

Parte 1

Introduzione al corso di formazione Full-Stack

Struttura

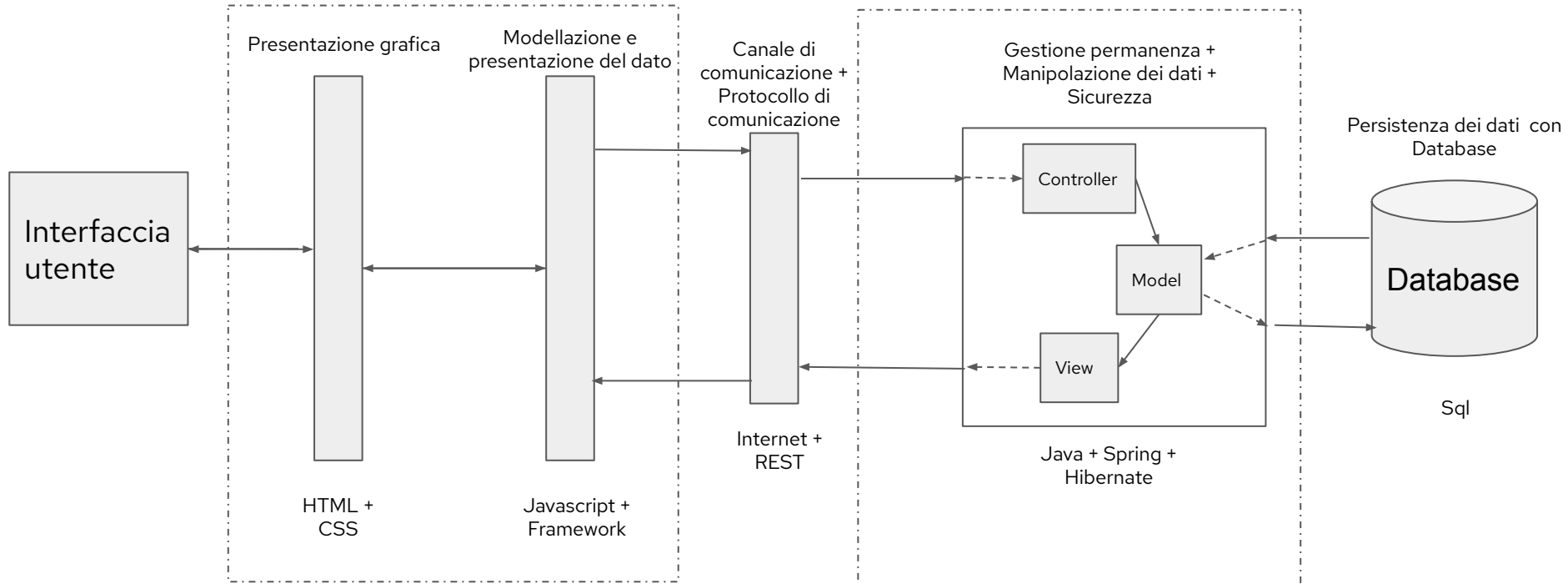


Frontend

Backend

Presentation layer

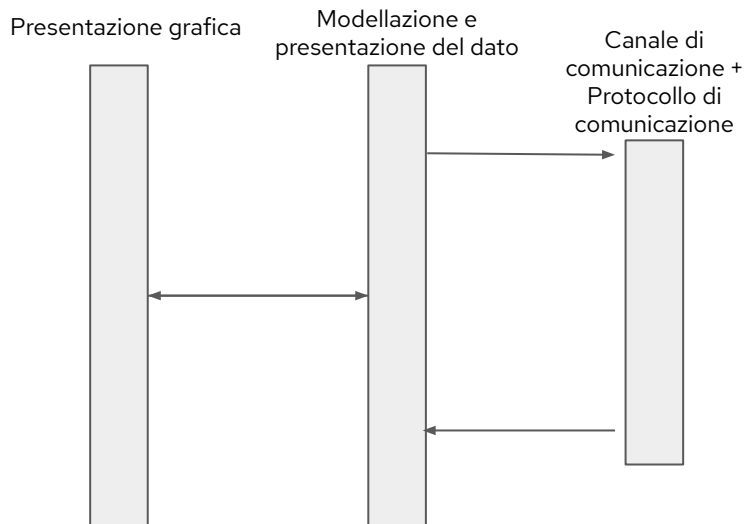
Data Access Layer



Conoscenze

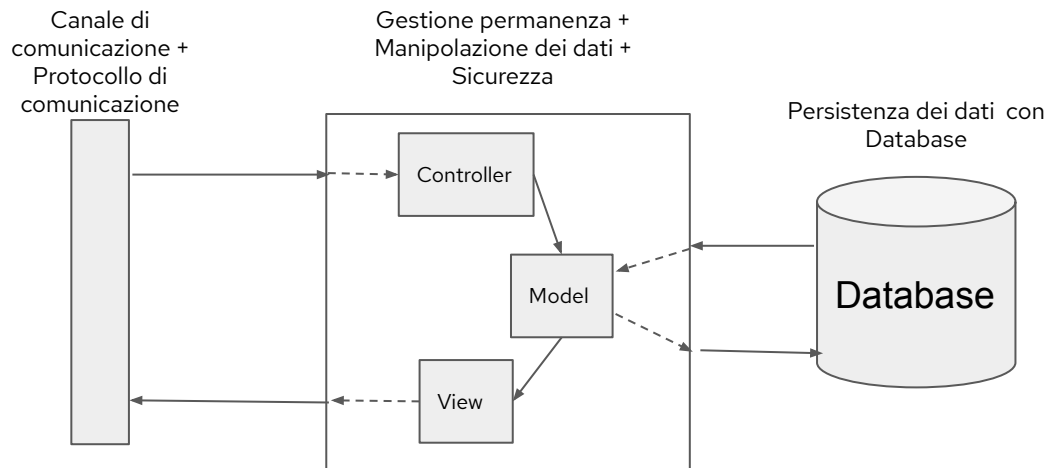
Front-end

- HTML
- CSS
- JAVASCRIPT
- TYPESCRIPT
- FRAMEWORK
- REST



Back-end

- Java
- Spring
- Database
- Sql
- Design MVC
- Rest



In comune

- REST
- Version Control System
