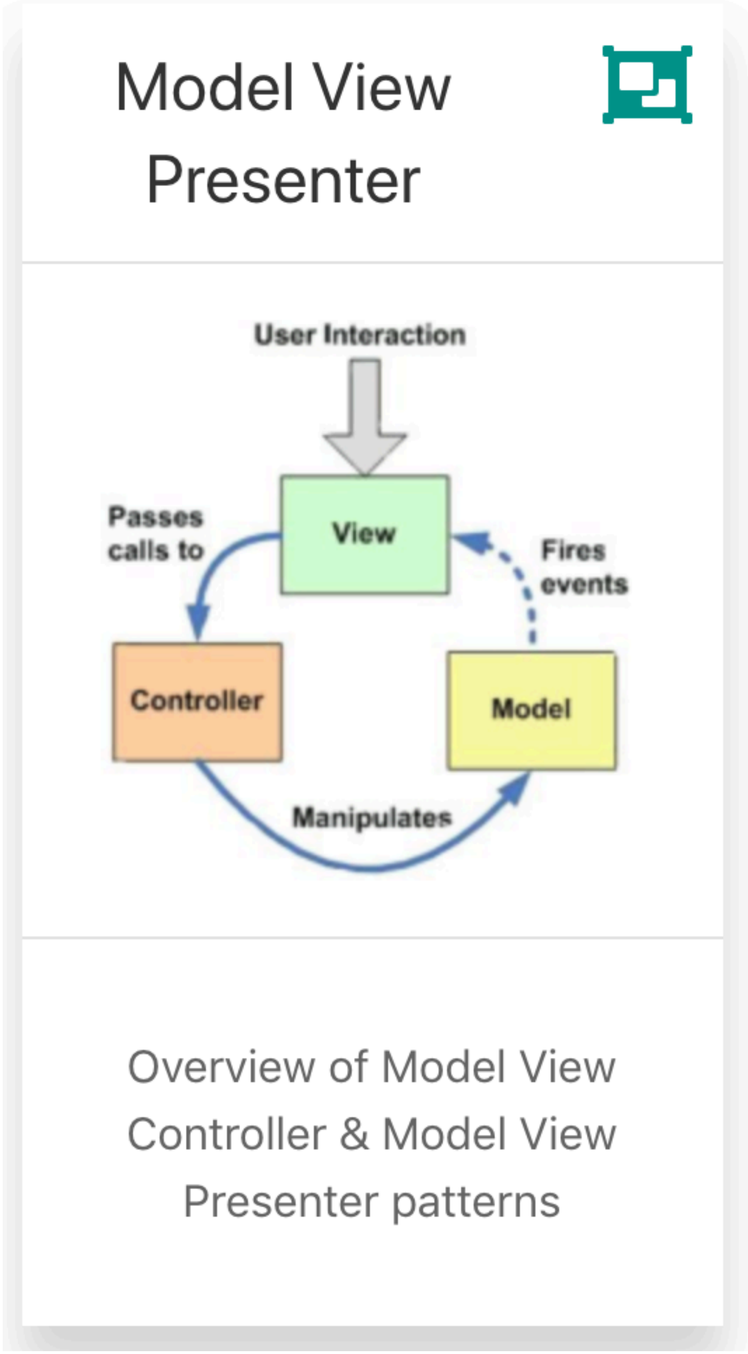
Model View Presenter



Separation of concerns

In computer science, separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern.

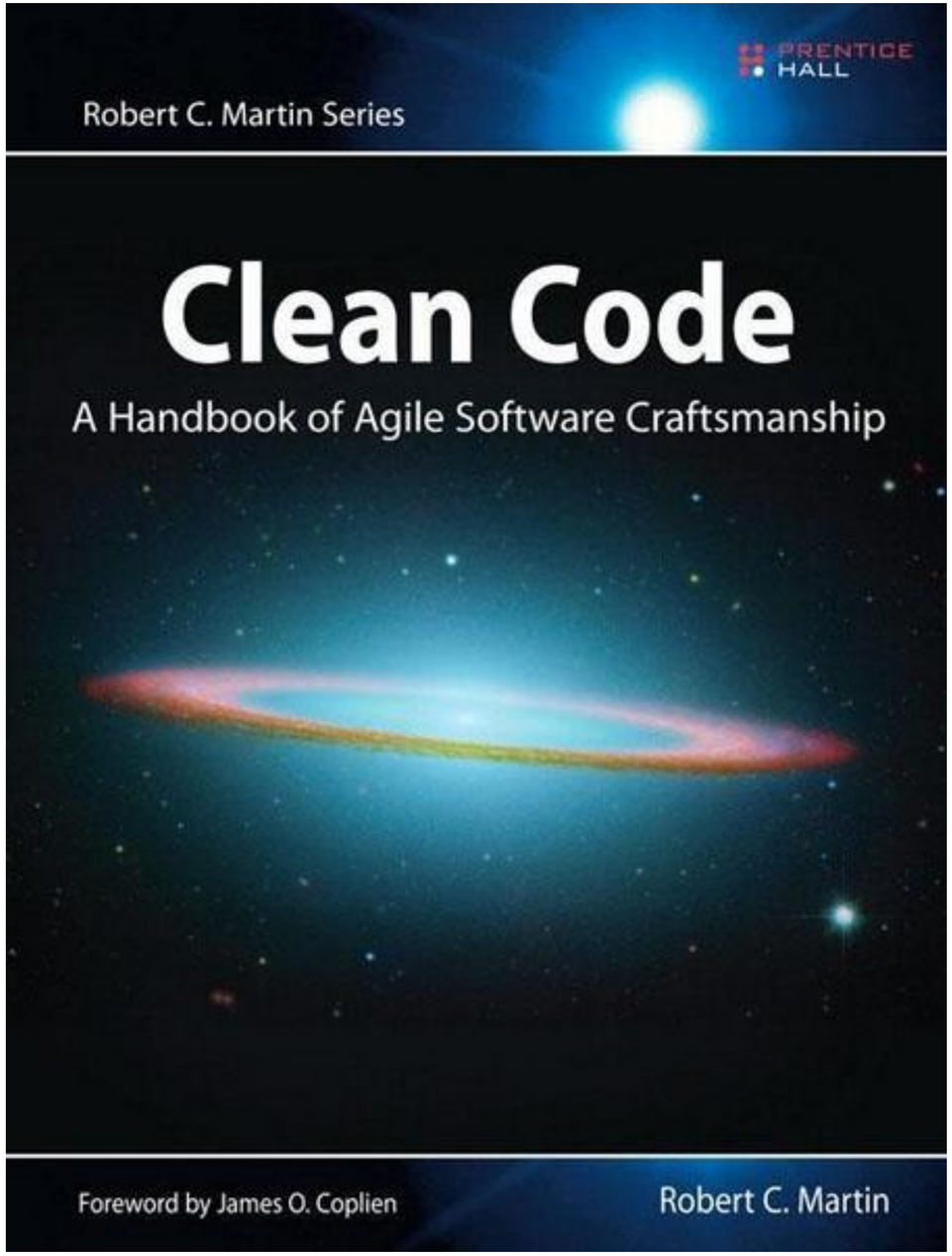
A concern is a set of information that affects the code of a computer program.

