SIMS - VEŽBE 05

MVC I OBSERVER ŠABLONI

ARHITEKTURALNI OBRAZCI

- Arhitektonski obrasci predstavljaju opšta i višekratno primenjiva rešenja za uobičajene probleme u arhitekturi softvera, unutar određenog konteksta
- Oni pomažu u projektovanju softverskih sistema tako što nude proveren način organizacije komponenti, komunikacije među njima i raspodele odgovornosti
- Arhitektonski obrasci se koriste za rešavanje:
  - Ograničenja performansi i kapaciteta hardvera
  - Postizanja visoke dostupnosti sistema
  - Minimizacije poslovnog rizika kroz pouzdane softverske strukture
  - Optimizacije skalabilnosti, održavanja i fleksibilnosti sistema

ARHITEKTURALNI OBRAZCI

- Neki od najčeščih šablona su:
    1. Layered pattern
    2. Client-server pattern
    3. Master-slave pattern
    4. Pipe-filter pattern
    5. Broker pattern
    6. Peer-to-peer pattern
    7. Event-bus pattern
    8. Model-view-controller pattern
    9. Blackboard pattern
    10. Interpreter pattern

DIZAJN OBRAZCI

- Dizajn paterni su ponovo iskoristiva rešenja koja se primenjuju na probleme koji se često ponavljaju tokom razvoja softvera. Kod nas se često nazivaju i "projektni uzorci".
- Umesto da svaki put kreiramo rešenje od nule, dizajn paterni nam nude proverene i efikasne modele rešavanja određenih problema u softverskom dizajnu.
- Tri glavne prednosti dizajn paterna:

1. **Dokazana rešenja**
   - Dizajn paterni su rezultat prakse i iskustva velikog broja programera
   - Predstavljaju isproban i proveren pristup rešavanju čestih problema
2. **Ponovna iskoristivost**
   - Paterni su često "gotova rešenja" koja možemo prilagoditi konkretnim potrebama aplikacije
   - To ubrzava razvoj i smanjuje rizik od grešaka
3. **Ekspresivnost**
   - Svaki patern ima prepoznatljivu strukturu i terminologiju
   - To omogućava lakšu komunikaciju i saradnju među programerima, jer svi razumeju o kom obrascu je reč i kako funkcioniše

DIZAJN OBRAZCI

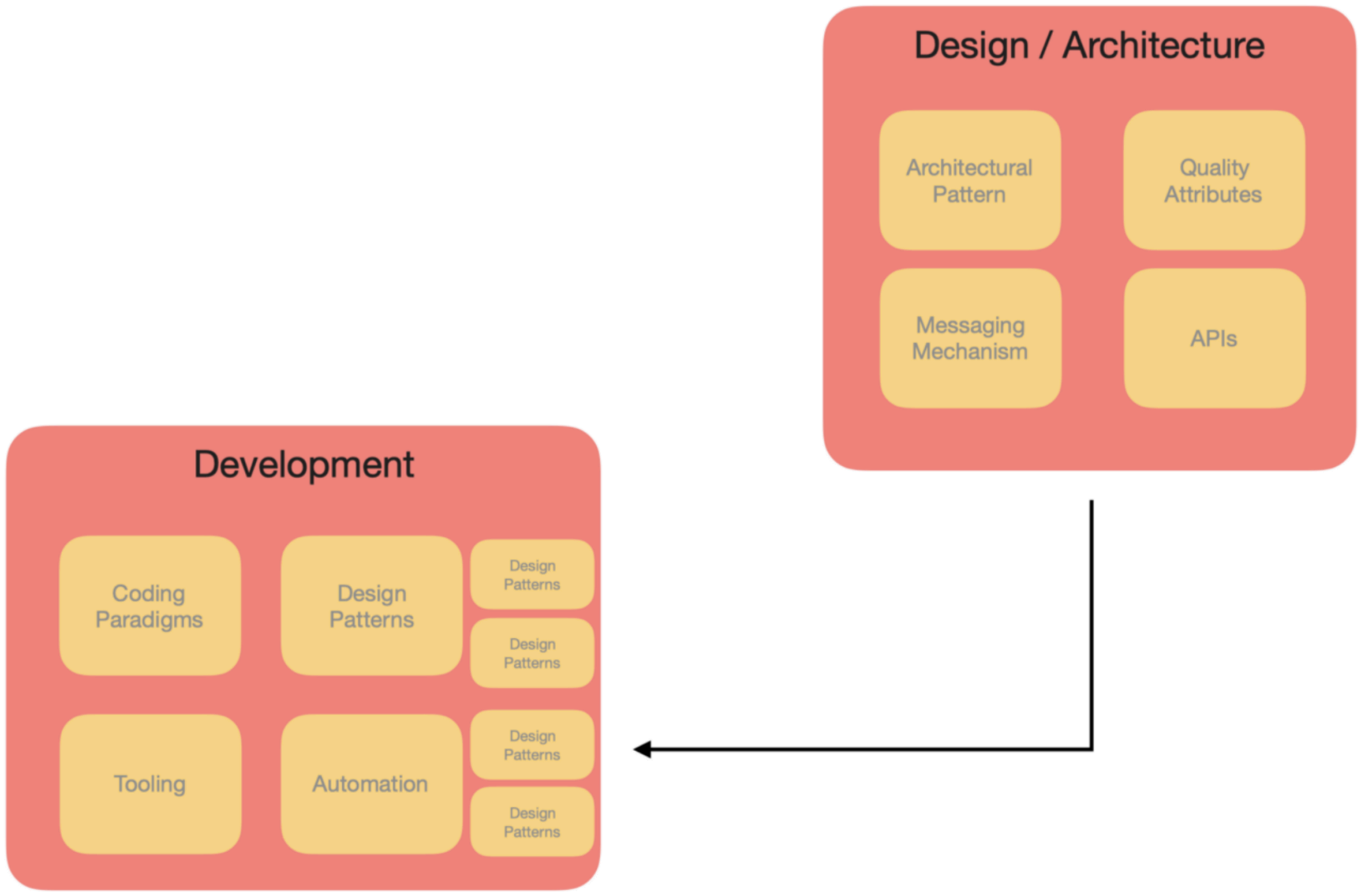
- Dizajn paterni se tradicionalno dele na tri kategorije, u zavisnosti od problema koje rešavaju:

