Паттерны разработки

MVC vs MVP vs MVVM vs MVI

Чего мы хотим добиться?

- Масштабируемость?
- Сопровождаемость?
- Надежность?

- **Разделение ответственности, повторное использование кода, тестируемость!!!**

Если мы хотим изменить что-то, мы не должны ходить по разным участкам кода.

Без разделения ответственности ни **повторное использование кода (Code Reusability)**, ни **тестируемость (Testability)** практически невозможно реализовать.

MVC

Термин ***модель-представление-контроллер (model-view-controller)*** используется с конца 70-х гг. прошлого столетия. Эта модель явилась результатом проекта Smalltalk в компании Xerox PARC, где она была задумана как способ организации некоторых из ранних приложений графического пользовательского интерфейса.

Модели

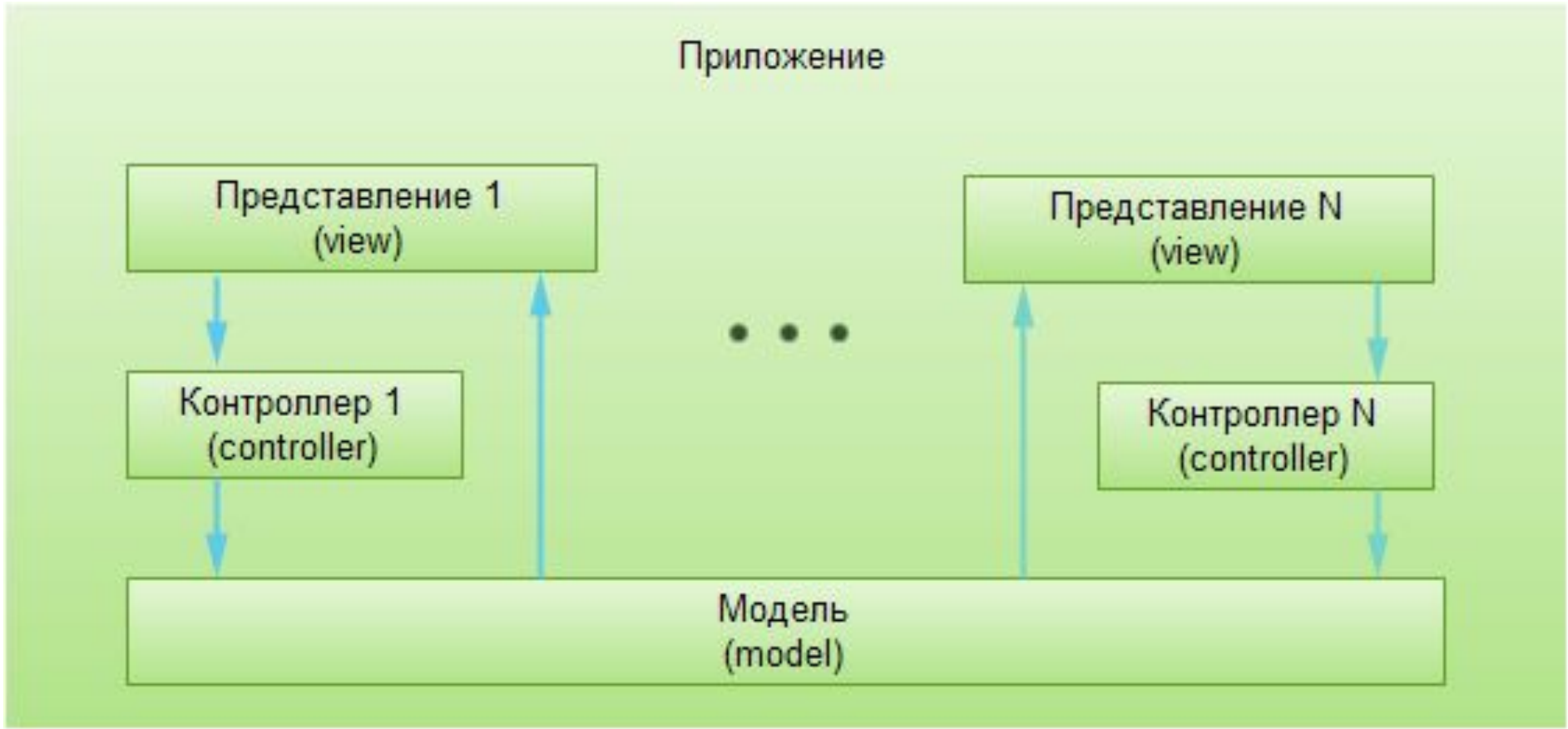
- Содержат или представляют данные, с которыми работают пользователи. Они могут быть простыми моделями представлений, которые только представляют данные, передаваемые между представлениями и контроллерами; или же они могут быть моделями предметной области, которые содержат бизнес-данные, а также операции, преобразования и правила для манипулирования этими данными.

