
# تکلیف ارائه کتبی

## داده‌های هوشیار در سیستم اندروید - Live Data in Android

نام استاد: استاد پیمان کبیری

دانشجو: محمد امین قسوری

شماره دانشجویی: ۹۷۵۲۱۴۳۲

خرداد ۱۴۰۰

# به نام خدا

سپاس،
از استاد حائری برای کمک‌هایشان در راستای یادگیری برنامه‌نویسی برای سیستم‌عامل اندروید و همچنین دکتر کبیری برای راهنمایی‌هایشان در گزارش نویسی، نهایت سپاس و تشکر را دارم.

# فهرست مطالب

# فهرست نگاره‌ها

# چکیده

بحث داده‌های هوشیار[1] در اندروید، بحثی بسیار داغی است که با استفاده از آن‌ها در معماری MVVM[2] می‌توانیم به ساختار برنامه‌ی خود کمک کنیم. یکی از امتیازهای استفاده از این داده‌های هوشیار در مقابل کلاس‌های درون‌ساخته‌شده‌ی[3] جاوا مانند کلاس Observable این است که داده‌های هوشیار با چرخه‌ی حیات اجزای اندروید[4] بسیار همروند[5] شده است. این موضوع باعث می شود تا از واقعه‌ی نشت حافظه[6] که بحث مهمی در طراحی نرم‌افزارهای گوشی است جلوگیری شود. داده‌های هوشیار، ایده‌ای شبیه به الگوی طراحی[7] مشاهده‌گر[8] را پیاده سازی کرده است و شامل کلاس‌هایی مانند LiveData، MutableLiveData، MediatorLiveData و ... است. همچنین با استفاده از این ابزار، توسعه‌دهنده برای به‌روزرسانی وضعیت برنامه خود دردسر کمتری خواهد داشت و دیگر لازم ندارد که به صورت دستی هر تغییری را به‌روزرسانی کند. از دیگر امتیازهای داده‌های هوشیار، بهبود هم سرعت شدن توسعه نرم‌افزار با ایده‌های تیم محصول خواهد بود و همچنین مشکل‌یابی و بهبود مشکل را برای ما راحت‌تر از قبل خواهد کرد.

[1] Live Data
[2] Model View View-Model
[3] Built-in
[4] Android life cycles
[5] Integrated
[6] Memory lick
[7] Design pattern
[8] Observer

# ۱ مقدمه

در طی چندین سال اخیر، به توصیه‌ی شرکت گوگل، توسعه‌دهنده‌های اندروید به استفاده از معماری جدیدی برای پیاده سازی برنامه های خود پرداخته‌اند. خود چارچوب[۹] اندروید شباهت زیادی به MVC[10] دارد. معماری که در چندین سال اخیر به توصیه مستندات گوگل خیلی مورد توجه قرار گرفته است، معماری MVVM است. در کل توسعه‌دهنده‌های اندروید بیشتر درگیر چگونگی معماری و طراحی خود در سیستم‌های نرم‌افزاری شده اند. با استفاده از ابزارها و کتابخانه‌هایی که در چارچوب اندروید وجود دارند می‌توان نرم‌افزارهای با کیفیت‌تری را تولید نمود. داده‌های هوشیار، یکی از مواردی است که می‌توانیم در جهت بهبود برنامه‌ی خود استفاده کنیم. همچنین داده‌های هوشیار با معماری MVVM و کتابخانه View Model اندروید هم‌روند شده است.[۱]

