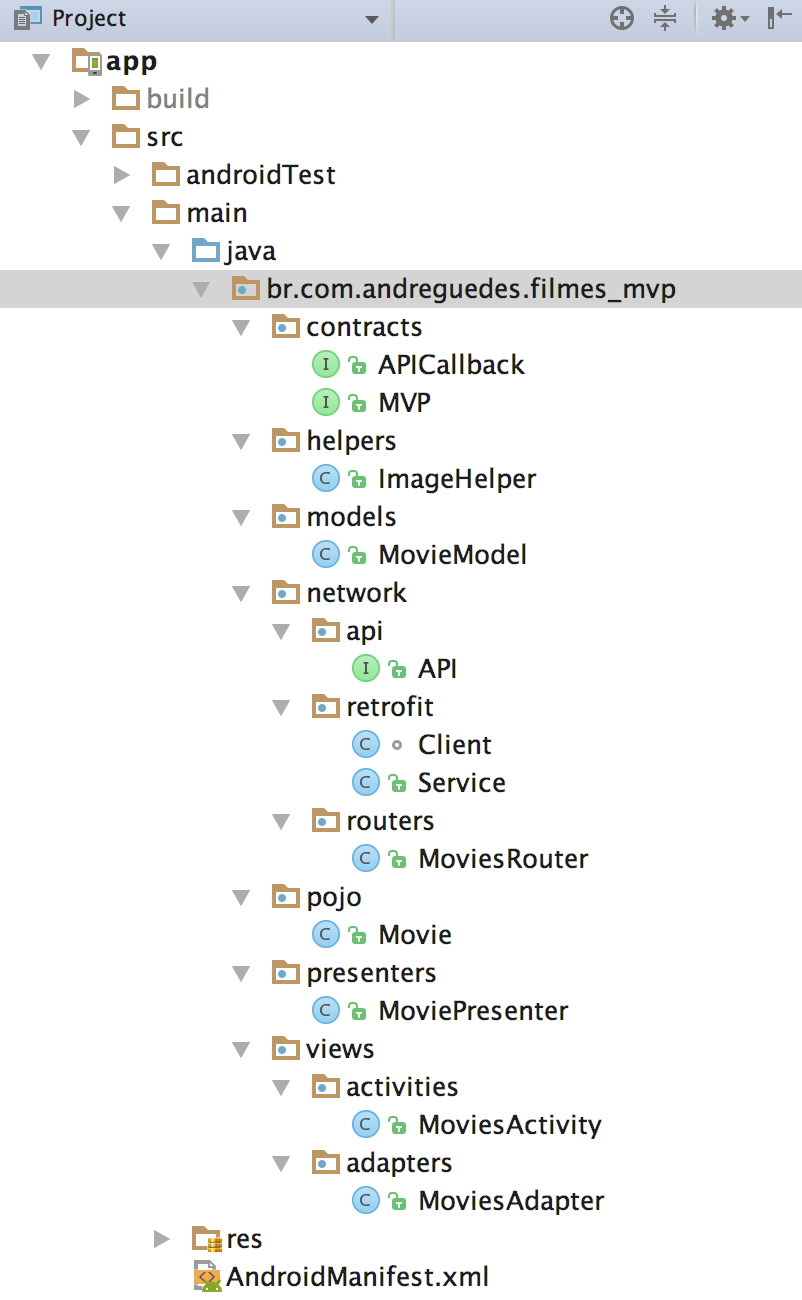
**Separando responsabilidades no Android com MVP**

Estamos familiarizados a trabalhar no Android com Activities e Fragments, e muitas vezes precisamos acessar dados via requisição, ou até mesmo de um banco de dados local, e exibir informações na tela do dispositivo para o usuário. Essa é uma tarefa que acontece quase que diariamente.

Se você sabe do que estou falando, já deve ter criado uma simples lista de algum JSON e exibido no seu RecyclerView, que foi declarado na sua Activity ou Fragment.

