報道各位

2018 年 12 月 28 日
株式会社インプレスR&D
https://nextpublishing.jp/

レガシーなコードを、MVP で分割してテスト可能に変える！

『テストが書けない人の Android MVP』発行

技術書典シリーズ、12月の新刊

インプレスグループで電子出版事業を手がける株式会社インプレスR&Dは、『テストが書けない人のAndroid MVP』（著者：高畑 匡秀）を発行いたしました。

『テストが書けない人のAndroid MVP』
https://nextpublishing.jp/isbn/9784844398714

著者：高畑 匡秀
小売希望価格：電子書籍版　1400 円（税別）／印刷書籍版　1600 円（税別）
電子書籍版フォーマット：EPUB3／Kindle Format8
印刷書籍版仕様：B5 判／カラー／本文 60 ページ
ISBN：978-4-8443-9871-4
発行：インプレス R&D

＜＜発行主旨・内容紹介＞＞

【レガシーなコードを MVP で分割してテスト可能に変える！】

本書は多くのプロジェクトで運用されているレガシーなコードをリファクタリングし、将来的にモダンなコードとしていくために必要なテストコードの書き方を解説したガイドブックです。

将来的に Dagger2 に置き換えることを目標としつつ、いくつかのレガシーパターンのリファクタリング例を紹介しています。

〈本書の想定読者〉

・レガシー化した Android のソースコードを今どきの MVP に置き換えたいが何から手を付ければいいかわからない

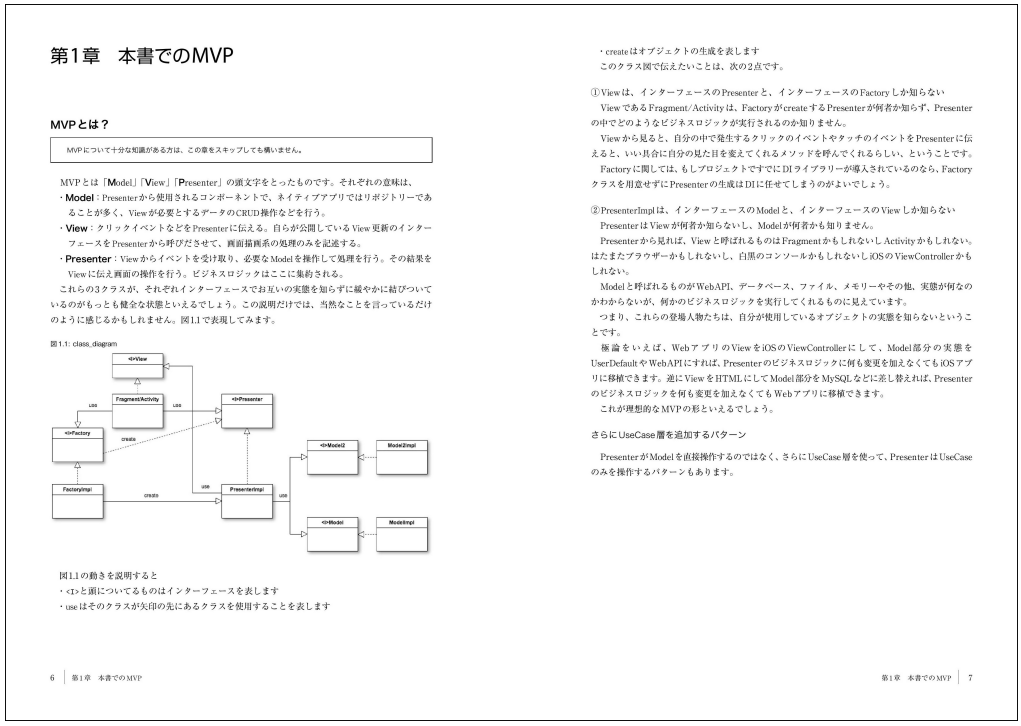
・MVP にしてみたけど Presenter に View 側の処理が入り込んだり、何が変わったかわからない

・MVP にしてみたけど結局テストコードが書けない

・Dagger2 や RxJava を使わないとテストは書けないと思っている

（本書は、次世代出版メソッド「NextPublishing」を使用し、出版されています。）

MVP の基本的な知識から丁寧に解説



第1章　本書でのMVP

MVPとは？

MVPについて十分な知識がある方は、この章をスキップしても構いません。

MVPとは「Model」「View」「Presenter」の頭文字をとったものです。それぞれの意味は、

・**Model**：Presenterから使用されるコンポーネントで、ネイティブアプリではリポジトリーであることが多く、Viewが必要とするデータのCRUD操作などを行う。