Представления

- Применяются для визуализации некоторой части модели в виде пользовательского интерфейса.

Контроллеры

- Обрабатывают поступающие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю.


Приложение
Представление 1
(view)
Представление N
(view)
Контроллер 1
(controller)
Контроллер N
(controller)
Модель
(model)

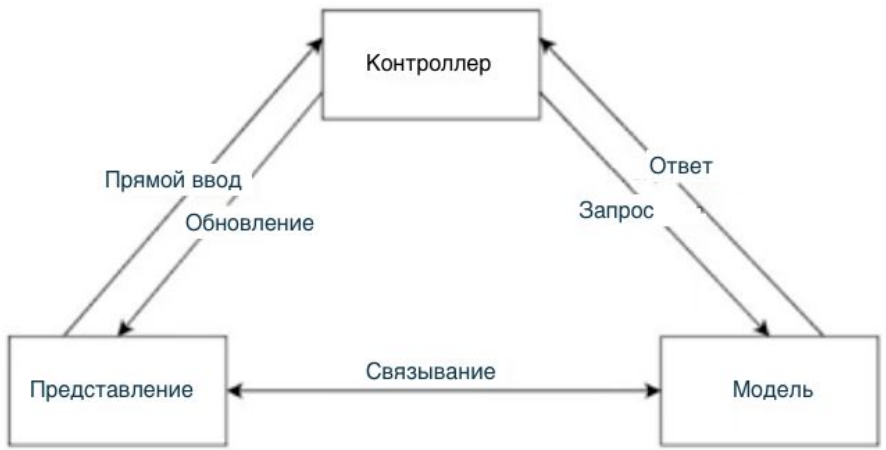
Два варианта MVC: **контроллер-супервизор** и **пассивное представление**

В мобильной экосистеме — практически никогда не встречается реализация контроллера-супервизора.

Архитектуру MVC можно охарактеризовать двумя пунктами:

1. Представление — это **визуальная проекция модели**
2. Контроллер — это **соединение между пользователем и системой**

Supervising Controller



Характеризующий элемент:
Представление связано с моделью

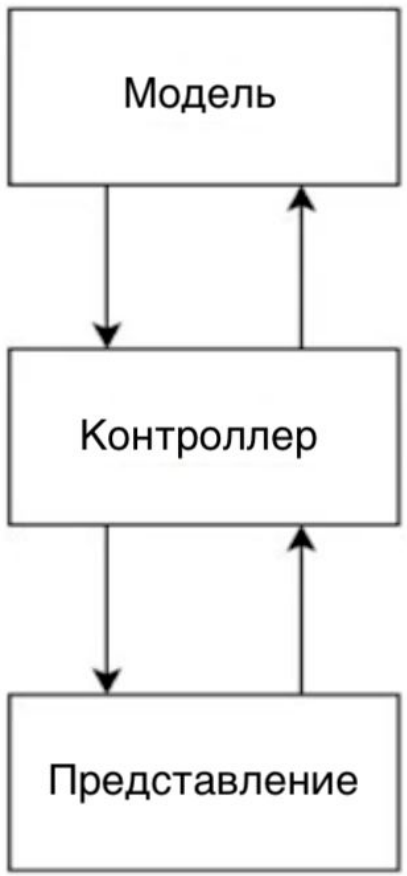
Идея:
Разделение между вводом и выводом

\+ Меньше кода

\+ Сложное unit-тестирование
\+ Сложная инкапсюляция
\+ Слабое разделение ответственности

Мое мнение: хорошо для небольших проектов или демо. Плохо масштабируется

MVC Passive View



Характеризующий элемент:

Представление, полностью управляемое контроллером

Идея:

Разделение между бизнес-логикой,
и логикой отображения

Бизнес-логика - это бизнес правила, которые реализует приложение. Как приложение работает. Реализована в контроллерах.