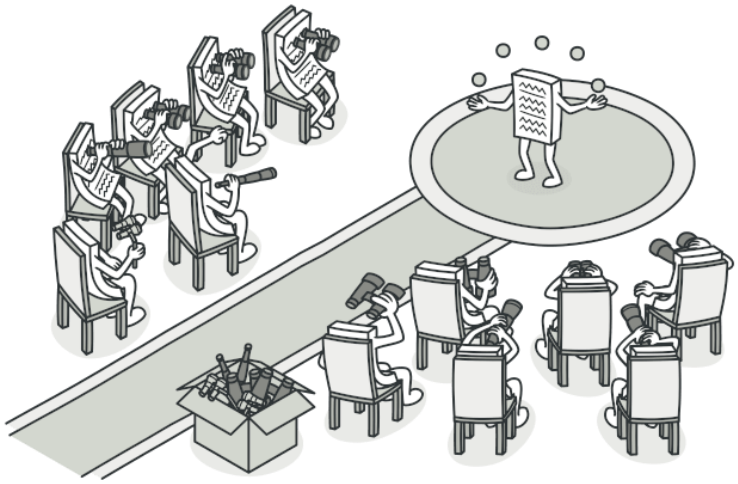
در ابتدا به بررسی کلاس‌های داده‌های هوشیار بپردازیم. داده‌های هوشیار به گونه‌ای الگوی طراحی مشاهده‌گر را در خود پیاده‌سازی نموده‌اند. ابتدا به الگوی طراحی مشاهده‌گر می‌پردازیم و به طور خلاصه این موضوع را بررسی می‌کنیم. در ادامه به عنوان نمونه سه کلاس از کلاس‌های داده‌های هوشیار در اندروید را بررسی می‌کنیم. همچنین در ادامه به نحوه پیاده سازی آن نیز اشاره‌ای خواهیم داشت. در نهایت هم به تفاوت‌های آن با کلاس‌های درون‌ساخته زبان جاوا و نحوه‌ی صحیح استفاده از آن در معماری MVVM می‌پردازیم.

# ۲ الگوی طراحی مشاهده‌گر

برای آنکه بتوانیم به موضوع داده‌های هوشیار یا پویا در چارچوب اندروید بپردازیم، لازم است ابتدا با مفهوم الگوی طراحی مشاهده‌گر آشنا باشیم. الگوی مشاهده‌گر شامل دو قسمت می باشد، مشاهده‌گر[۱۱] و مشاهده‌شونده[۱۲]. مشاهده‌شونده یکسری متغیرها دارد که وضعیت آن کلاس مشاهده‌شونده را مشخص می‌نماید. مشاهده‌گرها به گونه‌ای وابسته به آن متغیرها و وضعیت‌های مشاهده‌شونده هستند. برای همین نیاز داریم با به وجود آمدن تغییری در وضعیت آن‌ها باخبر شده و وضعیت برنامه خود را به روز رسانی کنیم.

به عنوان مثال، یک کلاس را به عنوان بیانگر وضعیت آب و هوایی در نظر بگیرید. این کلاس یک متغیر به نام isCloudly دارد که نشان دهنده آن است که هوا ابری است یا خیر. کلاسهای دیگری ممکن است وجود داشته باشند که با توجه به وضعیت این متغیر در این کلاس رفتارهای مختلفی داشته باشند. فرض کنید اگر هوا ابری باشد یک بازی فوتبال کنسل خواهد شد یا اگر هوا ابری نباشد سرعت بیشینه یک ماشین برای کلاس ماشین بیشتر خواهد بود. برای همین همه این کلاس‌ها که وابسته به این متغیر مشاهده شونده هستند یک مشاهده گر هستند و باید این کلاس را اصطلاحا مشاهده کنند.[۱۳]

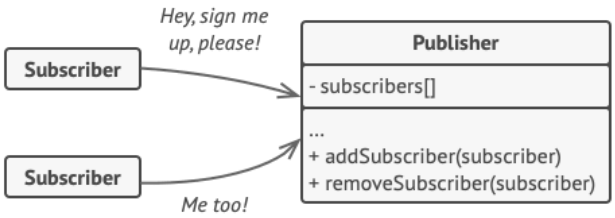
[9] Framework
[10] Model View Controller
[11] Observer
[12] Observable
[13] Observe



شکل ۱ الگوی طراحی مشاهده‌گر[۲]

