Столкнувшись с проблемой, следует проверить, не решена ли она уже.

KISS (просто и ясно).

BaseViewModel считывает и хранит коллекцию задач, также некоторые функции, которые используют все остальные VM. Имеет смысл гарантировать, что она в единственном экземпляре. (Singleton).

Publisher-subscriber может пригодиться при обмене информации между VM.

К теме паттернов буду постоянно возвращаться в процессе работы.

**Краткий конспект**

**Паттерн** представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования.

**Порождающие паттерны** — это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов. Среди них выделяются следующие:

* Абстрактная фабрика (Abstract Factory)
* Строитель (Builder)
* Фабричный метод (Factory Method)
* Прототип (Prototype)
* Одиночка (Singleton) - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа. Когда необходимо, чтобы для класса существовал только один экземпляр.

Для отличий:

Статический класс может использоваться как обычный контейнер для наборов методов, работающих на входных параметрах, и не должен возвращать или устанавливать каких-либо внутренних полей экземпляра.

Программа не может точно указать, когда загружается статический класс. Однако гарантируется загрузка этого класса, инициализация его полей и вызов статического конструктора перед первым обращением к классу в программе. Статический конструктор вызывается только один раз, и статический класс остается в памяти на время существования домена приложения, в котором находится программа.

Более привычно объявление нестатического класса с несколькими статическими членами, чем объявление всего класса как статического. Статические поля обычно используются для следующих двух целей: хранение счетчика числа созданных объектов или хранение значения, которое должно совместно использоваться всеми экземплярами.

**Структурные паттерны** - рассматривает, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты. К таким шаблонам относятся:

* Адаптер (Adapter)
* Мост (Bridge)
* Компоновщик (Composite) объединяет группы объектов в древовидную структуру по принципу "часть-целое и позволяет клиенту одинаково работать как с отдельными объектами, так и с группой объектов.
* Когда объекты должны быть реализованы в виде иерархической древовидной структуры
* Когда клиенты единообразно должны управлять как целыми объектами, так и их составными частями. То есть целое и его части должны реализовать один и тот же интерфейс
* Декоратор (Decorator)
* Фасад (Facade)
* Приспособленец (Flyweight)
* Заместитель (Proxy)

**Поведенческими** - они определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение. Среди подобных шаблонов можно выделить следующие:

* Цепочка обязанностей (Chain of responsibility) - поведенческий шаблон проектирования, который позволяет избежать жесткой привязки отправителя запроса к получателю, позволяя нескольким объектам обработать запрос. Все возможные обработчики запроса образуют цепочку, а сам запрос перемещается по этой цепочке, пока один из ее объектов не обработает запрос. Каждый объект при получении запроса выбирает, либо обработать запрос, либо передать выполнение запроса следующему по цепочке.
* Когда имеется более одного объекта, который может обработать определенный запрос
* Когда надо передать запрос на выполнение одному из нескольких объект, точно не определяя, какому именно объекту
* Когда набор объектов задается динамически
* Команда (Command)
* Интерпретатор (Interpreter)
* Итератор (Iterator)
* Посредник (Mediator)
* Хранитель (Memento)
* Наблюдатель (Observer)
* Состояние (State)
* Стратегия (Strategy)
* Шаблонный метод (Template method)
* Посетитель (Visitor)

**Доп информация**

Когда следует использовать абстрактные классы:

* Если надо определить общий функционал для родственных объектов
* Если мы проектируем довольно большую функциональную единицу, которая содержит много базового функционал.
* Если нужно, чтобы все производные классы на всех уровнях наследования имели некоторую общую реализацию. При использовании абстрактных классов, если мы захотим изменить базовый функционал во всех наследниках, то достаточно поменять его в абстрактном базовом классе.

Когда следует использовать интерфейсы:

* Если нам надо определить функционал для группы разрозненных объектов, которые могут быть никак не связаны между собой.
* Если мы проектируем небольшой функциональный тип.