・**View**：クリックイベントなどをPresenterに伝える。自らが公開している View 更新のインターフェースをPresenterから呼びださせて、画面描画系の処理のみを記述する。

・**Presenter**：Viewからイベントを受け取り、必要なModelを操作して処理を行う。その結果をViewに伝え画面の操作を行う。ビジネスロジックはここに集約される。

これらの3クラスが、それぞれインターフェースでお互いの実態を知らずに緩やかに結びついているのがもっとも健全な状態といえるでしょう。この説明だけでは、当然なことを言っているだけのように感じるかもしれません。図1.1で表現してみます。

図1.1: class_diagram

図1.1の動きを説明すると

・<I>と頭についているものはインターフェースを表します

・useはそのクラスが矢印の先にあるクラスを使用することを表します

・createはオブジェクトの生成を表します

このクラス図で伝えたいことは、次の2点です。

①Viewは、インターフェースのPresenterと、インターフェースのFactoryしか知らない

ViewであるFragment/Activityは、Factoryがcreateする Presenterが何者か知らず、Presenterの中でどのようなビジネスロジックが実行されるのか知りません。

Viewから見ると、自分の中で発生するクリックのイベントやタッチのイベントをPresenterに伝えると、いい具合に自分の見た目を変えてくれるメソッドを呼んでくれるらしい、ということです。

Factoryに関しては、もしプロジェクトですでにDIライブラリーが導入されているのなら、Factoryクラスを用意せずにPresenterの生成はDIに任せてしまうのがよいでしょう。

②PresenterImplは、インターフェースのModelと、インターフェースのViewしか知らない

PresenterはViewが何者か知らないし、Modelが何者かも知りません。

Presenterから見れば、Viewと呼ばれるものはFragmentかもしれないしActivityかもしれないし、はたまたブラウザーかもしれないし、白黒のコンソールかもしれないしiOSのViewControllerかもしれない。

Modelと呼ばれるものがWebAPI、データベース、ファイル、メモリーやその他、実態が何なのかわからないが、何かのビジネスロジックを実行してくれるものに見えています。

つまり、これらの登場人物たちは、自分が使用しているオブジェクトの実態を知らないということです。

極論をいえば、WebアプリのViewをiOSのViewControllerにして、Model部分の実態をUserDefaultやWebAPIにすれば、Presenterのビジネスロジックに何も変更を加えなくてもiOSアプリに移植できます。逆にViewをHTMLにしてModel部分をMySQLなどに差し替えれば、Presenterのビジネスロジックを何も変更を加えなくてもWebアプリに移植できます。

これが理想的なMVPの形といえるでしょう。

さらにUseCase層を追加するパターン

PresenterがModelを直接操作するのではなく、さらにUseCase層を使って、PresenterはUseCaseのみを操作するパターンもあります。

MVP 化する際の心得を 4 つにまとめて紹介

第2章　MVP化の心得

この章では、MVP化にあたって筆者が考えるポイントを3つ紹介します。

心得1：ViewとPresenterのインターフェースを「声に出して」抽出する

MVP化するときに、よく問題になるのは何をView側の処理として、何をPresenter側の処理とするのかということです。このインターフェースの抽出が、メンバーによって差が出てしまうことがあります。チーム内で意識を合わせておかないと、Presenterの中にView側で処理するべきことが混じったり、逆にPresenter側で処理することがView側に混ざることが起きます。

筆者がお勧めするインターフェースの抽出方法は、**一度そのViewの処理を声にだして整理してみる**ことです。

例えば、多少の違いはあれど、多くの場合は、

「ユーザーが○○したら（されたら）、△△して、□□**を表示する**」

という流れになります。そして、○○や△△、□□のところに入るのは、

・○○：Viewが呼ぶであろうPresenterのインターフェース

・△△：Presenterの中で行うビジネスロジック

・□□を表示する：Presenterが呼ぶであろうViewのインターフェース

になります。

例として、「ユーザーが**新規登録ボタンをクリック**したら、**サーバーからユーザー情報を取得**して、**結果が成功なら完了メッセージを表示する**」という処理を考えてみましょう。声に出して読んでみましたか？

この場合、

・○○：新規登録ボタンをクリックしたら（PresenterのclickRegisteredButton）

・△△：サーバーからユーザー情報を取得（Presenterのビジネスロジック）

・□□を表示する：結果が成功なら完了のメッセージを表示する（ViewのshowCompletedMessage）

となります。つまりContractのインターフェースは次のようになるでしょう。

リスト2.1: Contractのインターフェース例①

```
1: interface Contract {
2:
3:     interface Presenter{
4:         fun clickRegisteredButton()
5:     }
6:
7:     interface View {
8:         fun showCompletedMessage()
9:     }
10:
11: }
```