## ۲٫۱ پیاده‌سازی این الگوی طراحی

برای اینکه بتوانیم همچنین الگوی طراحی را پیاده‌سازی نمایید، نیاز است که لیستی از مشاهده‌گرها در کلاس مشاهده‌شونده نگه‌داری بشود. به عملیات افزودن یک مشاهده‌گر به مشاهده شونده، ثبت[14] گفته می‌شود و به عملیات حذف یک کلاس یا مشاهده‌گر از لیست مشاهده‌شونده، حذف[15] گفته می شود. تا زمانی که رفتار یک کلاس وابسته به وضعیت یک کلاس مشاهده‌شونده باشد، این کلاس باید در لیست مشاهده‌شونده قرار داشته باشد و در آن ثبت شود. زمانی که رفتار این کلاس مستقل از مشاهده‌شونده می‌شود، باید آن را از این لیست حذف نمایید.[۲]




شکل ۲ نمودار کلاس ها[۲]

با داشتن این لیست از مشاهده‌گرها می‌توانیم به راحتی آنها را از تغییری که در متغیر کلاس مشاهده‌شونده ایجاد می‌شود، باخبر سازیم. برای این کار لازم است که تغییراتی که در جهت وضعیت کلاس مشاهده شونده است با استفاده از یک تابع تنظیم کننده[16] صورت گیرد. همانطور که در کد زیر می بینید ابتدا وضعیت کلاس مشاهده‌شونده را تغییر می‌دهیم و سپس به ازای همه مشاهده‌گرها در لیست مشاهده‌شونده یک بار تابع onChange یا تغییر را صدا خواهیم زد. همچنین مقدار جدید برای آن متغیر را در آرگومان انتخابه قرار می دهیم کلاس های مشاهده گر با گرفتن این آرگومان می‌توانند تغییرات و رفتارهایی که وابسته به این وضعیت بوده را کنترل کنند و در صورت نیاز به‌روزرسانی انجام دهند.

[14] Register/Subscribe
[15] Unregister/Unsubscribe
[16] Setter

این کد نمونه ای از پیاده‌سازی الگوری مشاهده‌گر در جاوا است.[۲]

```
package refactoring_guru.observer.example.publisher;

import refactoring_guru.observer.example.listeners.EventListener;

import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class EventManager {
    Map<String, List<EventListener>> listeners = new HashMap<>();

    public EventManager(String... operations) {
        for (String operation : operations) {
            this.listeners.put(operation, new ArrayList<>());
        }
    }

    public void subscribe(String eventType, EventListener listener) {
        List<EventListener> users = listeners.get(eventType);
        users.add(listener);
    }

    public void unsubscribe(String eventType, EventListener listener) {
        List<EventListener> users = listeners.get(eventType);
        users.remove(listener);
    }

    public void notify(String eventType, File file) {
        List<EventListener> users = listeners.get(eventType);
        for (EventListener listener : users) {
            listener.update(eventType, file);
        }
    }
}
```

# ۳ داده‌های هوشیار

حال که با الگوی طراحی مشاهده‌گر آشنا شدیم می‌توانیم به موضوع داده‌های هوشیار در اندروید بپردازید. استفاده از داده‌های هوشیار در زمان هایی که ما از معماری MVVM استفاده می کنیم می‌تواند مفید باشد. در معماری MVVM ما با هر تغییر در سمت مدل هایمان نیاز است که رابط گرافیکی را به طور کلی آپدیت کنیم. این داده‌های هوشیار در اینجا به ما بسیار کمک می کند. به‌گونه‌ای که هرنوع تغییر وضعیتی که نیاز به به‌روزرسانی رابط کاربری[۱۷] داشته باشد را می توانیم مدیریت کنیم. با استفاده از این داده‌های پویا منتظر تغییرات وضعیت بمانیم و به محض آن که تغییری در وضعیت این متغیرها به وجود آمد به‌روزرسانی‌های لازم را برای رابطه گرافیکی انجام دهیم.

مبحث داده‌های هوشیار در چهار چوب اندروید در سال‌های اخیر بسیار مورد استفاده قرار گرفته است به‌گونه ای که ابزار های مدیریت پایگاه داده[۱۸] برای مدیریت داده‌های کاربر بر بروی گوشی، مانند کتابخانه Room که توسط گوگل ساخته شده است، از این داده‌های هوشیار برای پیاده سازی پایگاه داده استفاده کرده‌اند و بصورت همروند عمل میکنند. قبل از صحبت در مورد نحوه استفاده از داده‌های پویا در MVVM لازم است که به تفاوت آن‌ها با مشاهده‌شونده اشاره کنیم و همچنین انواع اصلی آن را با هم بررسی کنیم.