1. **Kreacioni paterni**
   - Fokusirani su na način kreiranja objekata, uz kontrolu kompleksnosti i fleksibilnosti instanci koje kreiramo
   - Koriste se kada bi standardno kreiranje objekata (kroz new) vodilo ka prevelikoj zavisnosti ili složenosti
   - Neki od paterna koji spadaju u ovu grupu su: Constructor, Factory, Abstract, Prototype, Singleton and Builder
2. **Strukturalni paterni**
   - Bave se organizacijom i kompozicijom klasa i objekata
   - Omogućavaju da se promene u jednom delu sistema ne preliju na ostatak sistema, tj. sistem ostaje stabilan i fleksibilan
   - Pomažu da svaki deo sistema radi ono za šta je najprikladniji
   - Neki od strukturalnih paterna su: Decorator, Facade, Flyweight, Adapter i Proxy.
3. **Bihevioralni paterni**
   - Fokusirani su na poboljšanje komunikacije i interakcije između objekata
   - Definišu načine razmene poruka, saradnje i obrade događaja među objektima
   - Poznati primeri su: Iterator, Mediator, Observer i Visitor

RAZLIKE

- Razlika između softverske arhitekture i dizajn paterna
  - Arhitektura softvera definiše širu sliku sistema – kako su komponente organizovane, kako komuniciraju i koje tehnologije i alati se koriste.
  - Dizajn paterni definišu rešenja na nivou implementacije, tj. kako će pojedinačne klase, objekti i metode biti strukturirani da reše konkretne probleme.
- Arhitektura određuje:
  - Komponente sistema
  - Fizičku lokaciju komponenti (npr. klijent, server)
  - Tehnologije i alate koje će se koristiti
  - Pravila komunikacije (protokoli, API-jevi)
- Dizajn određuje:
  - Kako se implementiraju pojedinačne komponente
  - Strukturu klasa, objekata, interfejsa
  - Principi OOP-a: enkapsulacija, nasledjivanje, polimorfizam
  - Primenu SOLID principa
  - Upotrebu dizajn paterna za specifične potrebe (npr. kreiranje objekata, povezivanje, obrada događaja)

PUT RAZVOJA


Design / Architecture
Architectural Pattern
Quality Attributes
Messaging Mechanism
APIs
Development
Coding Paradigms
Design Patterns
Design Patterns
Design Patterns
Tooling
Automation
Design Patterns
Design Patterns