- OOP
- Concetti base programmazione
- Design pattern
- Altro...

Parte 2

I fondamenti della programmazione

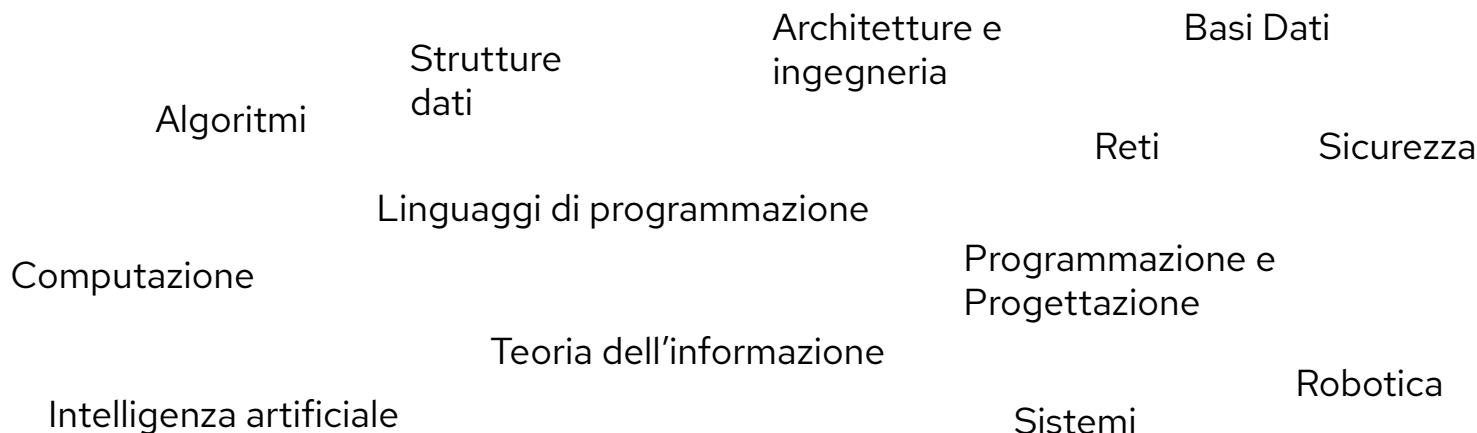
Concetti base



Informatica



Scienza che si occupa del trattamento dell'informazione mediante procedure automatizzate. Studia l'informazione secondo gli aspetti teorici, aspetti computazionali, logici e pratici, tecniche dell'implementazione della sua automazione...