Логика отображения - как приложение выглядит, какие цвета и текст будут отображены. Реализовано в слое представления

Три уровня

- **Уровень представления**

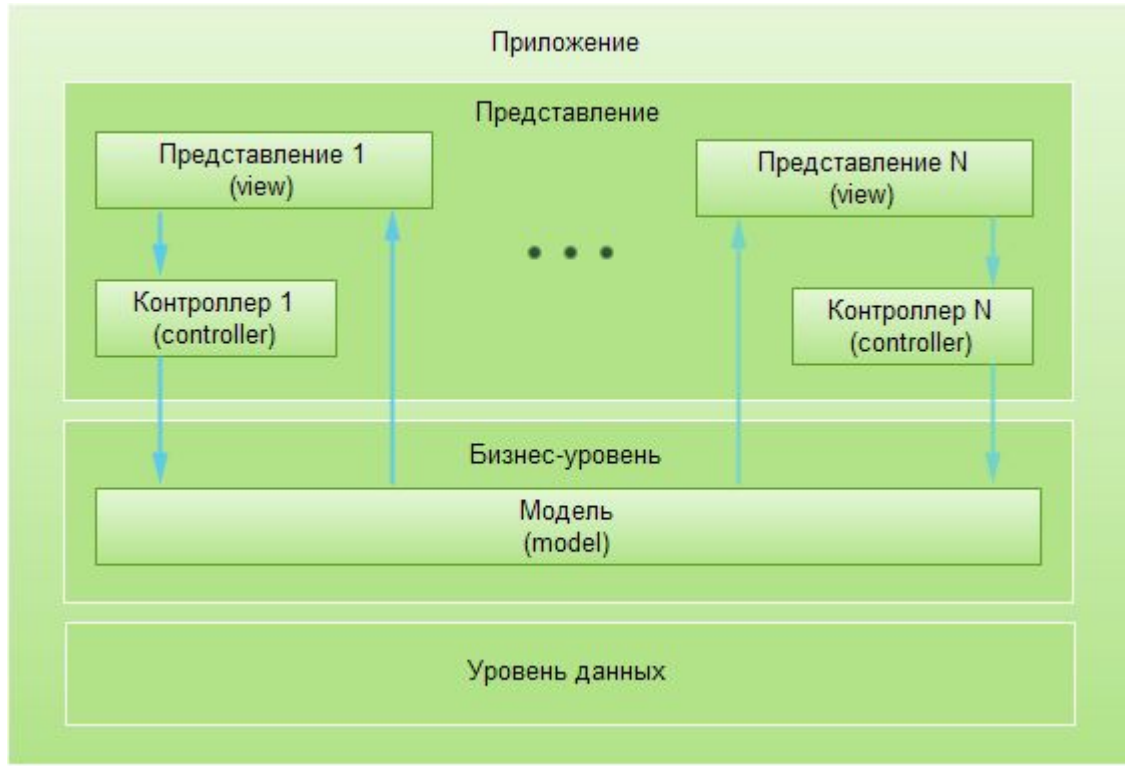
Данный уровень отвечает за отображение данных, обеспечение обратной связи с пользователем, сбор пользовательской информации, которая передается в уровень бизнес-логики для обработки.

- **Бизнес-уровень**

Бизнес-уровень или, если говорить проще, уровень приложения, обеспечивает логику взаимодействия представления и данных. В MVC бизнес-уровень реализует структуру модели.

- **Уровень данных**

Уровень данных отвечает за получение, передачу и сохранение данных в файле, базе данных, службе или XML.


Приложение
Представление
Представление 1
(view)
Представление N
(view)
Контроллер 1
(controller)
Контроллер N
(controller)
Бизнес-уровень
Модель
(model)
Уровень данных

Massive View Controller

Можно разделить представления или определить субпредставления (subviews) с их собственными контроллерами.