PROBLEM KORISNIČKOG INTERFEJSA

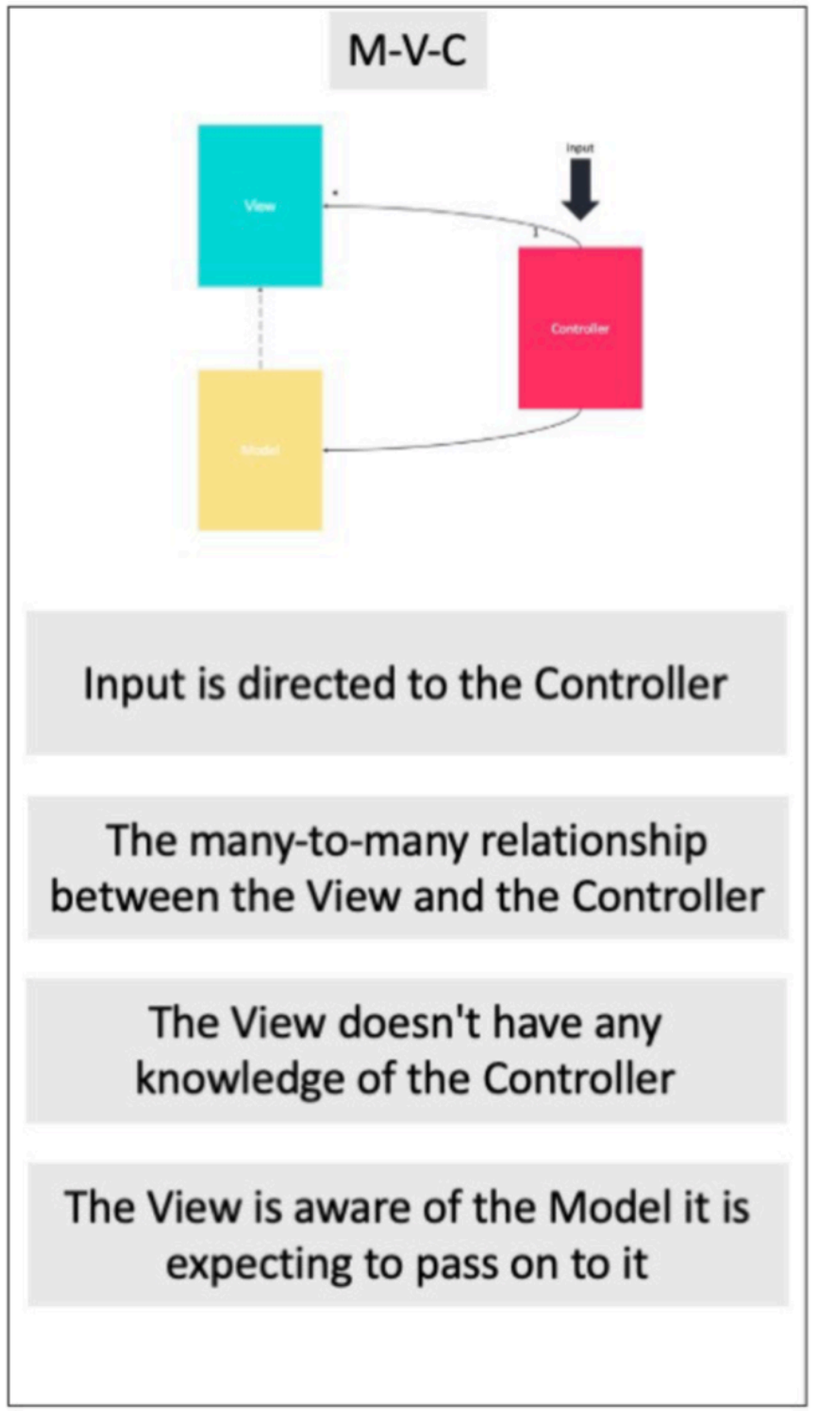
- Nakon što su računari izašli iz domena striktno vojne primene, postali su predmet interesovanja studenata tehničkih nauka, matematičara i naučnika
- Njihova šira upotreba bila je ograničena zbog složenog načina rada, koji je zahtevao unos instrukcija putem komandne linije
- Da bi računari postali pristupačniji širokom krugu korisnika, bilo je neophodno razviti grafički korisnički interfejs (GUI), koji bi omogućio lakšu i intuitivniju interakciju.
- U okviru razvoja programskog jezika Smalltalk-80 u istraživačkom centru Xerox PARC, Trigve Reenskaug je osmislio arhitektonski obrazac poznat kao Model-View-Controller (MVC)
- On je primetio da se programska arhitektura aplikacija sa grafičkim interfejsom može efikasnije organizovati ako se podeli na tri međusobno povezana dela:
  - Model: zadužen za upravljanje podacima i poslovnom logikom,
  - View: odgovoran za prikaz podataka korisniku,
  - Controller: posreduje između korisničkog inputa i modela, upravljajući reakcijama sistema

MVC

- Model-View-Controller (MVC) je arhitektonski obrazac koji se koristi u razvoju softvera
- U složenim aplikacijama koje prikazuju korisniku ogromne količine podataka programeri često žele da razdvoje kod koji se bavi podacima od onog koji se bavi interfejsom, tako da razvoj oba postane lakši i jednostavniji
- MVC rešava ovaj problem razdvajanjem podataka i biznis logike od njihovog prikaza i interakcije sa korisnikom, uz to uvodeći i komponentu zaduženu za koordinisanje prve dve
- Tri segmenta:
  - **Model** - model
  - **View** - pogled
  - **Controller** - kontroler
- Ciljevi:
  - Razdvajanje odgovornosti u kodu
  - Lakše održavanje i izmene
  - Jednostavnije testiranje pojedinačnih komponenti
  - Paralelan rad različitih timova (front-end/backend)

