A VERY CLEAN UNCLE

Strukturiranje koda i primjena MVP arhitekturalnog obrasca

UNCLE BOB

Clean arhitektura i SOLID principi

R. C. MARTIN (UNCLE BOB)

- Programira jedno 100 godina
- Agile manifesto
- Clean Code
- Clean Coder
- Clean Architecture
- TDD
- ...

ŠTO JE CLEAN

- Mindset koji nam pomaže strukturirati kod za druge ljude
- Prati SOLID principe
- Primjer : “Bolje je 5 minuta imenovati varijablu/ metodu kvalitetno, nego kasnije mijenjati”
- Development je 10% programiranje, 90% komunikacija — kod se najviše čita, najmanje piše

SOLID PRINCIPI

- S - Single responsibility princip
- O - Open-Closed princip
- L - Liskov substitution princip
- I - Interface segregation princip
- D - Dependency inversion princip

SINGLE RESPONSIBILITY

- Svaka stavka treba raditi jednu stvar i samo jednu stvar (i da i ne?)

- Stavka — od velike komponente pa do najužeg dijela koda koji možete konstruirati (varijabla/metoda)

- Mijenjanjem te stavke nećemo poremetiti ostatak koda (ujedno i Open-Closed princip)

OPEN CLOSED PRINCIP

- “Everything must be open for extension, but closed for modification.”

- Kad pravimo nekakav API - neko sučelje - treba biti takav da, ako mu dodamo novo ponašanje, ne smije se pokidati ništa drugo

- Ako promijenimo jedno od ponašanja, ostala bi trebala ostati netaknuta (i da i ne?)

LISKOV SUBSTITUTION PRINCIPLE

- Ako izgleda kao patka, glasa se kao patka, onda je to patka

- Svaki tip koji koristimo se treba moći zamijeniti s jednom od supertipova ili subtipova bez da se nešto posebno pokida

- Tipičan primjer : klasa Osoba i subtipovi Student i Profesor — trebamo moći koristiti samo Osobu pomoću objekata P i S, kako bi ono radilo za oba slučaja

INTERFACE SEGREGATION PRINCIPLE

- Low coupling, high cohesion
- Bolje je napraviti više manjih interface-a i implementirati ih sve, nego jedan veći general-purpose interface
- Veći interface -> moramo implementirati metode koje nam ne trebaju
- Puno manjih interfaceova -> samo implementiramo što nam treba

DEPENDENCY INVERSION PRINCIP

- Apstrakcije ne bi trebale ovisiti o detaljima
- Detalji ovise o apstrakcijama
- “Low level” stvari ne trebaju ovisiti o “high level stvarima”
- Bolje je ovisiti o interfaceu, nego o konkretnoj klasi

MVP PATTERN

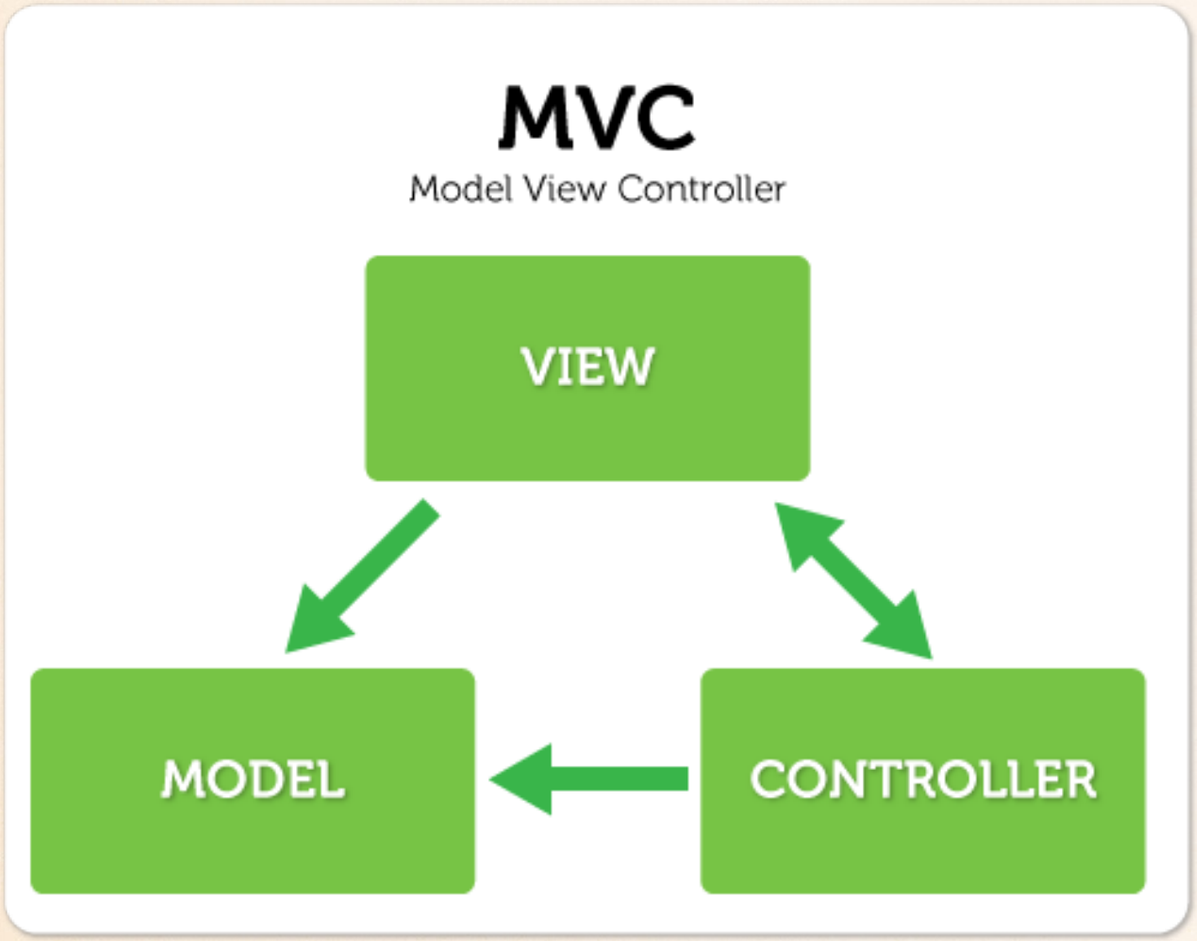
- Obrazac koji se koristi kako bi se bolje strukturirala aplikacija