## ۳٫۱ تفاوت کلاس داده‌ی هوشیار و تفاوت با مشاهده‌شونده عادی

در کل یک کلاس به نام LiveData وجود دارد که تقریبا می توان گفت همان کلاس مشاهده‌شونده ما است. فرق اصلی داده‌های پویا با کلاس مشاهده‌شونده درون‌ساخته‌شده در زبان جاوا این مسئله هست که در LiveData لیست مشاهده‌شونده به صورت مدیریت شده قرار دارد و حضور مشاهده‌گرها در این لیست با وضعیت زنده بودن اجزا اندروید تکامل یافته و همروند شده است. به‌گونه‌ای که با پایان یافتن عمر یکی از اجزای اندروید مانند یک صفحه[۱۹] تمام مشاهده‌گرهای وابسته به آن چرخه حیات از بین خواهند رفت[۱] و حذف خواهند شد. این عملیات نمونه‌ای از مزایای کلاس داده هوشیار بر کلاس مشاهده شونده در زبان جاوا است چرا که این موضوع باعث

[17] User Interface
[18] Database Management Systems Tools
[19] Activity

می‌شود ما به مشکل نشت مموری نخوریم. همچنین این روند به ما کمک می‌کند تا دیگر لازم نباشد به صورت دستی بخواهیم رابط کاربری گرافیکی را برای برنامه خود کنترل نماییم.

## ۳/۲ نحوه ساخت یک داده هوشیار

کلاس‌های داده‌های هوشیار از مفهوم چند نوعی[20] استفاده می‌کنند بصورتی که ما نوع متغییر و وضعیت را در هنگام تعریف این نوع مشخص میکنیم.

```
public MutableLiveData<String> currentName = new MutableLiveData();
```

## ۳/۳ بررسی کلاس‌های داده‌های هوشیار

### ۳/۳/۱ کلاس LiveData [۳]

ما در اینجا می‌خواهیم به بررسی سه نمونه از کلاس‌های داده پویا در اندروید بپردازیم. اولین و پایه‌ای‌ترین کلاس که به عبارتی پدر کلاس‌های دیگر هم است. کلاس LiveData که دقیقا همین الگوی طراحی مشاهده‌گر را در خود پیاده‌سازی کرده است. نکته‌ای که در مورد این کلاس وجود دارد این است که متغیرهایی و شی‌هایی که از این جنس ساخته می شوند قابل تغییر نخواهند بود، یعنی نمیتوان وضعیت آن‌ها را تغییر داد. پس تابعی در آنها موجود نیست که با استفاده از آن‌ها بتوان وضعیت مشاهده‌شونده را تغییر داد. در اینجا شاید سوال به وجود بیاید که چرا باید همچین کلاسی را داشته باشیم. جلوتر در زمانی که از استفاده داده پویا در لایه View Model صحبت می کنیم به فلسفه وجود این کلاس پی میبریم.

### ۳/۳/۲ کلاس MutableLiveData [۴]

کلاس دومی که در اینجا قصد بر بررسی آن را داریم، کلاس داده هوشیار تغییرپذیر یا MutableLiveData است. همان‌گونه که از اسم این کلاس مشخص است برخلاف پدرش که همان LiveData است، می‌توان با استفاده از توابع تنظیم‌کننده‌ای مقدار آن را اصطلاحا تنظیم کرد[21] یا فرستاد[22]. پس مقدار وضعیت و متغیر درون این داده پویا را تغییر می‌توان داد. بعد از این تغییر همه مشاهده‌کنندگان موجود در لیست این مشاهده‌شونده از مقدار جدید این متغیر در سرتاسر برنامه باخبر خواهند شد و می‌توانند رفتار خود را به‌روزرسانی نمایند.