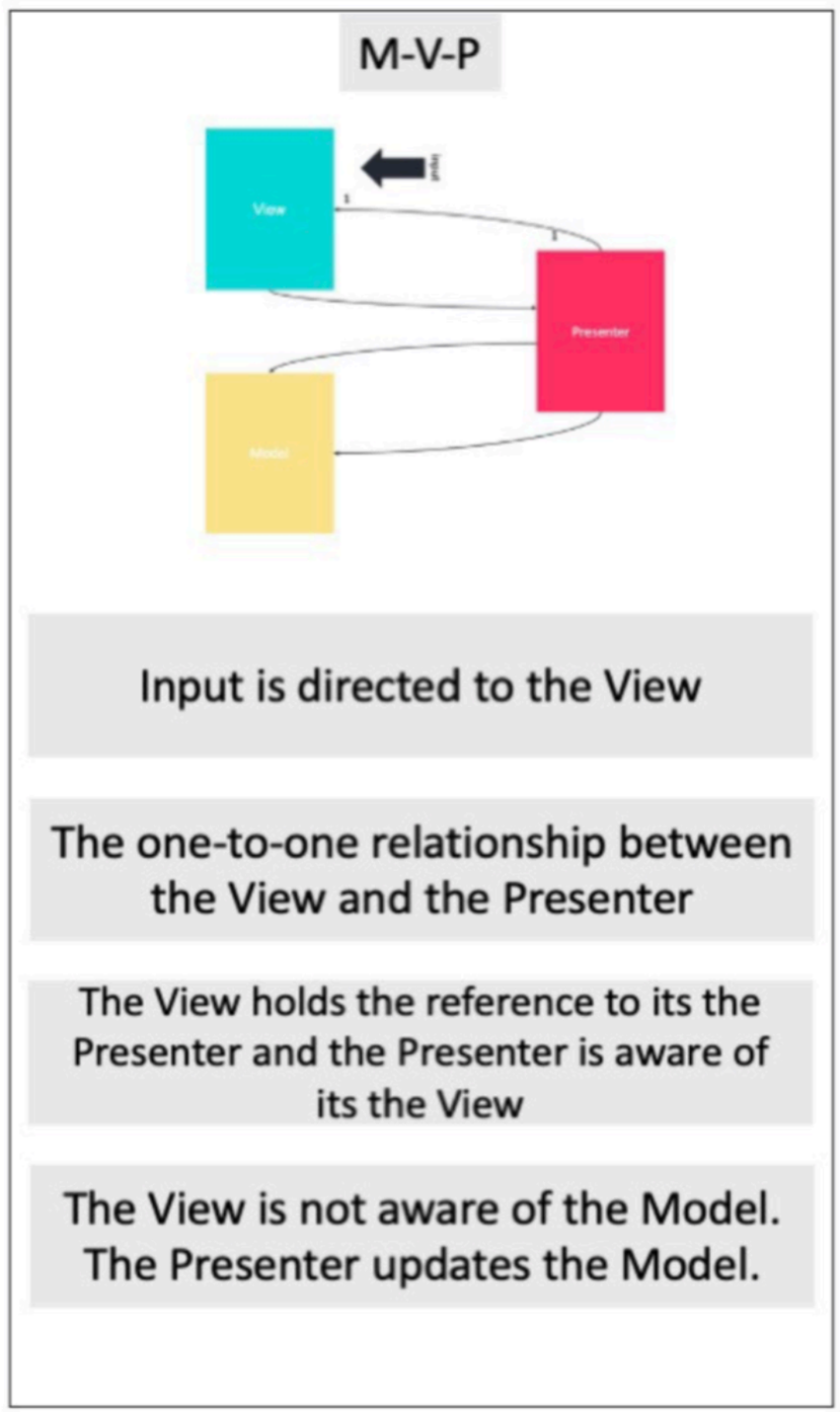
VERZIJE MVC

- Postoje različite verzije MVC šablona:
    1. Osnovni MVC
        - **Model**: Podaci i poslovna logika
        - **View**: Prikaz korisniku
        - **Controller**: Prima korisnički input, ažurira Model i View
        - View komunicira direktno sa Modelom da bi prikazao podatke
        - Gde se koristi: Web aplikacije (npr. Spring MVC, Ruby on Rails)