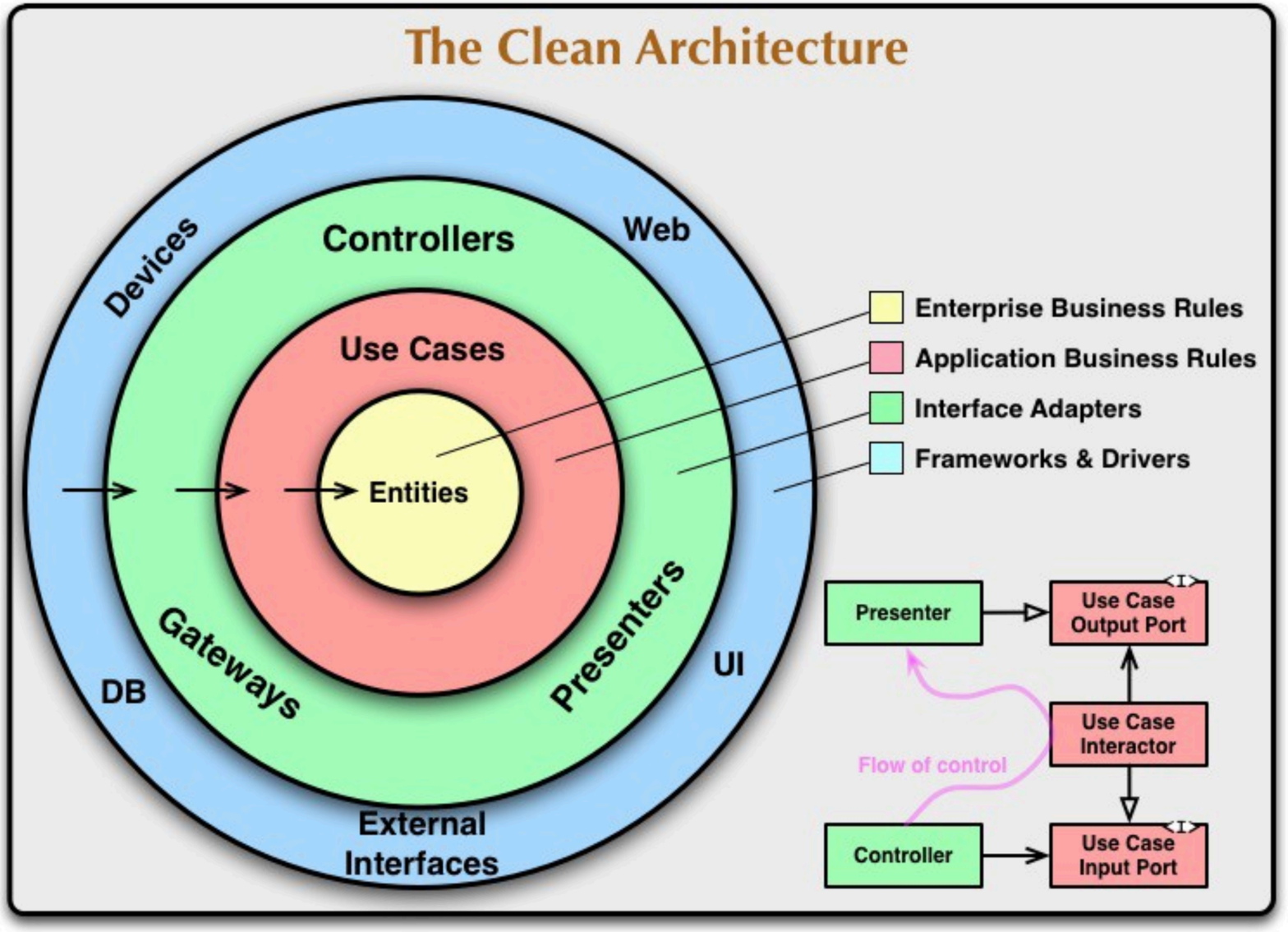
https://en.wikipedia.org/wiki/Separation_of_concerns

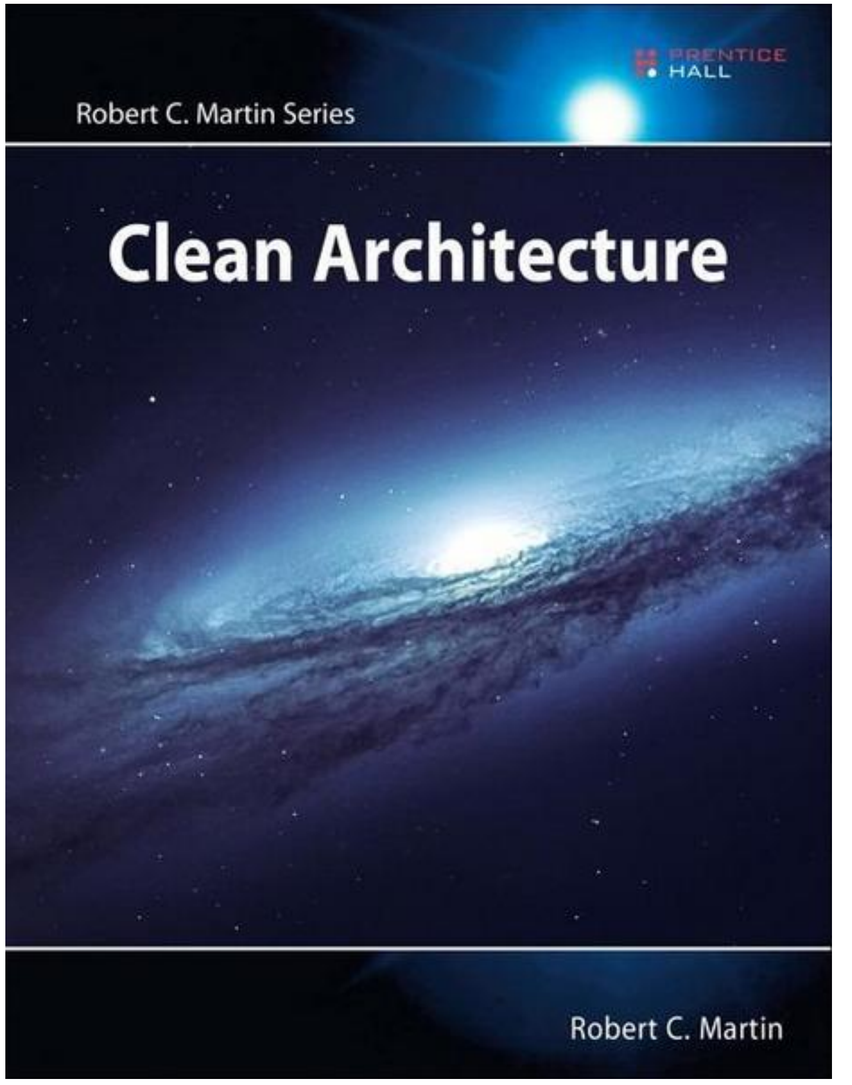
Proper MVP and MVC implementations have the following characteristics:

