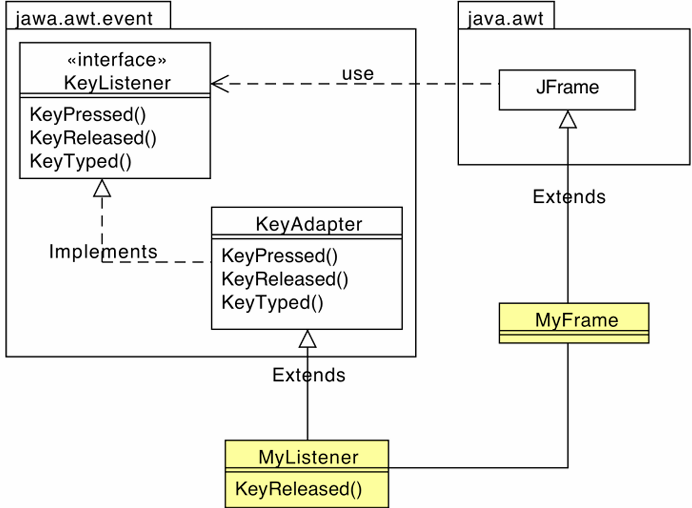
Classe particolare che può avere una sola istanza

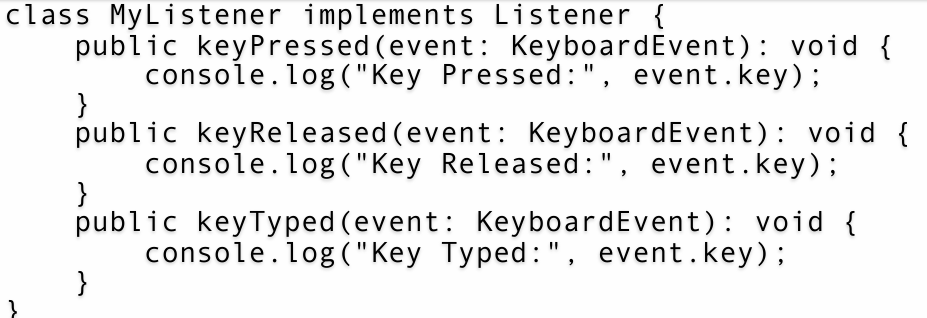
* Anche se faccio tante new della stessa classe → ritorno puntatore alla singola istanza
  + Es: variabile statica che si salva il nr di oggetti implementati di un determinato tipo
    - Uso una singleton che chiamo ogni volta che creo un oggetto

## **Structural Patterns:**

Si occupano di definire come classi e oggetti sono composti in strutture più grandi

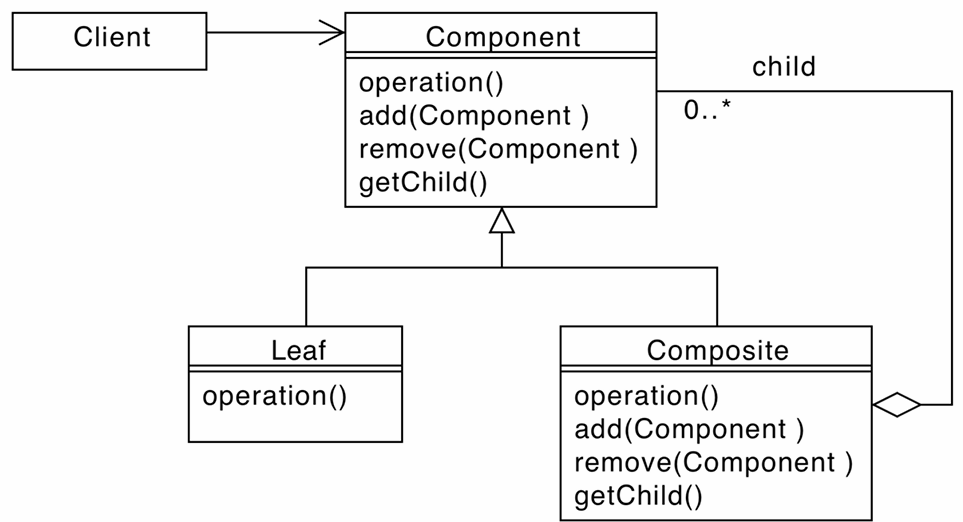
#### **Adapter**:

****Classe che fornisce determinate features da implementare. Ereditarietà fondamentale.  
C’è bisogno di un’interfaccia listener