Programmazione



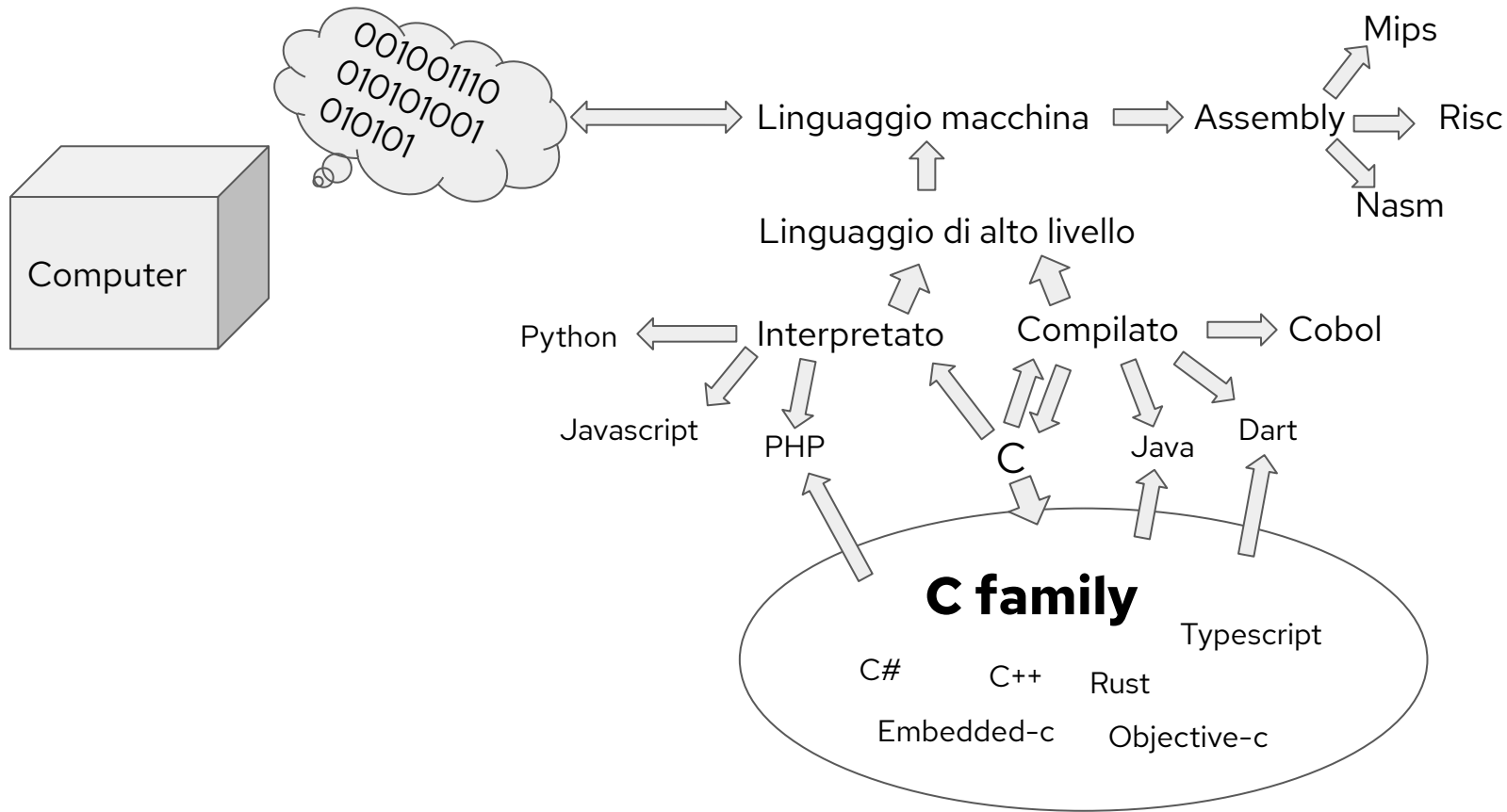
È il processo di progettazione e costruzione di un programma eseguibile da un calcolatore per realizzare o eseguire un compito specifico.