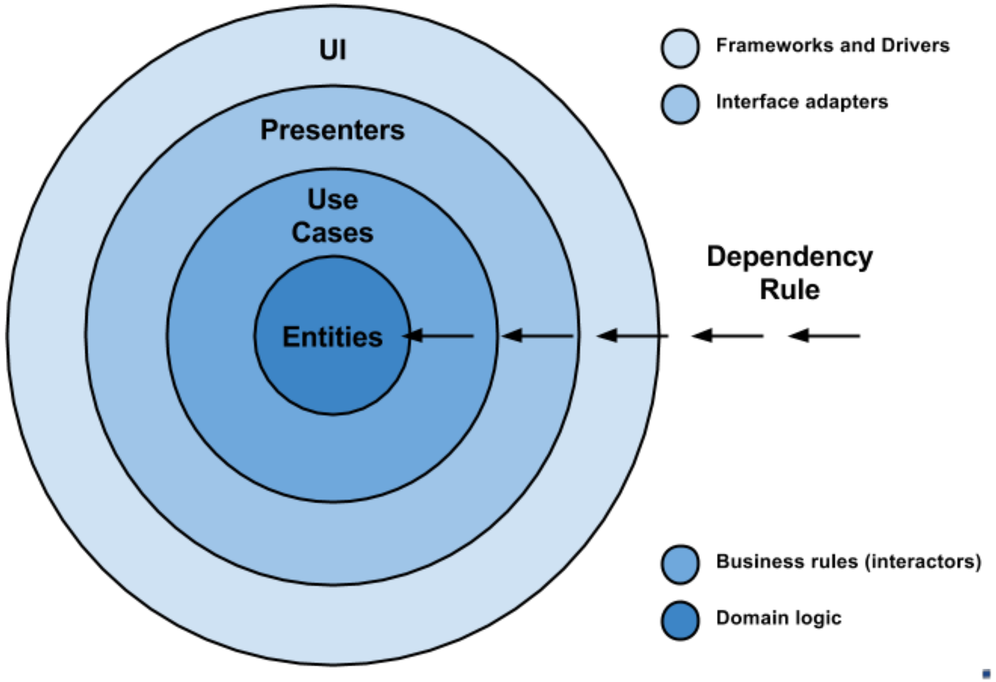
- Readable and maintainable code
- Modular code which provides high degree of decoupling
- More testable code
- Code which is fun to work with

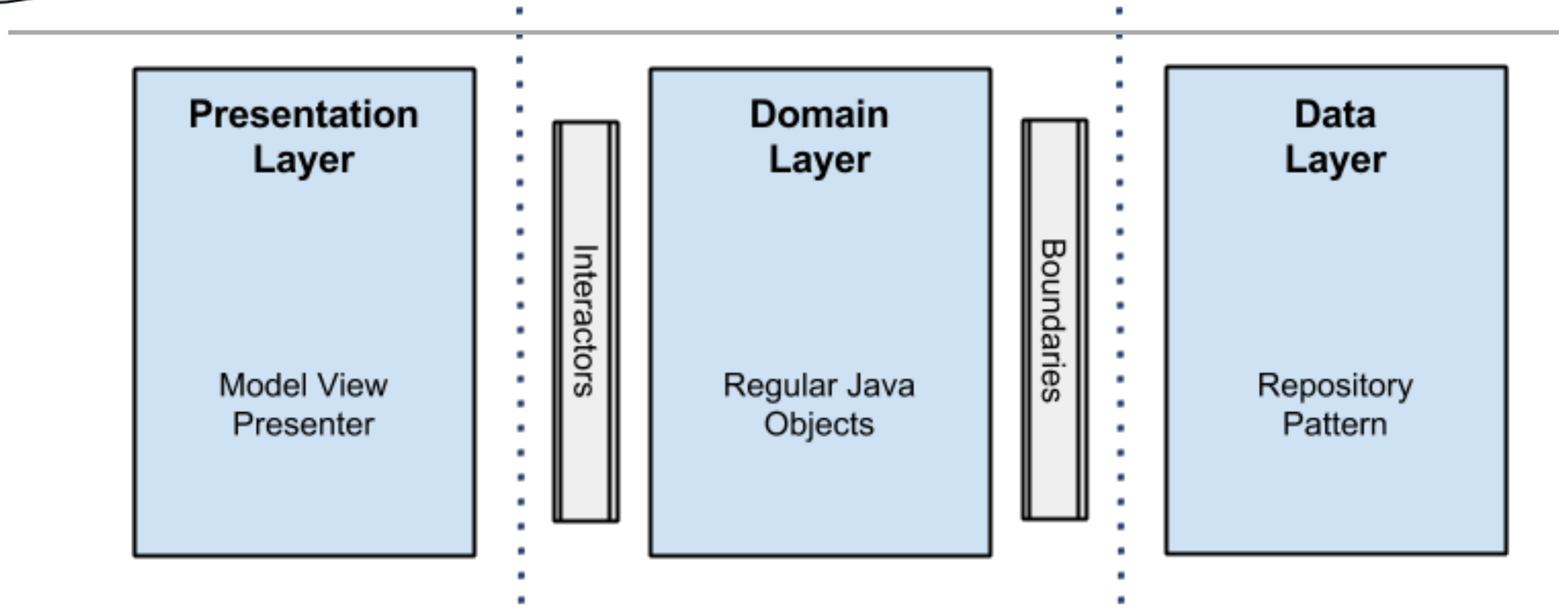
The above characteristics are generally associated with “clean code”.