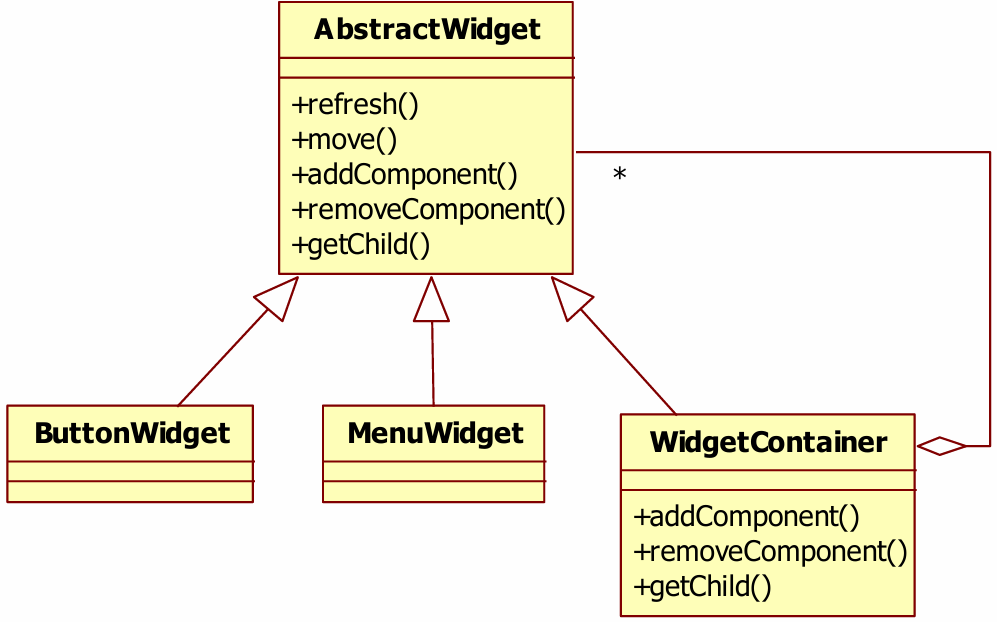


#### Bridge

#### **Composite:**

Rappresentare delle gerarchie di oggetti

In ogni componente ho:

* **Leaf** → contiene operation
* **Composite** → possibilità di aggiungere componenti figli

-es su slide per le operazioni, SLIDE 157

#### **Façade**

Immagine che contiene testo, ricevuta, Carattere, schermata

Descrizione generata automaticamenteImmagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamenteSalva gli oggeti ed espone i metodi. Mi permette di non dover dihiarare tutte le volte gli oggetti ma Facade.

#### Flyweight

#### Proxy

## **Behavioral Patterns:**

Lavorano con gli algoritmi → che fare quando chiamo determinate funzioni.

### OBJECT LEVEL

#### Chain of Responsibility

#### Command

#### Iterator

#### Mediator

#### Memento

#### **Observer**

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamenteSe il cambio di un oggetto Observable influenza altri oggetti Observer (che possono essere non noti in partenza)   
Un insieme di elementi implementano l’interfaccia Observable e si iscrivono, tutti questi oggetti vengono trattati come oggetti che implementano Observable → se succede qualcosa update (lista di observer).

* **Immagine che contiene testo, Carattere, schermata, bianco

  Descrizione generata automaticamente**Sono asincroni → possono accadere in un qualasiasi momento
* **addObserver →** subscribe agli aggiornamenti di un tipo Observable

**Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente**

*Immagine che contiene testo, Carattere, ricevuta, schermata

Descrizione generata automaticamenteSe lo stato è cambiato, faccio un for su tutti gli observer e gli mando l’aggiornamento*

#### State

#### **Strategy**

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteMolte classi/algoritmi hanno un core specifico e comportamenti diversi:

* classe astratta che ha una implementazione generale
  + diverse implementazioni della stessa cosa

#### Visitor

### CLASS LEVEL

#### Template Method

#### Interpreter

**https://**[**github**](https://github.com/torokmark/design_patterns_in_typescript)**.com/torokmark/design\_patterns\_in\_typescript**

## Esempi

#### Espressione tra costante e variabile

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente

#### Statistiche questionari indipendentemente dal file di provenienza

Immagine che contiene testo, diagramma, linea, schermata

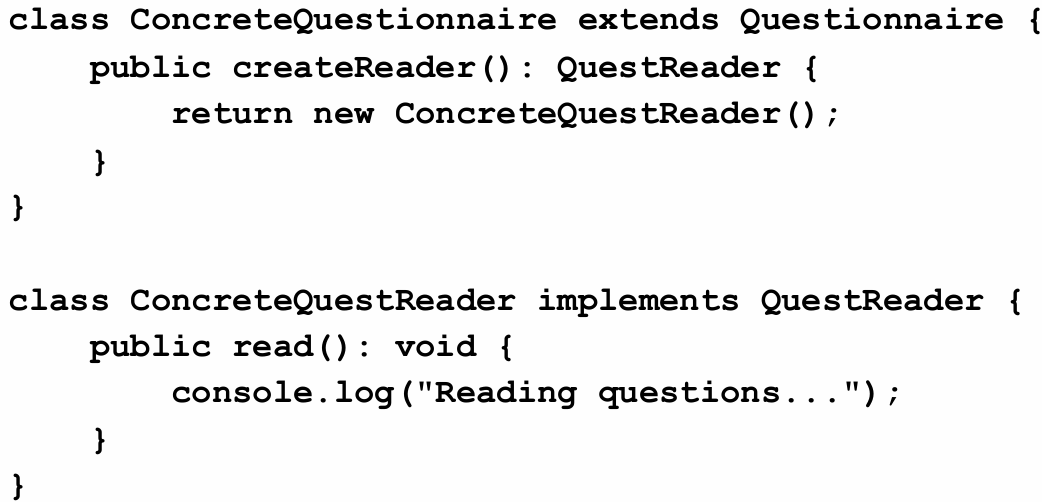
Descrizione generata automaticamenteImmagine che contiene testo, Biglietto Post-it, schermata, diagramma

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente



## Verification

**Immagine che contiene testo, schermata, Parallelo, linea

Descrizione generata automaticamenteFunctional requirements**

* **Traceability matrix** 
  + Ogni requisito funzionale che arriva dai requisiti deve essere supportato da almeno una classe nel sw design
* **Scenarios executed on architecture** 
  + Immagine che contiene diagramma, testo, linea, Piano

    Descrizione generata automaticamenteOgni scenario deve avere un insieme di chiamatye a metodi che devono essere in grado di farlo funzionare
* Inspection

**Non functional requirements**

* Performance
* Scenarios enriched with time model
* (Inspection)

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamenteffvc

Immagine che contiene testo, schermata, Carattere, bianco

Descrizione generata automaticamente