Per programmare abbiamo bisogno di un linguaggio che ci permetta di comunicare con la macchina

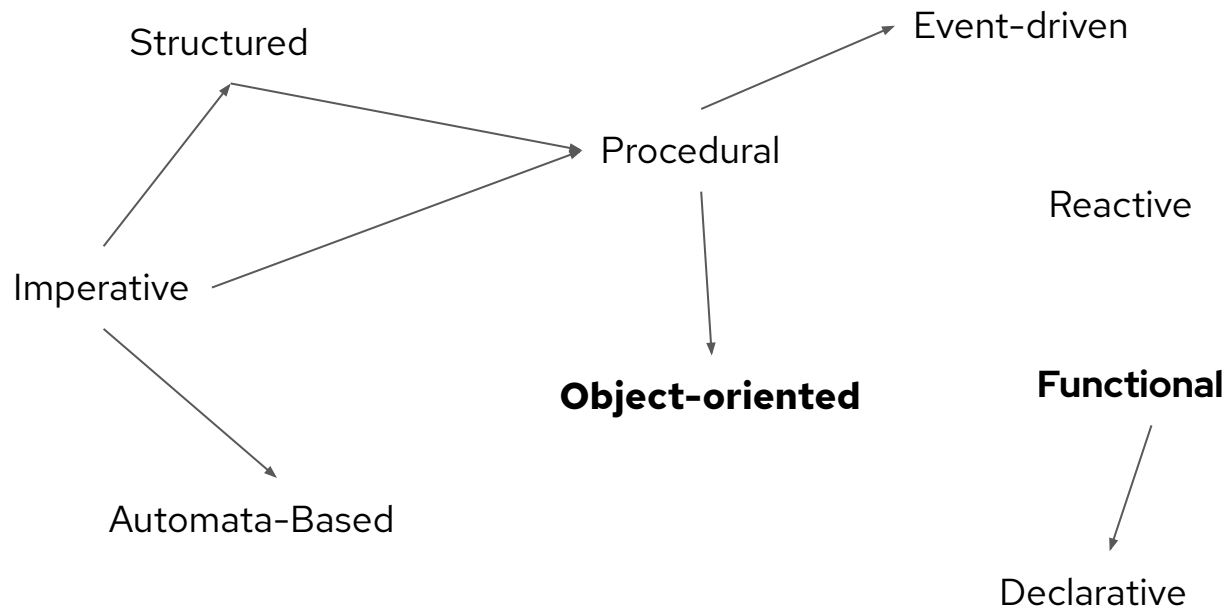


Linguaggi di programmazione

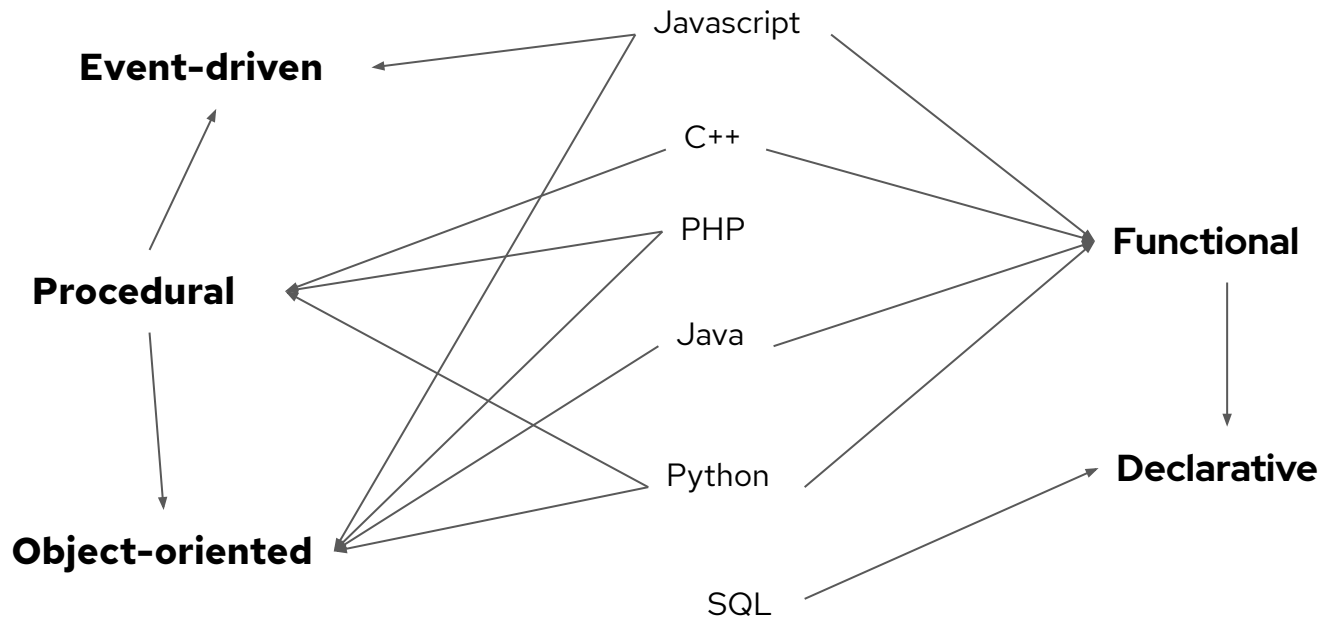




Paradigmi della programmazione



Paradigmi della programmazione



OOP



Object-Oriented Programming (OOP)

Programmazione orientata agli oggetti è un paradigma che si basa sul concetto degli **oggetti**.

Gli oggetti sono contenitori di: dati sotto forma di **campi** (attributi o proprietà) e di codice sotto forma di **procedure** (metodi).

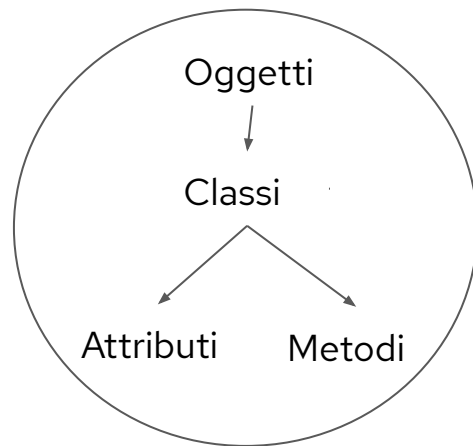
Linguaggi di questo tipo sono basati sulle **classi** che definiscono il **tipo** degli oggetti.

Proprietà che determinano OOP

Incapsulamento (black-box) = I dati che definiscono lo stato di un oggetto e i metodi che ne definiscono la sua logica sono accessibili solo ai metodi interni, ma non agli utilizzatori esterni.

Ereditarietà = meccanismo che permette di derivare da nuove classi a partire da quelle già definite realizzando una gerarchia di classi.

Polimorfismo = Lo stesso codice può essere utilizzato con istanze di classi diverse, aventi una superclasse comune. (Attenzione a non confonderla con l'ereditarietà)



Design pattern




Modelli - Design pattern