もうひとつ、やってみましょう。

「画面が表示されたら、**DBから自分の過去のお気に入りTweetを取得**して、**リストを表示する**」です。読んでみましょう。

この場合、

・○○：画面が表示されたら（PresenterのonShowed）

・△△：DBから自分の過去のお気に入りTweetを取得（Presenterのビジネスロジック）

・□□を表示する：リスト表示する（ViewのshowTweetList）

この場合のContractのインターフェースは次のようになるでしょう。

リスト2.2: Contractのインターフェース例②

```
1: interface Contract {
2:
3:     interface Presenter{
4:         fun onShowed()
5:     }
6:
7:     interface View {
8:         fun showTweetList(tweetList:List<Tweet>)
9:     }
10:
11: }
```

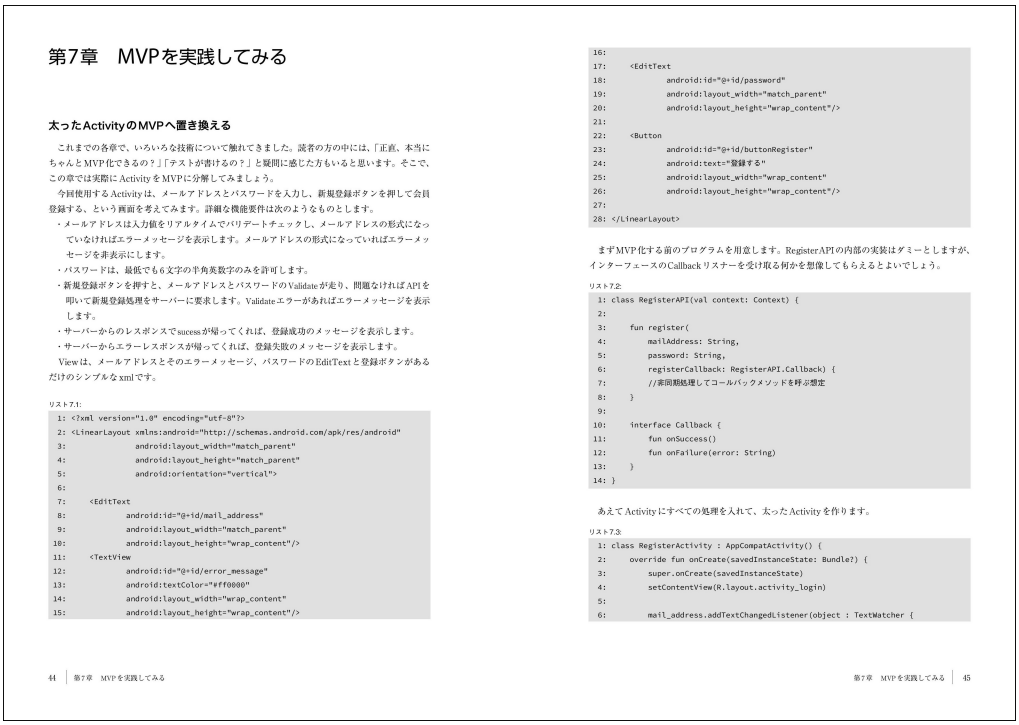
心得2：可能な限りViewにifを書かない

プログラムはif、for、switchなどの分岐の多さで複雑度が変わります。

MVP化のメリットは、Viewから余計な条件分岐をなくすことでもあります。すべての条件分岐をPresenterの中に押し込め、そのすべての条件分岐をUnitTestで網羅することによってビジネスロジックの正当性を証明できます。

そしてPresenterをAndroid特有の関心事から切り離されていれば、常にUnitTestで回すことが

実際のコードの MVP 化についてサンプルを題材に学習



第7章　MVPを実践してみる

太ったActivityのMVPへ置き換える

これまでの各章で、いろいろな技術について触れてきました。読者の方の中には、「正直、本当にちゃんとMVP化できるの？」「テストが書けるの？」と疑問に感じた方もいると思います。そこで、この章では実際にActivityをMVPに分解してみましょう。

今回使用するActivityは、メールアドレスとパスワードを入力し、新規登録ボタンを押して会員登録する、という画面を考えてみます。詳細な機能要件は次のようなものとします。