Однако, при таком подходе появляются некоторые проблемы:

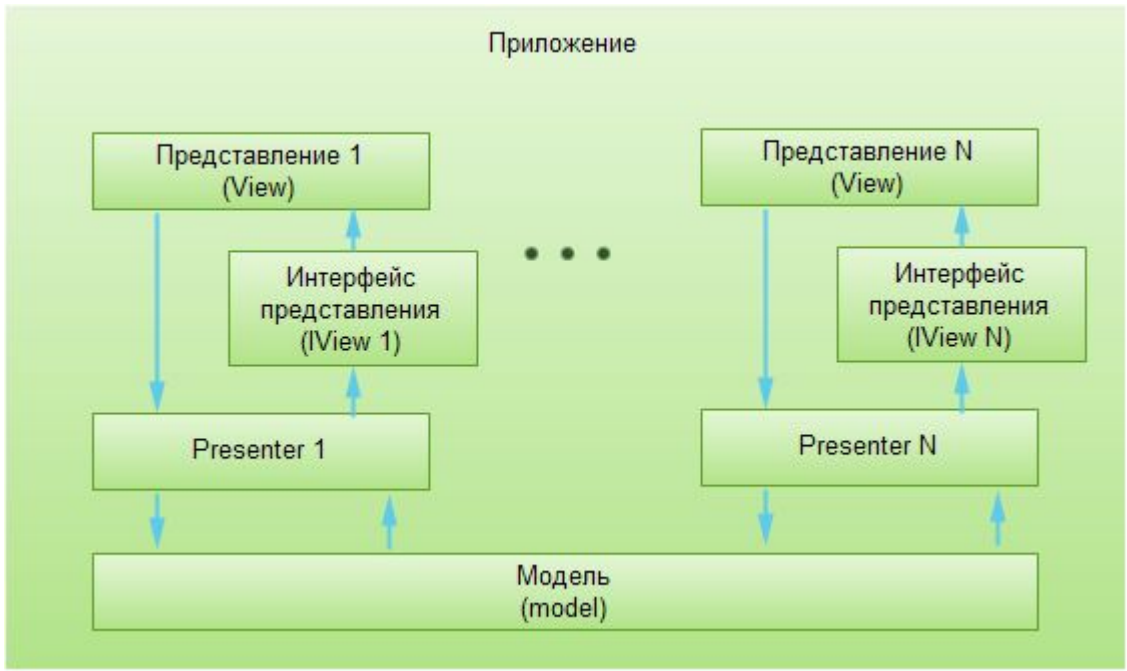
- Объединение логики отображения и бизнес логики
- Трудности при тестировании

Решение этих проблем кроется за созданием абстрактного интерфейса для представления

Все это — и есть главная идея MVP.