The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port


Robert C. Martin Series
PRENTICE
HALL
Clean Architecture
Robert C. Martin


UI
Presenters
Use Cases
Entities
Frameworks and Drivers
Interface adapters
Dependency Rule
Business rules (interactors)
Domain logic

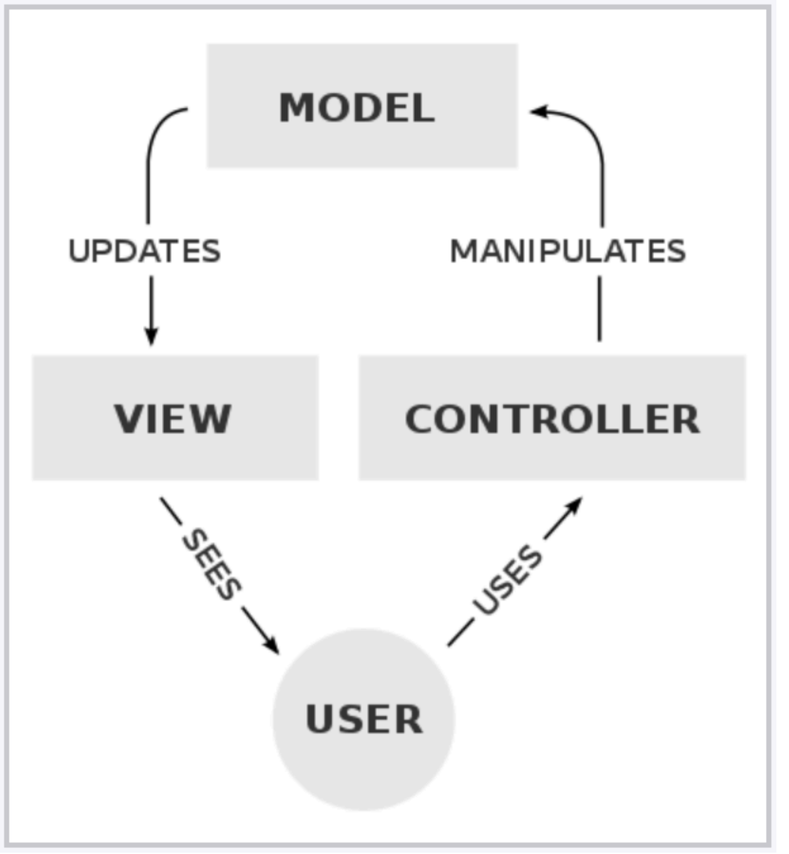

Presentation Layer
Model View Presenter
Interactors
Domain Layer
Regular Java Objects
Boundaries
Data Layer
Repository Pattern

Model–view–controller

From Wikipedia, the free encyclopedia

Model–view–controller is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.[1][2] The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Traditionally used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications and even mobile, desktop and other clients.[3] Popular programming languages like Java, C#, Ruby, PHP have MVC frameworks that are used in web application development straight out of the box.

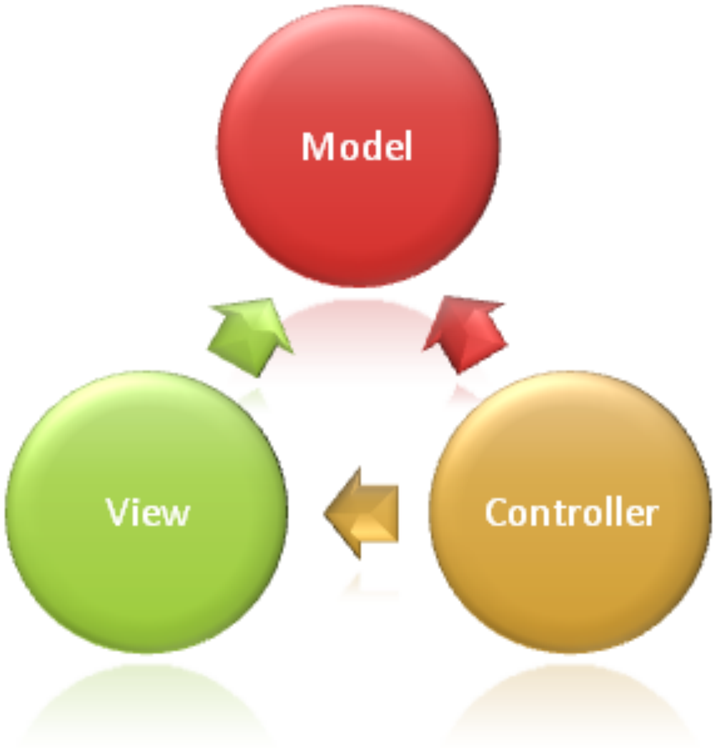


Components

- The **model** is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

- A **view** can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

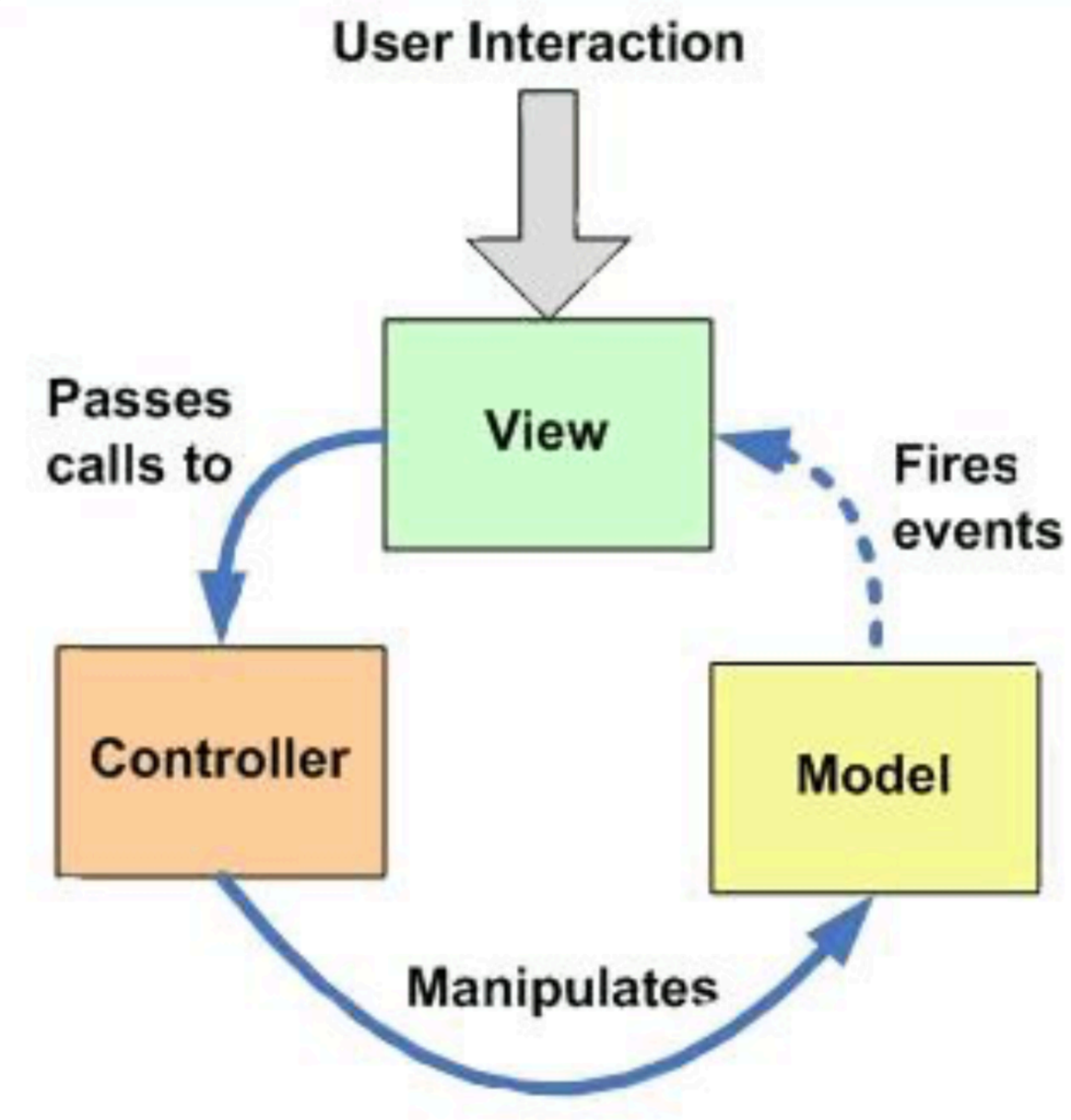
- The **controller**, accepts input and converts it to commands for the model or view



Interactions

In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.