Si tratta di metodi di risoluzione a problemi di ingegneria del software comuni (ideati inizialmente dalla GoF).

Trattano risoluzioni efficaci per la progettazione del software != algoritmi risolutivi a tutti i problemi.

Sono classificati in 3 categorie principali:

- Creazionali - Creational
- Strutturali - Structural
- Comportamentali - Behavioral

- 
- nome
 - problema che risolvono
 - soluzione al problema
 - conseguenze delle soluzione

Noi analizzeremo quelli che mostrano le interazioni tra oggetti.

Creazionali

Problema = fornire la capacità di creare oggetti sulla base di un criterio richiesto e in modo controllato.

Soluzioni:

- **Abstract factory**
- Builder
- **Dependency injection**
- **Factory method**
- Lazy initialization
- Object pool
- Prototype
- Resource Acquisition Initialization
- **Singleton**
- Multiton

Strutturali

Problema = organizzare classi e oggetti diversi per formare strutture più grandi e fornire nuove funzionalità.

Soluzioni:

- **Adapter**
- **Bridge**
- **Composite**
- **Decorator**
- Extension object
- Proxy
- Facade
- Flyweight
- Front Controller
- Marker
- Module
- **Twin**

Comportamentali

Problema = identificazione di modelli di comunicazione comuni tra gli oggetti e la loro realizzazione.