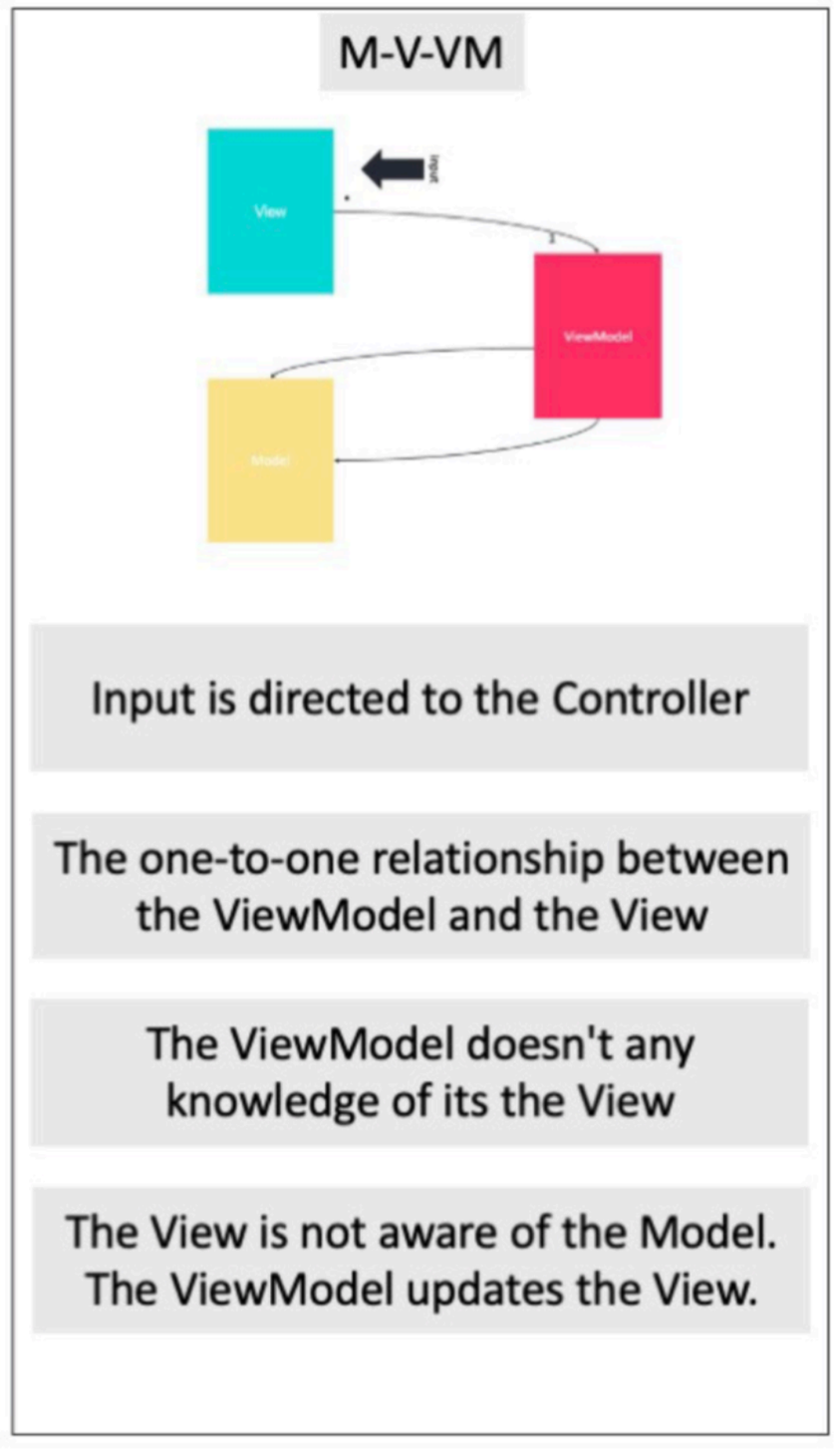
VERZIJE MVC

- Postoje različite verzije MVC šablona:
    2. MVP (model - viewer - presenter)
        - **Model**: Podaci i logika
        - **View**: Pasivan prikaz korisniku – ne komunicira direktno sa Modelom
        - **Presenter**: Posreduje između View i Modela
            - Prima input od View-a
            - Manipuliše Modelom
            - Ažurira View
- Prednost: Bolja za testiranje i složene korisničke interfejse
- Gde se koristi: Desktop aplikacije, Android