- The **model** is responsible for managing the data of the application. It receives user input from the controller.
- The **view** means presentation of the model in a particular format.
- The **controller** responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

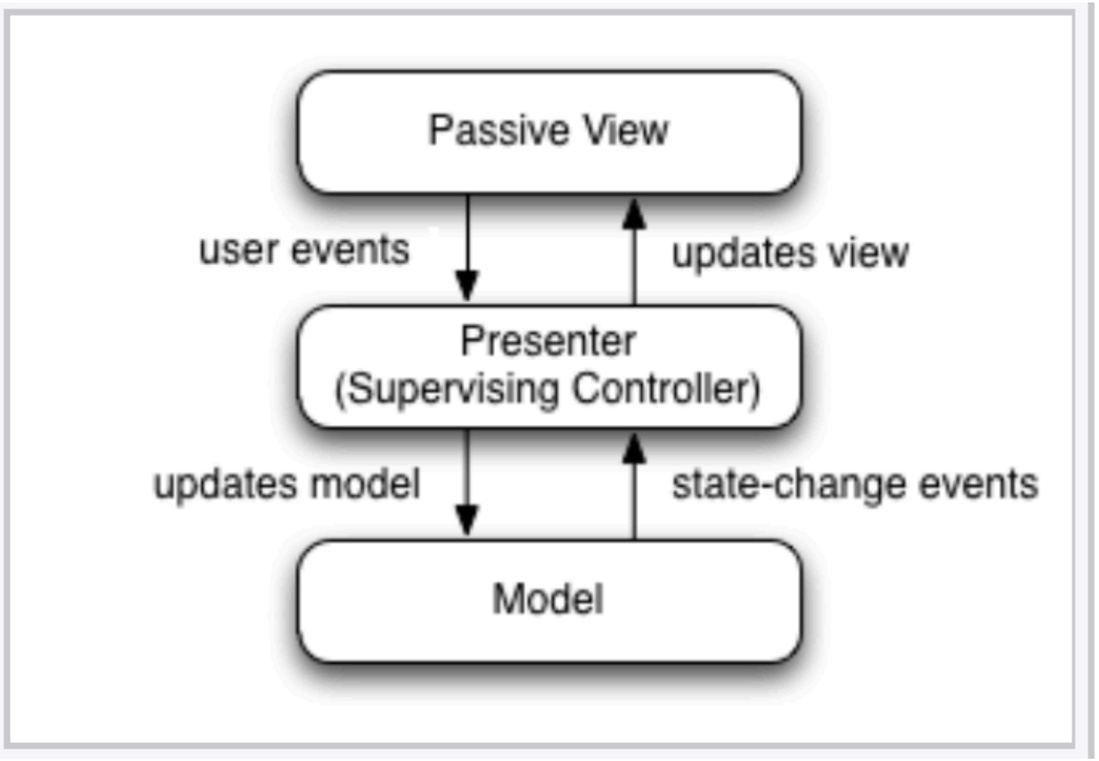


Model–view–presenter

From Wikipedia, the free encyclopedia

Model–view–presenter (**MVP**) is a derivation of the model–view–controller (MVC) architectural pattern, and is used mostly for building user interfaces.

In MVP, the *presenter* assumes the functionality of the "middle-man". In MVP, all presentation logic is pushed to the presenter.[1]