Soluzioni:

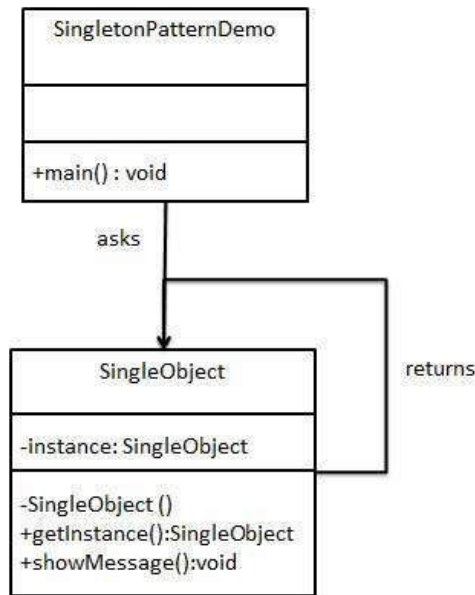
- **Strategy**
- Blackboard
- Chain of Responsibility
- **Command**
- Interpreter
- **Iterator**
- **Mediator**
- State
- **Template method**
- **Visitor**
- **Memento**
- Null Object
- **Observer (publish/subscribe)**
- Servant
- Specification

Un po' di esempi di design pattern



Singleton

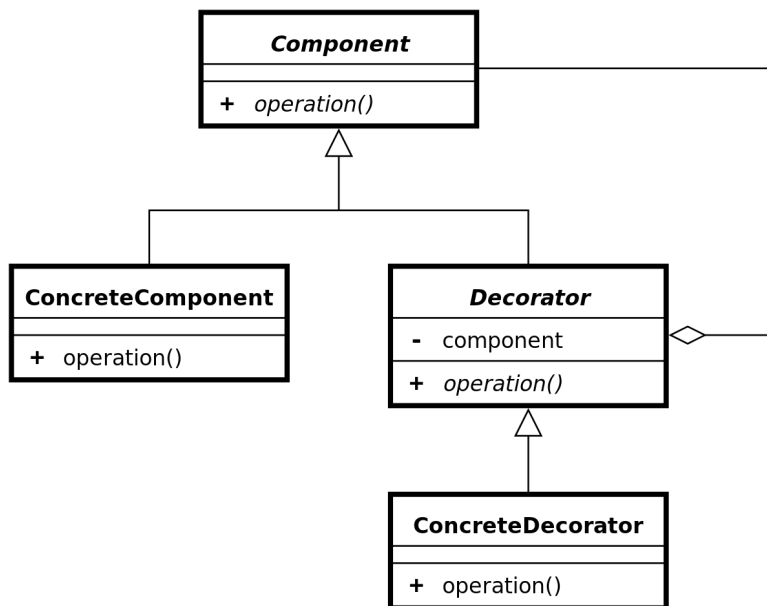
Garantisce che di una determinata classe venga creata una sola istanza a cui è possibile accedere globalmente.



Problematiche: frequentemente utilizzata in scenari dove non da alcun beneficio, introducendo restrizioni eccessive relative a una classe.

Decorator

Consente di aggiungere nuove funzionalità **dinamicamente** ad oggetti già esistenti.