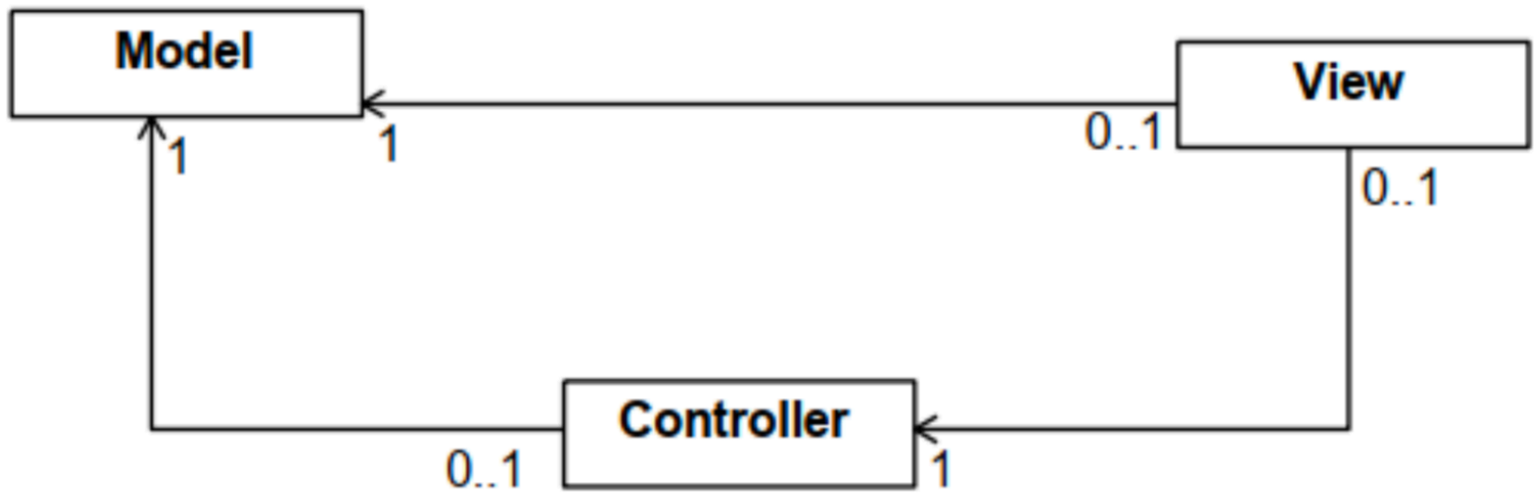


VERZIJE MVC

- Postoje različite verzije MVC šablona:
    3. MVVM (model - view - viewModel)
        - **Model**: Podaci i poslovna logika
        - **View**: Prikaz korisniku
        - **ViewModel**: Sadrži prezentacionu logiku i koristi data binding za automatsku sinhronizaciju između Modela i View-a
- Prednost:
    - Manje ručnog povezivanja između View i Modela
    - Automatsko ažuriranje View-a kada se Model promeni
- Gde se koristi: Moderni UI frejmvorci (npr. Angular, React, WPF, Xamarin)



OSNOVNI MVC ŠABLON



Osnovni MVC šablon

OSNOVNI MVC ŠABLON

- **Model (model)**
  - Sadrži podatke u formatu koji je pogodan za konkretnu aplikaciju
  - Obuhvata poslovnu logiku, odnosno definiše:
    - Šta možemo raditi sa podacima
    - Kako se podaci menjaju u skladu sa pravilima aplikacije
  - Model često koristi trajnu memoriju:
    - Npr. baze podataka, fajlovi, API-jevi
  - MVC ne propisuje način pristupa podacima:
    - To je detalj implementacije Modela
    - Pristup podacima je obično enkapsuliran unutar Modela i skriven od View-a i Controller-a
  - Model je nezavisan od View-a i Controller-a
    - Model ne zna ništa o prikazu (View), niti o načinu na koji su podaci zatraženi (Controller)
    - Time se postiže slaba sprega (low coupling)

OSNOVNI MVC ŠABLON

- **View (pogled)**
  - Ima zadatak da prikazuje podatke iz Modela u formatu pogodnom za interakciju sa korisnikom. Njegova osnovna uloga je da omogući komunikaciju između korisnika i aplikacije, pružajući vizuelni prikaz podataka i omogućavajući unos i pokretanje akcija
  - View je najčešće sastavljen od klasa korisničkog interfejsa
    - Za njegovu implementaciju obično se koristi neka grafička biblioteka poput Swing-a ili JavaFX-a
    - Unutar jedne aplikacije može postojati više različitih View-ova koji prikazuju podatke iz istog Modela, ali su prilagođeni različitim potrebama ili kontekstima upotrebe
- View ne sadrži poslovnu logiku, već je fokusiran isključivo na prezentaciju podataka i interakciju sa korisnikom. Kada korisnik izvrši neku akciju (npr. klik, unos teksta), View taj input prosleđuje Controller-u, koji odlučuje kako treba da se reaguje
- View je često dizajniran tako da bude reaktivan – kada se podaci u Modelu promene
  - View bi trebalo automatski da ažurira prikaz kako bi korisnik uvek video najnovije informacije
  - Kako ovo da postignemo?