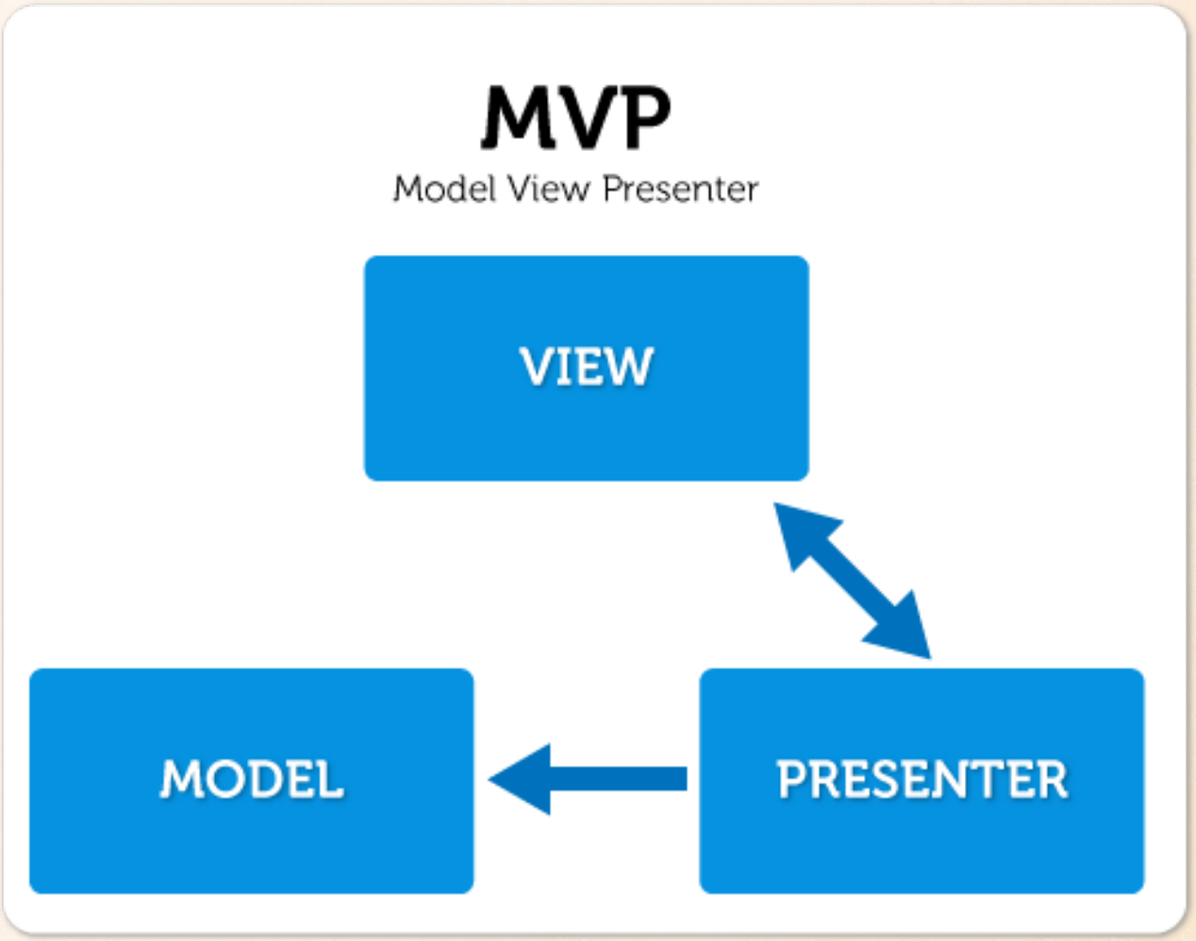
- Model - Vanjski entiteti (Baza, Networking..)

- Presenter - Sva logika, flow control, rad s podacima

- View - "UI" sloj, Android platforma

MVC <-> MVP


MVC
Model View Controller
VIEW
MODEL
CONTROLLER


MVP
Model View Presenter
VIEW
MODEL
PRESENTER

ZAŠTO?

- Odvajanje koda nam olakša refaktoriranje, testiranje, proširivanje funkcionalnosti..
- Ljepše je! Nemamo God-object klase - klase s po 500-1000 linija koje rade sve
- Neke stvari postaju reusable (vanjski slojevi)

ZADAĆA

- Weather appovi! Skinite weather app projekt s mog githuba i prebacite kod u MVP strukturiranu aplikaciju, koristeći alate koje smo do sad prošli (Retrofit, bazu za cacheiranje, Butterknife,...)