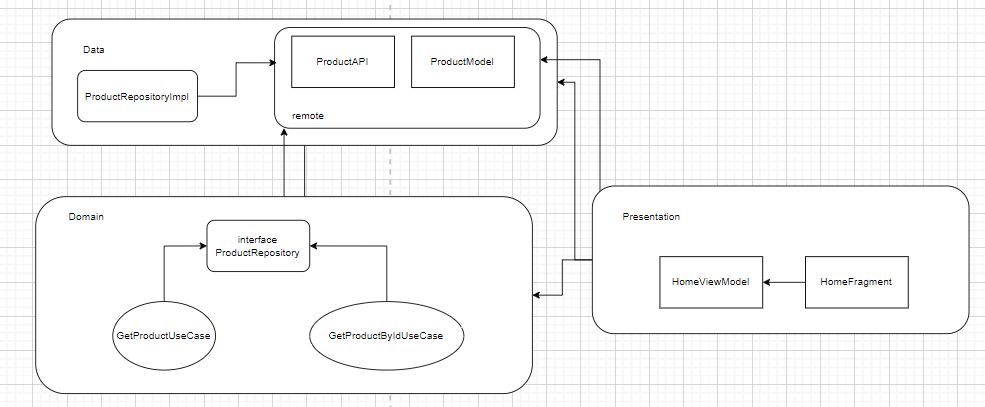
Паттерн Адаптер. Леонов Руслан 932103

Версия без паттерна

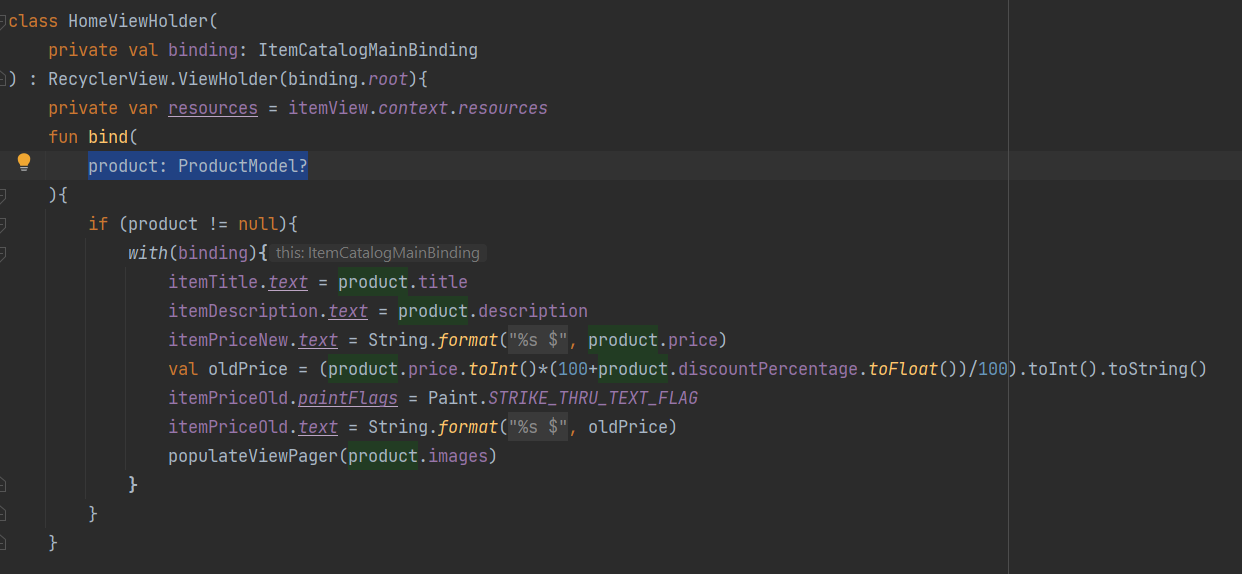


В приложении имеется сущность ProductModel, поставляемая в комплекте с модулем remote для получения данных по сети. Предположим, что этот модуль нам дают извне, мы не можем на него влиять, только использовать «as is». По требованиям проектирования архитектуры, логика приложения строится вокруг использования сущностей (фактически, это entity из терминологии баз данных). Поскольку в данной реализации мы не можем пользоваться адаптерами, встает вопрос: что же нам делать? Есть несколько «костыльных» решений, а так же способ, целиком и полностью нарушающий архитектуру: мы будем использовать служебную сущность ProductModel во всем приложении. Собственно, рассматривать будем последний способ, и выстраивать проблематику «неиспользования» паттерна Адаптер будем именно на нем.

При таком подходе диаграмма архитектуры будет выглядеть примерно так (uml-требования и правила не соблюдены, диаграмма несет исключительно информационный и демонстрационный характер)



Чтобы четко понимать, какие проблемы обнажило использование ProductModel по всему приложению – опишу базовые архитектурные требования подхода Clean Architecture (не вдаваясь в подробности, только минимальный набор, касательно проблематики). Например, модуль remote это служебный модуль слоя Data, доступ к нему, по-правильному, должен иметь только слой Data (например, через какой-либо RepositoryImpl, к которому уже цепляются UseCase, а к ним ViewModels и так далее). То есть объекты модуля remote не покидают слоя Data. Служебная сущность ProductModel – это, как раз, один из объектов remote’а, и мы видим, что в нашем «неправильном» решении все приложение, все слои имеют доступ к этому классу. Из-за этого, кстати, создается ссылочность слоя Domain на слой Data, а этого быть не должно, Domain – самый независимый слой в приложении.



Вот так, например, выглядит наполнение элемента прокручиваемого списка с помощью данных из объекта ProductModel. То есть совершенно нормально, что мы как бы распределяем поля объекта по графическим объектам. А теперь представим, что тот, кто поставляет нам модуль remote, решил поменять ProductModel. Например, он решил, что поля называются как-то некрасиво, и решил их переименовать. У нас при очередной сборке, естественно, все упадет, и придется поднимать документацию по изменениям, и вручную подгонять имена полей под новые. При этом не забыть поменять все использования. В большом проекте такие проблемы превращаются в сущий кошмар.

Версия с Паттерном



С помощью класса ProductConverter мы из ProductModel получаем объект Product (который уже является частью непосредственно нашего приложения и управляется нами). ProductModel не покидает своего слоя. Все приложение работает с Product. Более того в конвертере мы можем, например, задать правила, по которым данные из ProductModel будут переходить в Product. Это нужно, потому что ProductModel – служебная сущность, она нужна для правильной десериализации, логика её построения может быть неудобной для наших задач. Именно поэтому мы самостоятельно пишем Product такой, какой нам надо и заполняем его так, как нам надо.