OSNOVNI MVC ŠABLON

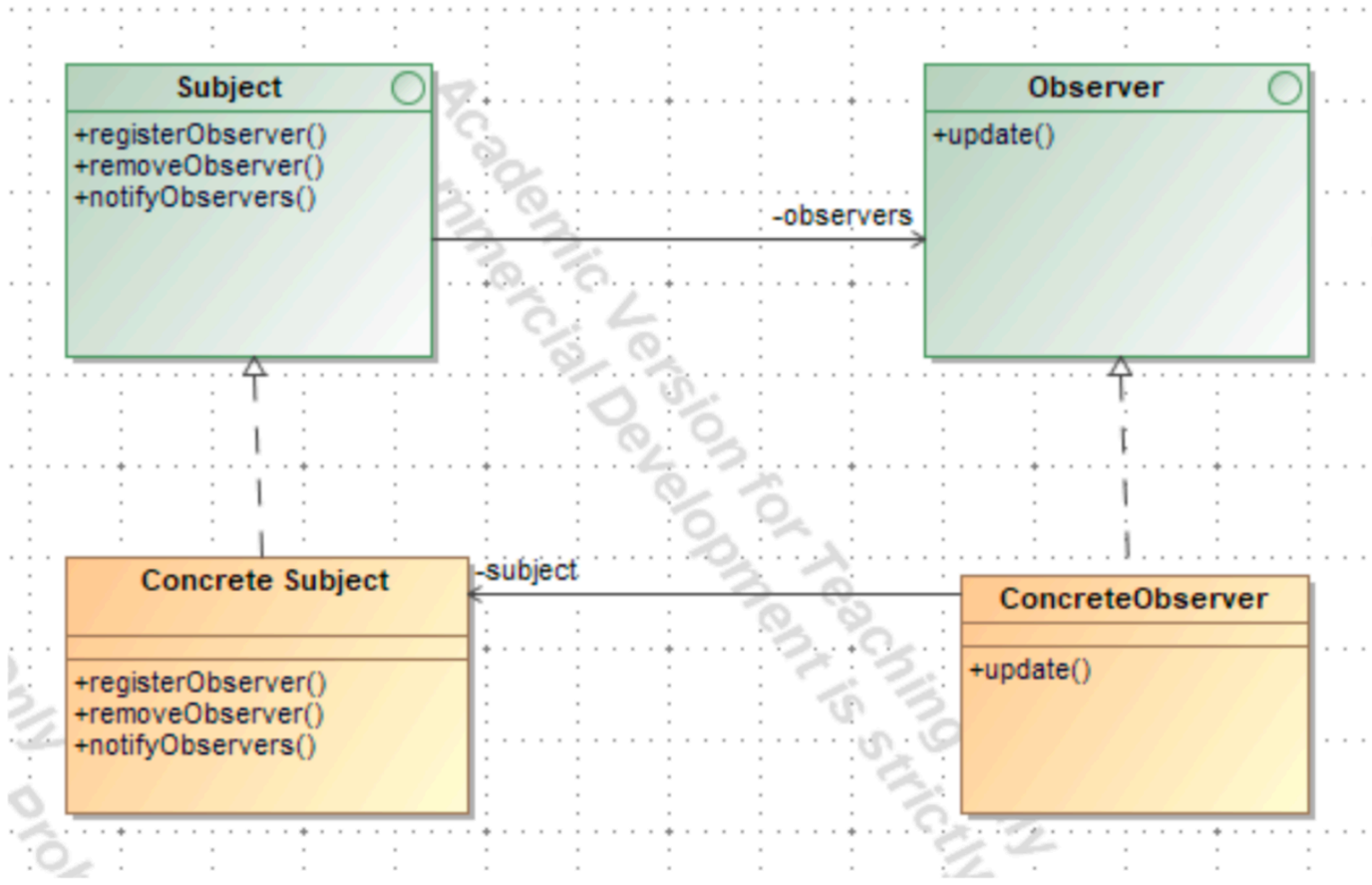
- **Controller (kontroler)**
  - Koordiniše komunikaciju između Modela i View-a, najčešće kao odgovor na korisnički unos. Njegov zadatak je da obradi događaje koje View registruje, kao što su klik na dugme, unos podataka ili izbor iz liste
  - Kada korisnik izvrši neku akciju, View obaveštava Controller, a Controller tada:
    - **Validira unete podatke**, proverava njihovu ispravnost
    - **Konvertuje podatke** u potreban format, ako je to neophodno za dalje procesiranje
    - **Poziva odgovarajuće metode Modela** koje implementiraju poslovnu logiku za tu akciju
    - U slučaju greške, obaveštava View o problemu, najčešće kroz bacanje izuzetaka ili druge mehanizme signalizacije, kako bi korisniku bio prikazan odgovarajući odgovor ili poruka o grešci
- Controller ne sadrži podatke niti ih direktno prikazuje – njegova odgovornost je obrada korisničkog inputa i upravljanje tokom podataka između View-a i Modela. Na ovaj način omogućava da Model ostane fokusiran na logiku i podatke, a View na prikaz i interakciju
- Controller se često implementira tako da bude lagan i specifičan za određene View-ove, što omogućava fleksibilnost i lakše testiranje ponašanja aplikacije