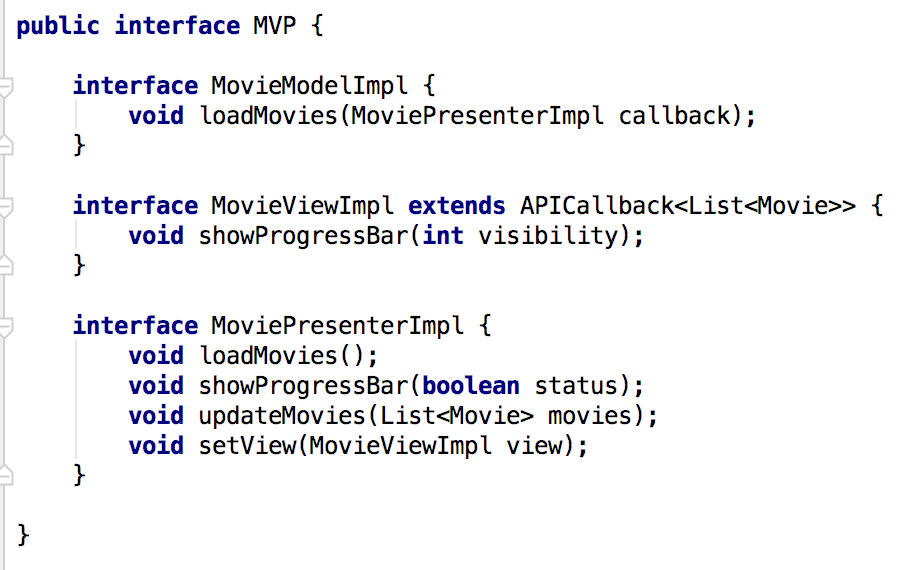
Basicamente, criamos 3 camadas no nosso projeto que servirão para separar **Models**, **Views** e **Presenters**, além de outras existentes para outros fins, inclusive uma para **network** utilizando o Retrofit. Esse vai ser assunto para um post em breve, amigos...

Vale lembrar que existem outros tipos de abordagens sobre o MVP e formas de separação. Para melhor entendimento, resolvi separar as camadas do projeto de exemplo da maneira que achei mais fácil de entender. =)



Na imagem acima podemos ver a estrutura do projeto dentro do Android Studio.

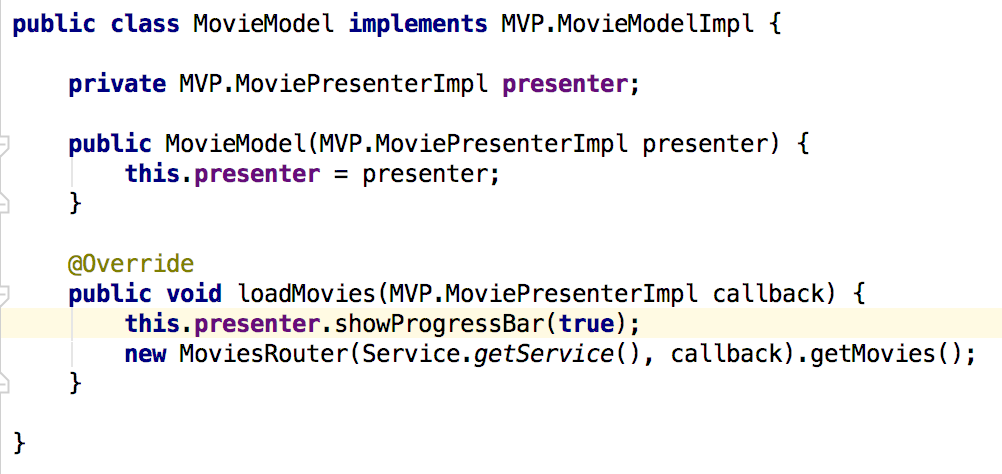
Tudo começa por uma interface principal que deverá conter métodos de implementações para as views, models e presenters, a interface **API**. No nosso exemplo, a listagem de filmes será resgatada de um JSON e iremos atualizar a lista de filmes assim que o retorno do mesmo for realizado com sucesso e também, deve-se ocultar a barrinha de progresso para o usuário. Essas ações devem ser resumidas em métodos, que serão de responsabilidade de cada camada a sua devida implementação. Veja como nossa interface ficou.



Podemos ver que alguns métodos tem assinaturas baseadas nas ações que tomaríamos caso os mesmos fossem criados em Activities, por exemplo, a atualização da lista de filmes na implementação dos presenters.

Então, bora conferir como ficaram os códigos!!!

Na camada de **models** criamos uma classe que implementa *MovieModelImpl*, e que recebe a implementação de *MoviePresenterImpl.* Nessa camada podemos fazer acesso a dados do banco, serviços da web (método loadMovies), etc. Nesse cenário visualizamos que a camada de Model apenas se comunica com a camada Presenter. A imagem abaixo mostra a implementação final da classe MovieModel.



Já a camada de **presenters**, é a representação do meio-campo entre as **views** e **models,** realizando a separação necessária e implementando todas as operações pertinentes ao resgate de informações na model e retorno das mesmas para a view.



Dentro da camada de **views** incluímos todas as nossas Activities, Fragments e Adapters. E será dentro da Activity que vamos implementar os métodos para atualizar a view. Nossa Activity deve implementar *MovieViewImpl* e ter uma referência a camada de presenter, por meio de uma interface.



Note que os códigos referentes a listagem se resumem ao método onSuccess(), que retorna a lista de filmes e solicita a atualização dos mesmos utilizando o adapter. A instância de um novo presenter se dá no método onResume(), onde a view é injetada e logo após é feito a requisição dos filmes para o serviço.

Para quem já desenvolveu um projeto simples em Android com a mesma finalidade, conseguirá visualizar o tamanho do impacto positivo que causa esse tipo de arquitetura.

Até a próxima!!!