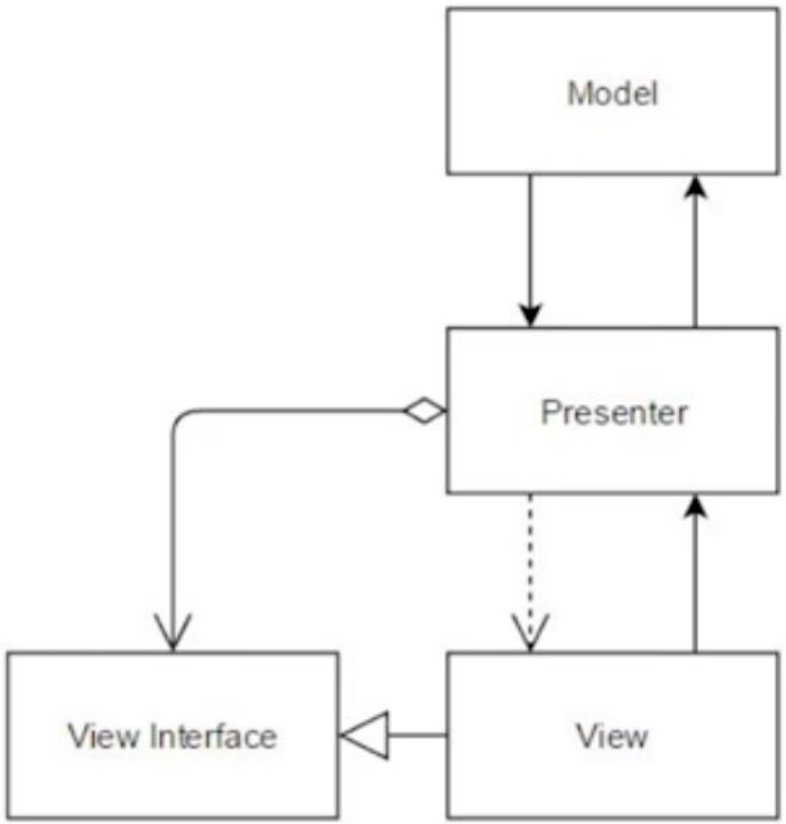
MVP

- ***Model View Presenter (MVP)*** - шаблон, который впервые появился в IBM, а затем использовался в Taligent в 1990-х. MVP является производным от MVC, при этом имеет несколько иной подход. В MVP представление не тесно связано с моделью, как это было в MVC.


Приложение
Представление 1
(View)
Интерфейс
представления
(IView 1)
Представление N
(View)
Интерфейс
представления
(IView N)
Presenter 1
Presenter N
Модель
(model)

- Как видно на этой диаграмме, **Presenter** занял место **контроллера** и отвечает **за перемещение данных**, введенных пользователем, а также **за обновление представления при изменениях**, которые происходят в модели.
- Presenter общается с представлением через интерфейс, который позволяет увеличить тестируемость, так как модель может быть заменена на специальный макет для модульных тестов.