### ۳/۳/۳ کلاس MediatorLiveData [۵]

کلاس دیگری که بررسی نکرده‌ایم کلاس MediatorLiveData است که خود فرزند MutableLiveData هست. حال فرق که اینکه باکلاس پدر خود دارد این است که می تواند تعدادی منبع[23] را در خود جای دهد. این منابع از جنس داده پویا هستند یعنی این کلاس خود یک داده پویا است که شامل منابعی از داده پویا خواهد بود. حال با تغییر وضعیت در هرکدام از آن منابع نیاز است که تغییری در متغیر این مشاهده‌شونده داده بشود. در حقیقت این کلاس هم مشاهده‌شونده است هم مشاهده‌گر. این منابع مانند مشاهده‌گرها می‌توانند در این کلاس ثبت یا حذف بشنود.

## ۳/۴ فرق تنظیم یک مقدار با فرستادن یک مقدار

ما برای تغییر وضعیت متغیر درون این داده پویا نیاز داریم که از دو تابع setValue یا postValue استفاده کنیم. فرق این دو متد در بحث موازی‌بودن یا متوالی‌بودن آنها است. اگر از تابع setValue استفاده کنیم به محض اجرای آن خط از کد مقدار جدید در متغیر قرار داده شده و تمام مشاهده‌گرها با مقدار جدید آن وضعیت به روز رسانی خواهد شد. این در حالی است که اگر تابع postValue را استفاده کنیم، داده به صورت موازی تغییر پیدا می‌کند و این بدین معنی است که ممکن است چندین لحظه بعد از اجرای آن خط از کد

[20] Generics
[21] Set
[22] Post
[23] Source

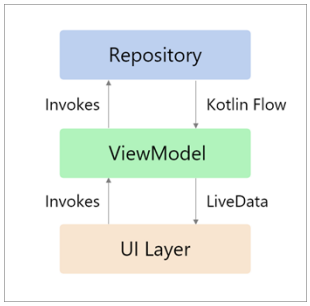
تازه وضعیت متغیر عوض بشود. برای همین به‌روزرسانی مشاهده‌گرها و اجرای خط های بعدی کد به‌گونه‌ای به صورت موازی و هم روند صورت می‌گیرد. پس باید توجه مهمی به این موضوع داشته باشیم که این اجرا شدن کد ها به صورتی است که ادامه اجرای کد دچار مشکلی نشود. بطور خلاصه تابع تنظیم کننده این تغییر وضعیت را در نخواره اصلی[۲۴] انجام می‌دهد در حالی که عملیات فرستادن در نخواره‌های غیر اصلی[25] انجام می‌گیرد.

نکته بسیار مهمی که در اینجا وجود دارد این است که در هنگام تغییر وضعیت اگر در نخواره اصلی نباشیم به هیچ وجه نمی‌توانیم از تابع تنظیم کننده استفاده کنیم. در صورت استفاده از این تابع در نخواره ای به جز نخواره اصلی با پیامی مبنا بر ممنوعیت استفاده از این تابع در زمان اجرای برنامه[۲۶] مواجه خواهیم شد و برنامه ما نابود[۲۷] می‌شود.[۱]

# ۴ استفاده از داده‌های هوشیار در معماری MVVM

## ۴/۱ تعریف در لایه ذخیره‌کننده

استفاده درست از مفهوم داده‌های پویا در معماری MVVM بدین شکل است که در لایه ذخیره‌سازی[۲۸] اطلاعات، متغیرهایی از نوع داده پویا داریم. لازم به بیان این موضوع دیده می‌شود که لایه ذخیره‌سازی بطوری مسئولیت نگهداری وضعیت برنامه را به عهده دارد. حال این لایه می‌تواند تابعی از رفتار کاربر باشد یا می تواند بیانگر وضعیت پایگاه داده ما و یا پاسخی که از سمت ابر[۲۹] می آید باشد. این داده‌های هوشیار باید از نوع MutableLiveData باشند.