・メールアドレスは入力値をリアルタイムでバリデートチェックし、メールアドレスの形式になっていなければエラーメッセージを表示します。メールアドレスの形式になっていればエラーメッセージを非表示にします。
・パスワードは、最低でも6文字の半角英数字のみを許可します。
・新規登録ボタンを押すと、メールアドレスとパスワードのValidateが走り、問題なければAPIを叩いて新規登録処理をサーバーに要求します。Validateエラーがあればエラーメッセージを表示します。
・サーバーからのレスポンスでsucessが帰ってくれば、登録成功のメッセージを表示します。
・サーバーからエラーレスポンスが帰ってくれば、登録失敗のメッセージを表示します。

Viewは、メールアドレスとそのエラーメッセージ、パスワードのEditTextと登録ボタンがあるだけのシンプルなxmlです。

リスト7.1:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical">

    <EditText
            android:id="@+id/mail_address"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
    <TextView
            android:id="@+id/error_message"
            android:textColor="#ff0000"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

    <EditText
            android:id="@+id/password"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>

    <Button
            android:id="@+id/buttonRegister"
            android:text="登録する"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

</LinearLayout>
```

まずMVP化する前のプログラムを用意します。RegisterAPIの内部の実装はダミーとしますが、インターフェースのCallbackリスナーを受け取る何かを想像してもらえるとよいでしょう。

リスト7.2:

```
class RegisterAPI(val context: Context) {

    fun register(
        mailAddress: String,
        password: String,
        registerCallback: RegisterAPI.Callback) {
        //非同期処理してコールバックメソッドを呼ぶ想定
    }

    interface Callback {
        fun onSuccess()
        fun onFailure(error: String)
    }
}
```

あえてActivityにすべての処理を入れて、太ったActivityを作ります。

リスト7.3:

```
class RegisterActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)

        mail_address.addTextChangedListener(object : TextWatcher {
```

44 | 第7章　MVPを実践してみる

第7章　MVPを実践してみる | 45

＜＜目次＞＞

第5章 コールバックをテスト
・インターフェースコールバックをテストする
・Timer 処理もテストする
・余裕があればリポジトリーパターンに置き換え
第6章 外部ライブラリー依存
・サードパーティのライブラリーをそのまま使ってはいけない
・ラップクラスで包むまたはリポジトリーパターンに置き換え
・コンストラクタインジェクションする
第7章 MVP を実践してみる
・太った Activity の MVP へ置き換える
・次のステップへ

＜＜著者紹介＞＞

高畑 匡秀(たかはた まさひで)
株式会社 Newspicks で Android アプリを中心にサーバーサイドの開発に携わる。テストコードをどうやったら書けるようになるのか考えるのが好き。

＜＜販売ストア＞＞

電子書籍:
Amazon Kindle ストア、楽天 kobo イーブックストア、Apple Books、紀伊國屋書店 Kinoppy、Google Play Store、honto 電子書籍ストア、Sony Reader Store、BookLive!、BOOK☆WALKER
印刷書籍:
Amazon.co.jp、三省堂書店オンデマンド、honto ネットストア、楽天ブックス
※ 各ストアでの販売は準備が整いしだい開始されます。
※ 全国の一般書店からもご注文いただけます。

【株式会社インプレス R&D】 https://nextpublishing.jp/
株式会社インプレス R&D（本社：東京都千代田区、代表取締役社長：井芹昌信）は、デジタルファーストの次世代型電子出版プラットフォーム「NextPublishing」を運営する企業です。また自らも、NextPublishing を使った「インターネット白書」の出版など IT 関連メディア事業を展開しています。
※NextPublishing は、インプレス R&D が開発した電子出版プラットフォーム(またはメソッド)の名称です。電子書籍と印刷書籍の同時制作、プリント・オンデマンド(POD)による品切れ解消などの伝統的出版の課題を解決しています。これにより、伝統的出版では経済的に困難な多品種少部数の出版を可能にし、優秀な個人や組織が持つ多様な知の流通を目指しています。

【インプレスグループ】 https://www.impressholdings.com/
株式会社インプレスホールディングス(本社:東京都千代田区、代表取締役:唐島夏生、証券コード:東証1部9479)を持株会社とするメディアグループ。「IT」「音楽」「デザイン」「山岳・自然」「旅・鉄道」「学術・理工学」を主要テーマに専門性の高いメデイア&サービスおよびソリューション事業を展開しています。さらに、コンテンツビジネスのプラットフォーム開発・運営も手がけています。

【お問い合わせ先】

株式会社インプレス R&D NextPublishing センター
TEL 03-6837-4820
電子メール: np-info@impress.co.jp