# Learn Android MVP Pattern By Example

May 9, 2016 / Design Pattern / By valokafor / 7 COMMENTS

4 Shares


In this tutorial, I will share a practical example of Model View Presenter (MVP) pattern in Android development. MVP is a design pattern and when it comes to Android development some of the examples available online are heavy on theories. And in theory Android MVP pattern is simple, it advocates separating business and persistence logic out of the Activity and Fragment. That is a noble goal, the only challenge is that in Android development we do not do MainActivity = new Activity().

The consideration of, and the handling of Activity/Fragment life cycles and other unique Android way of life that we have to deal with makes it hard to adopt generic design patterns into Android development. However with MVP, it has been my experience that even a small attempt in adopting Android MVP pattern provides more flexibility and ease of maintainability over non-pattern based Android development.

This tutorial is a continuation of my tutorial on Dependency Injection with Dagger 2. If you did not read that tutorial, I encourage you to do so as that will make it easy for you to understand the rest of this tutorial.

## Android Model View Presenter Pattern Defined

To understand Android MVP pattern we first need to understand the problem it is trying to solve. The goal of MVP in Android is to separate the known from the unknown. Let me explain this with the classic Hello World and the tried and true TextView. If I have an

app that all it needs to do is to display Hello World, then I can accomplish this app like this.

```
1  public class MainActivity extends AppCom
2
3      private TextView mTextViewGreeting;
4
5      @Override
6      protected void onCreate(Bundle saved
7          super.onCreate(savedInstanceStat
8          setContentView(R.layout.activity
9          Toolbar toolbar = (Toolbar) find
10         setSupportActionBar(toolbar);
11
12         mTextViewGreeting = (TextView) f
13         mTextViewGreeting.setText("Hello
14
15     }
16
17 }
```

In this scenario, any mention of design pattern or testing will be over engineering. Because TextView is a known quantity, we do not need to test it, it works given the correct input. However if business requirements change, and I am now asked to add some personalization to my greetings, then I can further update my Hello World to greet like this:

```
1  public class MainActivity extends AppCom
2
3      private TextView mTextViewGreeting;
4
5      @Override
6      protected void onCreate(Bundle saved
7          super.onCreate(savedInstanceStat
8          setContentView(R.layout.activity
9          Toolbar toolbar = (Toolbar) find
10         setSupportActionBar(toolbar);
11
12         mTextViewGreeting = (TextView) f
13         //mTextViewGreeting.setText("Hel
14         mTextViewGreeting.setText(getSal
15
16     }
17
18     private String getSalute() {
19         int hour = Calendar.getInstance(
20
21         String greeting = "Hi";
22         if (hour > 0 && hour <= 12)
23             greeting = "Good Morning";
24         else
25         if (hour > 12 && hour <= 18)
26             greeting = "Good Afternoon";
27         else
28         if (hour > 18 && hour <= 21)
29             greeting = "Good Evening";
30         else
31         if (hour > 21 && hour <= 24)
32             greeting = "Good Night";
33
34         return greeting;
```