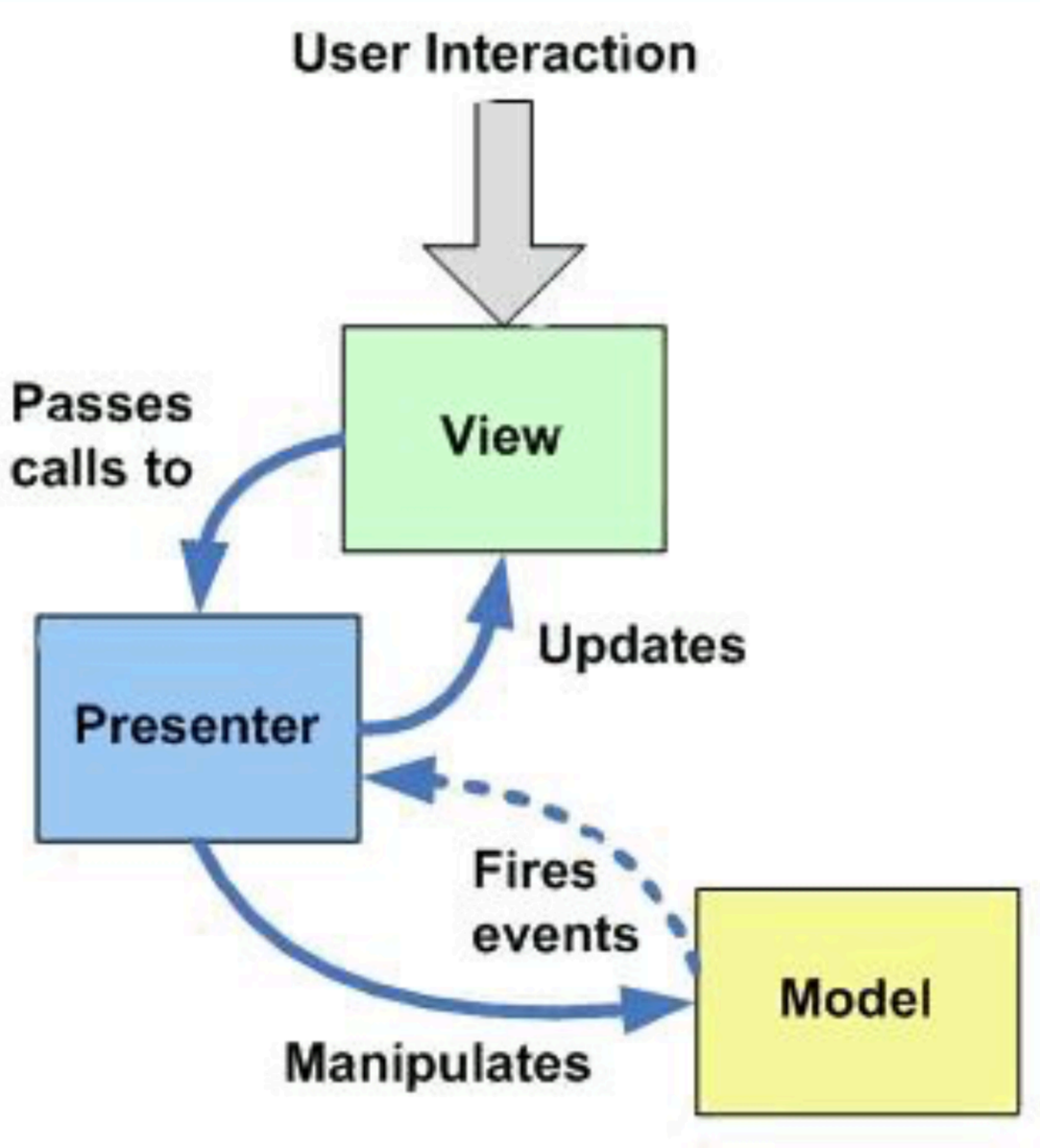


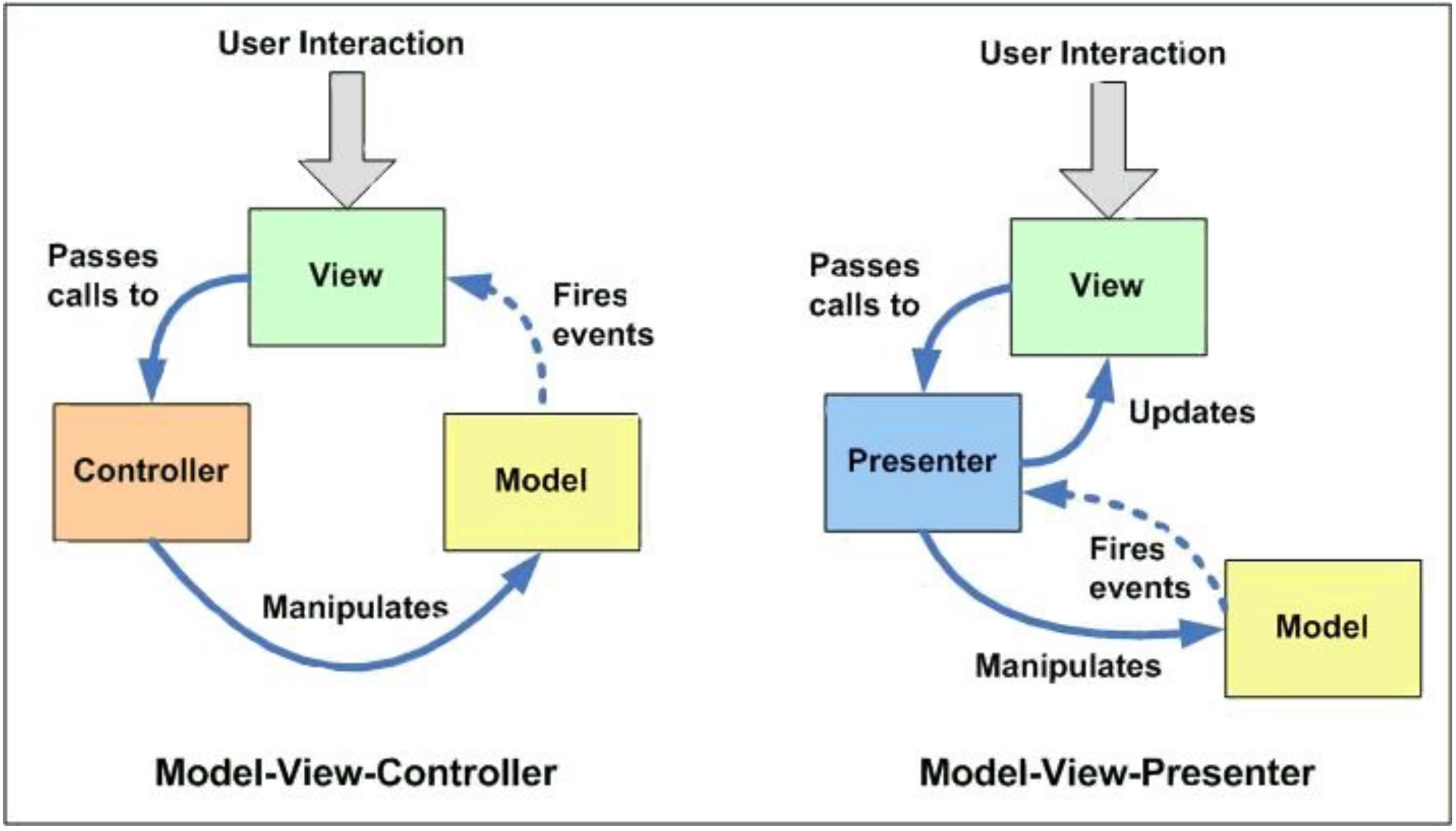
Components

MVP is a user interface architectural pattern engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic:

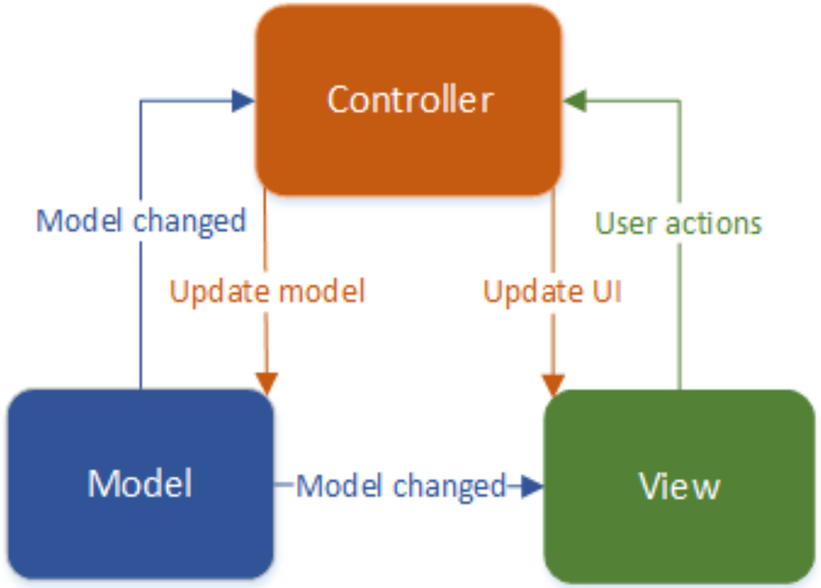
- The **model** is an interface defining the data to be displayed or otherwise acted upon in the user interface.
- The **view** is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.
- The **presenter** acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

Normally, the view implementation instantiates the concrete presenter object, providing a reference to itself.