OBSERVER ŠABLON

- Observer je dizajnerski šablon koji definiše relaciju "jedan-prema-više" između objekata:
  - kada se jedan objekat (Subject / Observable) promeni,
  - svi objekti koji ga posmatraju (Observers) automatski se obaveštavaju o promeni
- Komponente Observer šablona:

1. **Subject (Observable)**
   - Objekat koji čuva stanje i ima listu svih svojih posmatrača
   - Ima metode za:
     - dodavanje/uklanjanje posmatrača
     - obaveštavanje svih posmatrača kada dođe do promene
2. **Observer**
   - Interfejs ili apstraktna klasa koju implementiraju svi objekti koji žele da prate promene
   - Sadrži metodu, npr. update(), koja se poziva kada Subject signalizira promenu

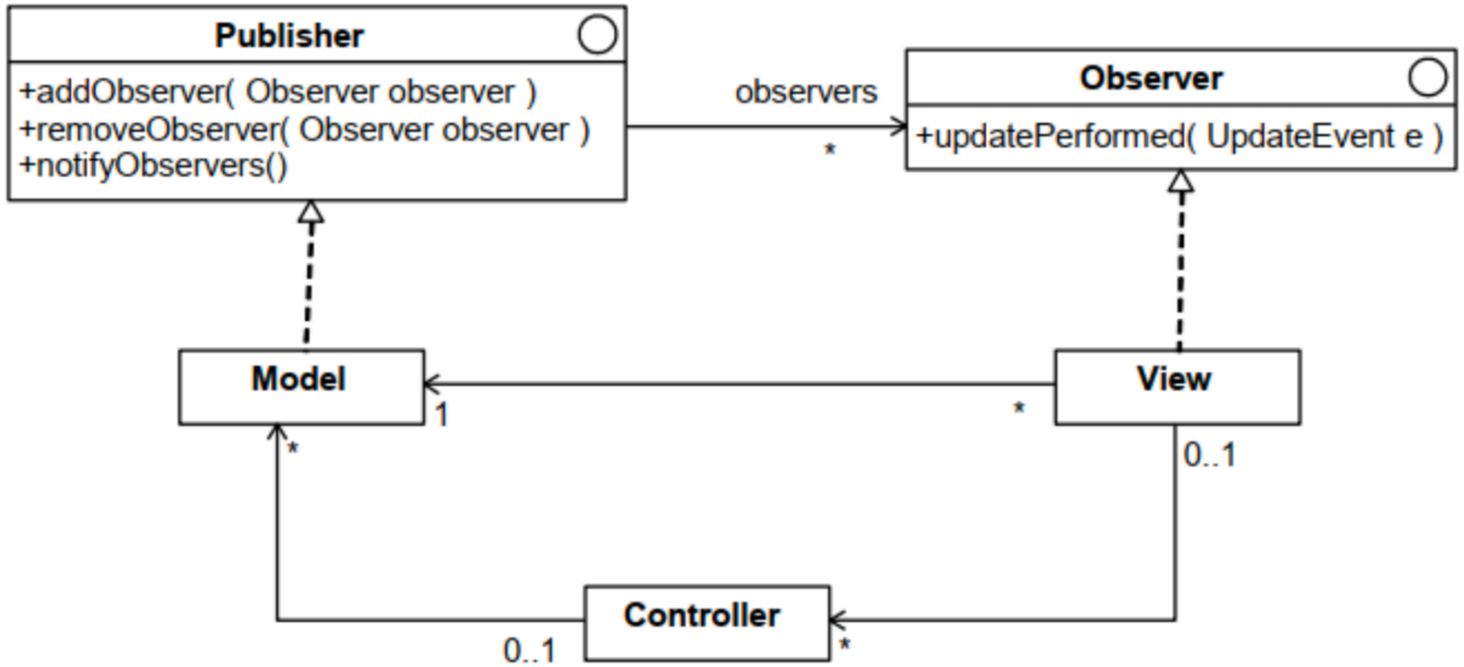
OBSERVER ŠABLON



Dijagram klasa *observer* šablona

MVC + OBSERVER ŠABLON

- Observer šablon se koristi kad je potrebno da se podaci iz modela prikazuju i menjaju na različite načine (tabela, stablo, djalog...). Tada osnovni MVC šablon proširujemo Observer šablonom.



MVC proširen *Observer* projektnim šablonom