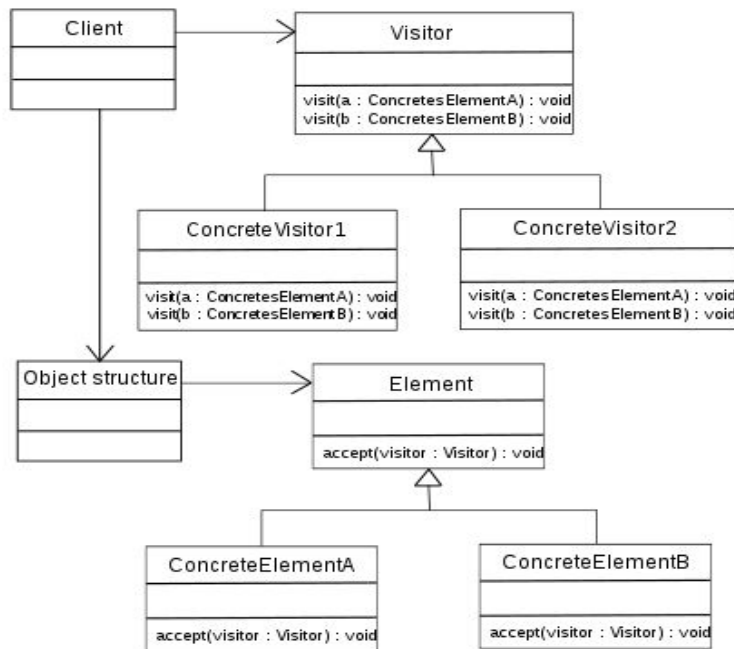


Il core del pattern si basa sugli elementi:

- Metodo `operation` che viene ridefinito per ogni figlio del `component` e del `decorator`
- Campo del `decorator` che è un elemento di tipo `component`

Visitor

Permette di definire nuove operazioni per classi appartenenti a una struttura senza modificare le classi.



Partecipanti al pattern:

- Visitor
- Elemento Visitabile
- Consumatore che istruisce gli elementi visitabili ad accettare il visitatore

Design pattern Architeturali



Design pattern architetturali

È un altro tipo di design pattern.

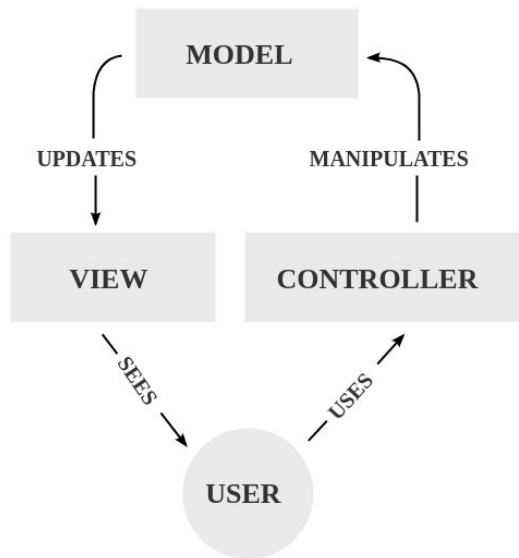
È una soluzione generale e riutilizzabile per un problema che si verifica comunemente nell'architettura del software all'interno di un determinato contesto.

Affrontano vari problemi, come le limitazioni delle prestazioni dell'hardware del computer o la minimizzazione di un rischio aziendale.

- **Client-server**
- **Data access object (DAO)**
- **Data transfer object (DTO)**
- **Inversion of control**
- Microservices
- **Model-view-controller (MVC)**
- Monolithic
- Publish-subscribe

MVC

È comunemente usato per lo sviluppo di interfacce utente, dividendo la relativa logica del programma in tre elementi interconnessi.



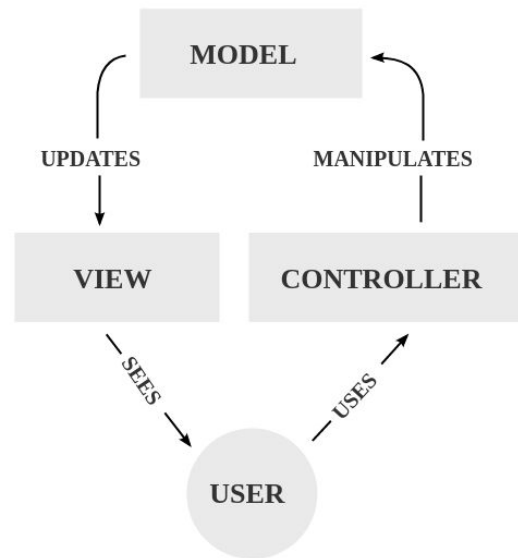
Usato per separare le rappresentazioni interne delle informazioni.

I componenti di questo pattern sono:

- Model
- View
- Controller

Componenti MVC

- Model = Componente centrale del modello.
Gestisce direttamente i dati, la logica e le regole dell'applicazione.
- View = Rappresenta le informazioni come un grafico, un diagramma, una tabella o, come nel nostro caso, un oggetto.
- Controller = Accetta gli input e li converte in comandi per il modello o la vista.



Q&A