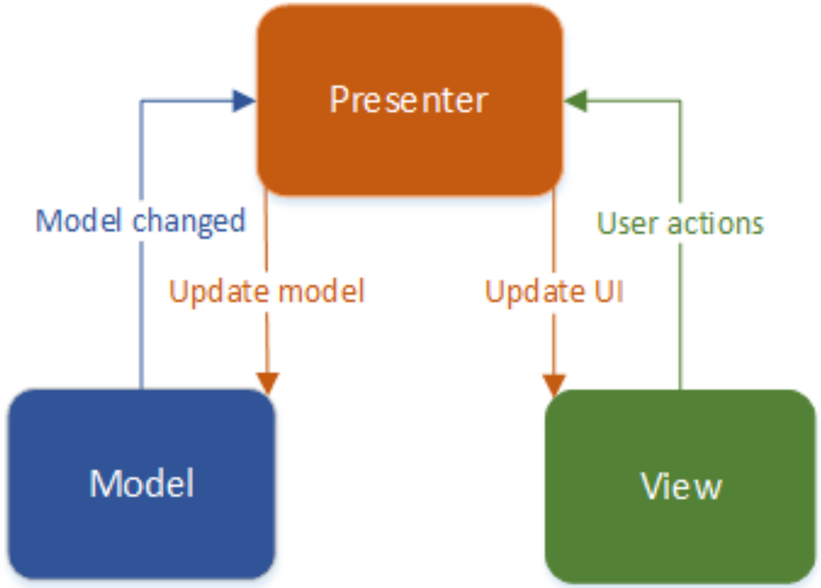



User Interaction
View
Passes calls to
Fires events
Controller
Model
Manipulates
Model-View-Controller
User Interaction
View
Passes calls to
Updates
Presenter
Fires events
Model
Manipulates
Model-View-Presenter

MVC



MVP

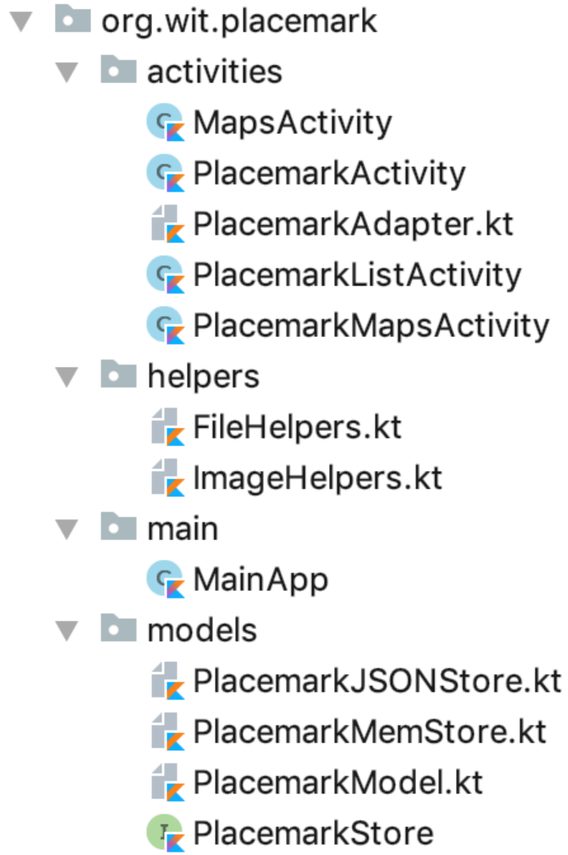


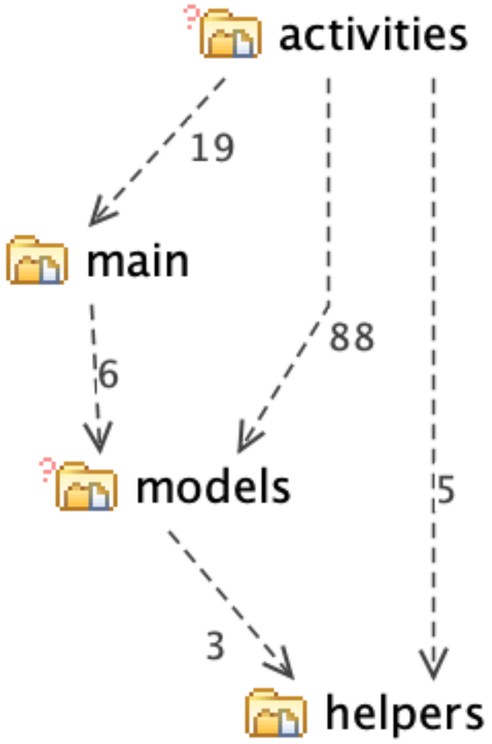
In MVC, the view gets notified of any change in model's state by the model itself.

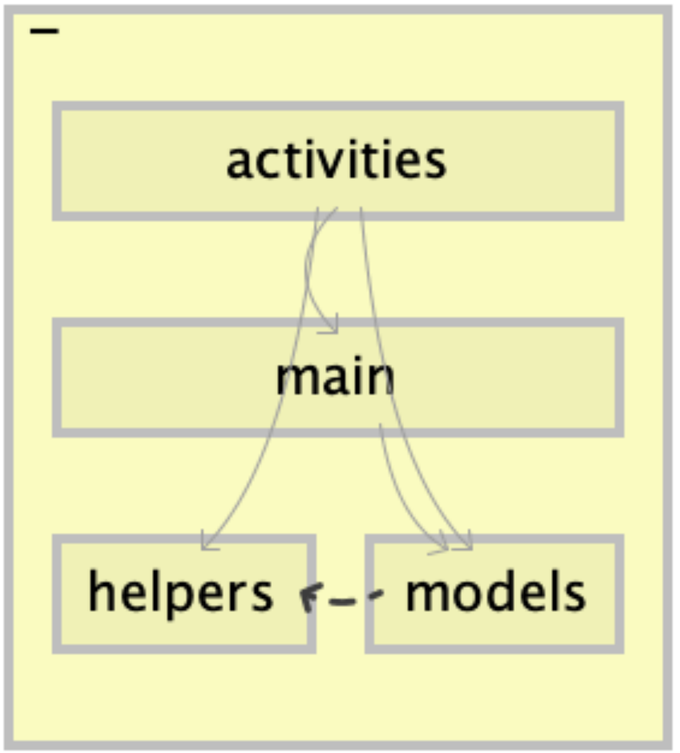
In MVP, the view knows nothing about the model, and it is the presenter's job to fetch the up to date data from the model, understand whether the view should be updated and bind a new data to the view.