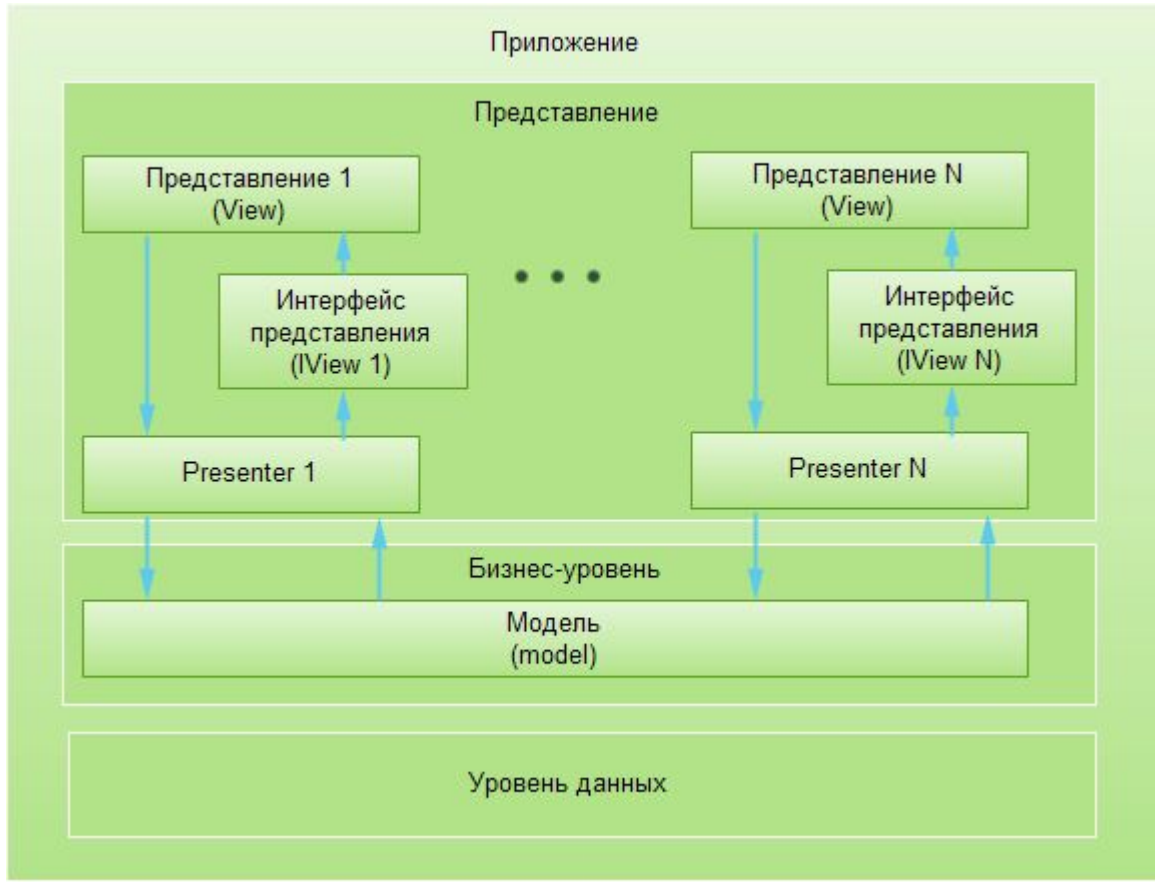
MVP



Характеризующий элемент:
View закрыт за интерфесом

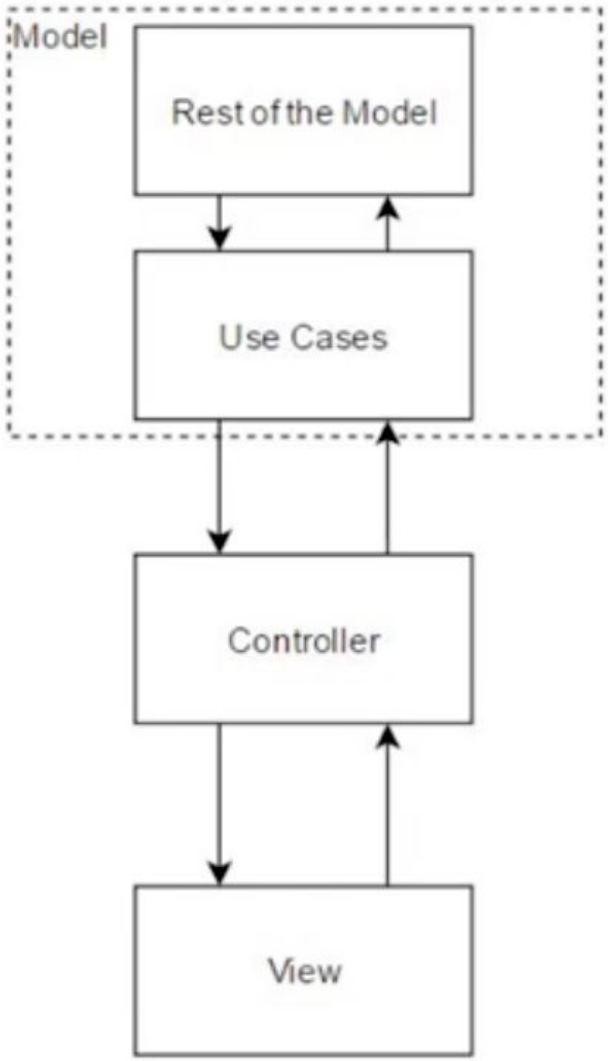
Идея:
Сделать Presenter независимым от View

+ Presenter очень легко тестируется
+ Presenter можно повторно использовать
+ Presenter'ы могу использоваться в общих модулях
- Необходимо создавать и поддерживать интерфейсы для представлений (Views)
- Лишний шаблонный код


Приложение
Представление
Представление 1
(View)
Интерфейс
представления
(IView 1)
Представление N
(View)
Интерфейс
представления
(IView N)
Presenter 1
Presenter N
Бизнес-уровень
Модель
(model)
Уровень данных

- В данном случае структура MVP аналогична MVC в том плане, что код доступа к данным находится вне этой модели, а вся бизнес-логика приложения концентрируется в модели (эта схожесть подтверждает то, что MVP является производным от MVC, изменилась только логика представления).