*شکل ۳ روند طراحی در MVVM[1]*

## ۴/۲ تعریف در لایه View Model

در قسمت دوم، یعنی View Model ما باید متغیرهایی را برای کلاس View Model تعریف کنیم. این متغییرها از LiveData هستند و مقادیر آن‌ها برابر متغیر متناظر آن‌ها در لایه ذخیره‌سازی است. حال سوالی که می‌تواند به وجود بیاید این است که ما در لایه‌ی ذخیره‌سازی از نوع MutableLiveData استفاده کردیم ولی در لایه View Model آنها را برابر متغیرهایی از نوع LiveData بودند قرار می دهیم. چگونه مشکلی به وجود نمی آید؟ بحث چندریختی[۳۰] یکی از اجزاء برنامه نویسی شی گرا[۳۱] است. همان‌گونه که می‌دانید یک متغیر از نوع پدش می‌تواند برابر یک متغیر از نوع بچه‌های خود قرار بگیرد و این دقیقاً یعنی همان چندریختی. پس مشکلی به وجود نخواهد آمد. تنها تأثیری که این کار ما خواهد داشت این است که کدهای نوشته شده در لایه View بدون اجازه ما قادر به تغییر

[24] Main Thread / UI Thread
[25] Background Threads
[26] Runtime
[27] Crash
[28] Repository
[29] Cloud
[30] Polymorphism
[31] Object Oriented Programming

وضعیت این داده های پویا نخواهند داشت مگر با استفاده از توابعی که در این View Model تعریف می کنید و در آن ها دوباره با استفاده از توابع که در ذخیره‌شونده موجود است به تغییر وضعیت آن داده پویا می پردازیم. البته لازم به ذکر است که راه‌های دیگری برای پیاده‌سازی وجود دارند که بستگی به شرایط دارند ولی ساختار اصلی به همین شکل است. همان گونه که دیدید کلاس لایو دیتا در View Model باعث شد که ما با وجود اینکه داده‌های پویایی از وضعیت کنونی برنامه داریم ولی نتوانیم در عین حال مقدار آن را تغییر دهیم. یعنی به‌گونه‌ای این رابطه یک طرفه است و تنها قادر به استفاده از آخرین وضعیت دقیق آن متغیر در مشاهده‌شونده هستیم و تمام روابط ما با لایه بعدی کنترل شده است. همه متدهایی که تغییری بوجود می‌آورند را باید خودمان تعریف کنیم.

## ۴/۳ لایه رابط کاربری یا VIEW

تنها کاری که لازم است انجام بدهیم این است که از تابع ()observe که در کلاس LiveData پیاده‌سازی شده است استفاده کنیم که به عنوان ورودی این تابع نیاز است که صاحب چرخه[۳۲] جز اندرویدی که در آن هستیم را قرار بدهیم. همچنین به عنوان آرگومان دوم یک تابع صدا زننده[۳۳] تعریف می شود که مسئول رفتار کلاس مشاهده‌گر است و همیشه به محض ایجاد تغییر در وضعیت متغیر مشاهده شونده، از تغییر و مقدار جدید آن برای متغیر باخبر می‌شود و می‌تواند وضعیت برنامه را مانند رابط گرافیکی کاربر را به‌روزرسانی نماید.

کد زیرنمونه استفاده از موارد بالا است.[۱]

```
// Create the observer which updates the UI.
final Observer<String> nameObserver = new Observer<String>() {
    @Override
    public void onChanged(@Nullable final String newName) {
        // Update the UI, in this case, a TextView.
        nameTextView.setText(newName);
    }
};

// Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.
model.getCurrentName().observe(this, nameObserver);
```

[32] Life Cycle Owner
[33] Callback Function

# ۵ منابع


[۱] D. A. Document. "Live Data." https://developer.android.com/topic/libraries/architecture/livedata (accessed.

[۲] Refactoring.guru. "Observer Design Pattern." https://refactoring.guru/design-patterns/observer (accessed.

[۳] D. A. Document. "Live Data Class." https://developer.android.com/reference/android/arch/lifecycle/LiveData (accessed.

[٤] D. A. Document. "Mutable Live Data." https://developer.android.com/reference/android/arch/lifecycle/MutableLiveData (accessed.

[٥] D. A. Document. "Mediator Live Data." https://developer.android.com/reference/android/arch/lifecycle/MediatorLiveData (accessed.