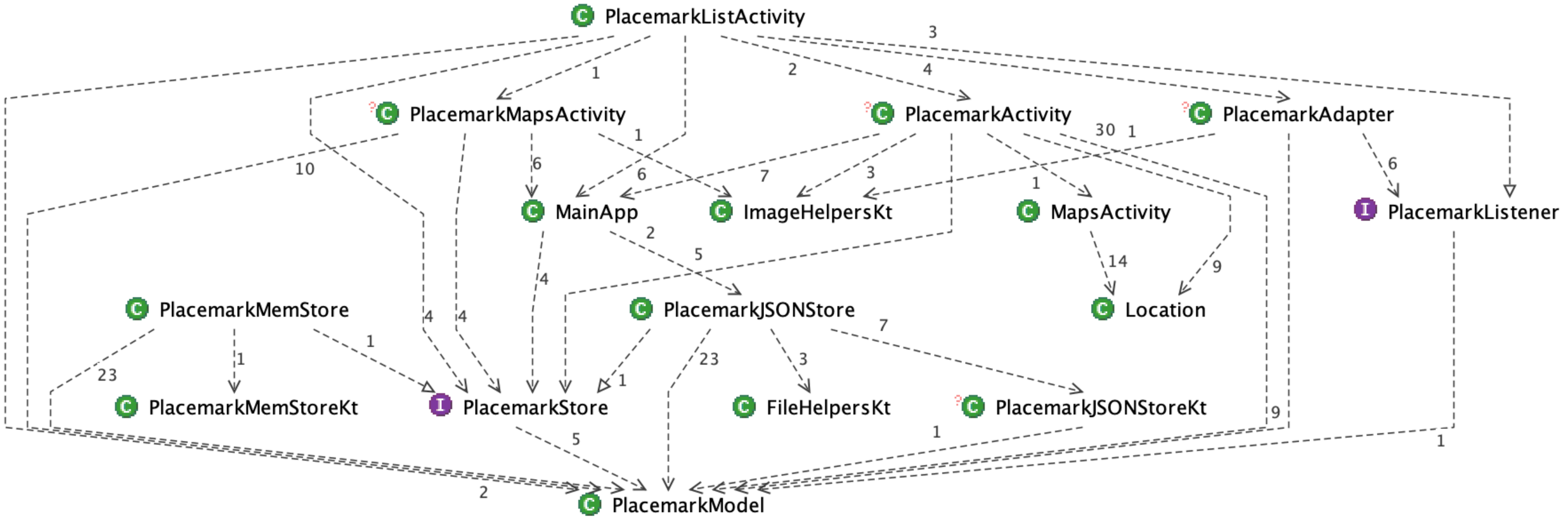
Views in MVC tend to have more logic in them because they are responsible for handling of notifications from the model.

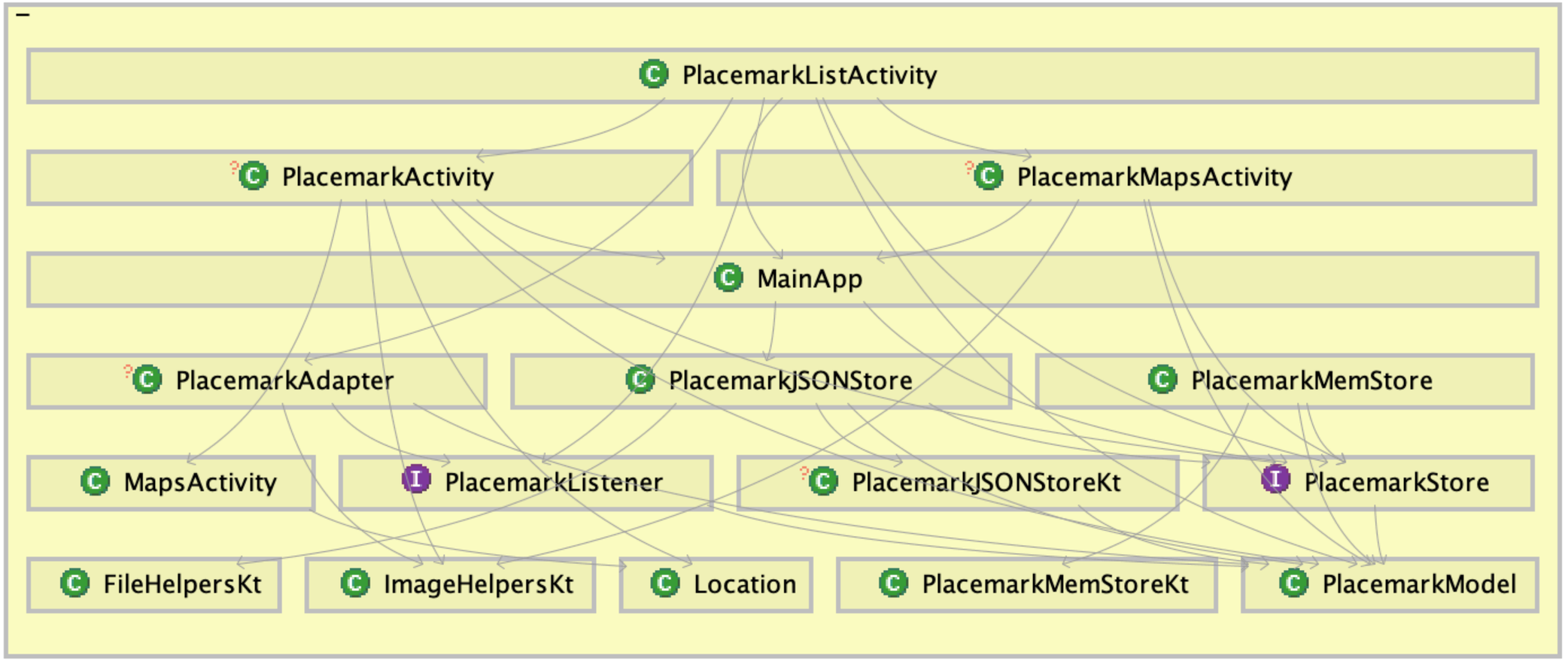
In MVP, the same logic is located in the presenter, which makes the views very "dumb" – their sole purpose becomes rendering of the data that was bound to them by the presenter and capturing user input.


org.wit.placemark
activities
MapsActivity
PlacemarkActivity
PlacemarkAdapter.kt
PlacemarkListActivity
PlacemarkMapsActivity
helpers
FileHelpers.kt
ImageHelpers.kt
main
MainApp
models
PlacemarkJSONStore.kt
PlacemarkMemStore.kt
PlacemarkModel.kt
PlacemarkStore


activities
19
main
6
88
models
5
3
helpers


activities
main
helpers
models


PlacemarkListActivity
PlacemarkMapsActivity
PlacemarkActivity
PlacemarkAdapter
MainApp
ImageHelpersKt
MapsActivity
PlacemarkListener
PlacemarkMemStore
PlacemarkJSONStore
Location
PlacemarkMemStoreKt
PlacemarkStore
FileHelpersKt
PlacemarkJSONStoreKt
PlacemarkModel


PlacemarkListActivity
PlacemarkActivity
PlacemarkMapsActivity
MainApp
PlacemarkAdapter
PlacemarkJSONStore
PlacemarkMemStore
MapsActivity
PlacemarkListener
PlacemarkJSONStoreKt
PlacemarkStore
FileHelpersKt
ImageHelpersKt
Location
PlacemarkMemStoreKt
PlacemarkModel

https://structure101.com/