Use Cases

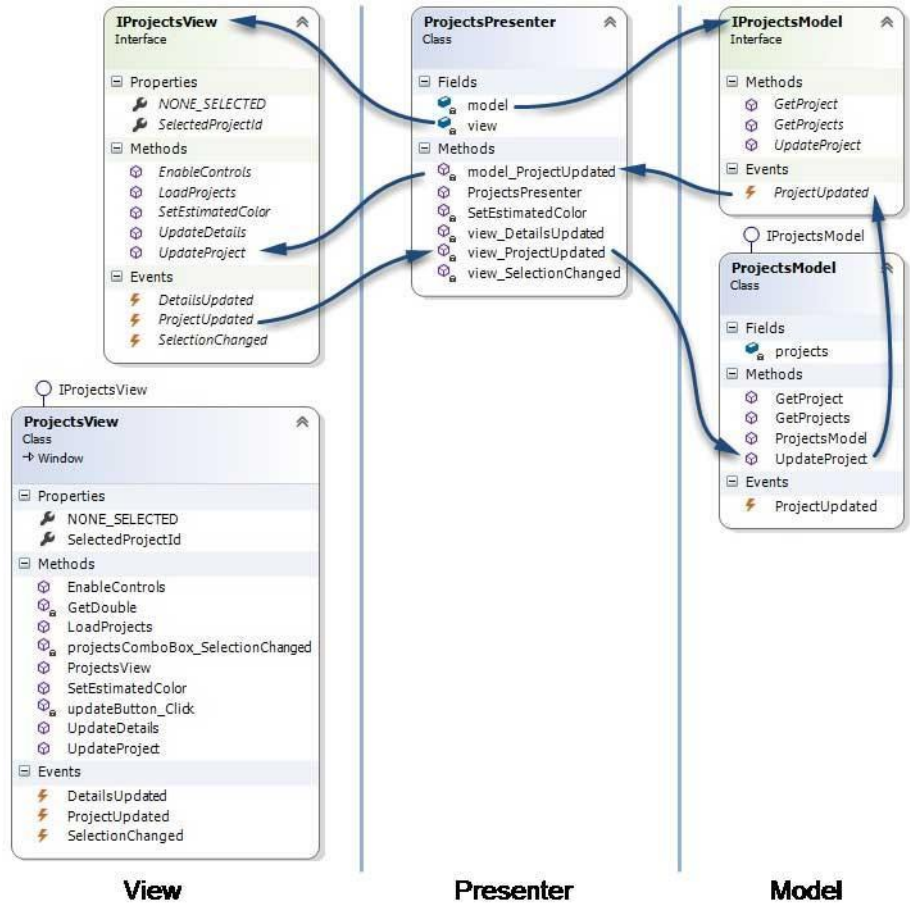


Главное изменение:

Вывод бизнес-логики в отдельные классы, называемые use cases

Идея:

Применение принципа единой ответственности к правилам бизнес-логики


IProjectsView
Interface
Properties
NONE_SELECTED
SelectedProjectId
Methods
EnableControls
LoadProjects
SetEstimatedColor
UpdateDetails
UpdateProject
Events
DetailsUpdated
ProjectUpdated
SelectionChanged
IProjectsView
ProjectsView
Class
Window
Properties
NONE_SELECTED
SelectedProjectId
Methods
EnableControls
GetDouble
LoadProjects
projectsComboBox_SelectionChanged
ProjectsView
SetEstimatedColor
updateButton_Click
UpdateDetails
UpdateProject
Events
DetailsUpdated
ProjectUpdated
SelectionChanged
ProjectsPresenter
Class
Fields
model
view
Methods
model_ProjectUpdated
ProjectsPresenter
SetEstimatedColor
view_DetailsUpdated
view_ProjectUpdated
view_SelectionChanged
IProjectsModel
Interface
Methods
GetProject
GetProjects
UpdateProject
Events
ProjectUpdated
IProjectsModel
ProjectsModel
Class
Fields
projects
Methods
GetProject
GetProjects
ProjectsModel
UpdateProject
Events
ProjectUpdated
View
Presenter
Model

Вот как это работает

- Если пользователь щелкнет на кнопке Update в пользовательском интерфейсе, сработает обработчик события IProjectView.ProjectUpdated, который вызовет метод ProjectsPresenter.view_ProjectUpdated() который обновит модель. При обновлении модели срабатывает обработчик события IProjectsModel.ProjectUpdated, который, через Presenter, обновляет графический интерфейс приложения.
- Подобная модель обладает лучшей тестируемостью нежели модель MVC, т.к. мы перенесли логику построения представления в Presenter
- **В MVC представление строилось непосредственно из модели.**

Итак, MVP представляет собой большой шаг вперед по сравнению с MVC в нескольких направлениях:

- MVP обеспечивает проверяемость состояния и логики представления, перемещая их в Presenter.
- Представление жестко отделяется от модели, при этом связь организуется через Presenter. В отличие от MVC, MVP допускает повторное использование бизнес-логики без непосредственного видоизменения модели (за счет использования интерфейса IView). Например, если вы захотите перенести это WPF-приложение на Silverlight, вам потребуется только создать представление в Silverlight, реализующее IProjectsView, а IProjectsPresenter и IProjectsModel можно использовать повторно.