آموزش برنامه نویسی اندروید
در محیط اندروید استودیو

خداحافظ ()findViewById؛ سلام View Binding

مدرس : سیدمهدی مطهری

www.android-studio.ir

به نام خدا

View Binding

www.android-studio.ir

فکر می‌کنم عنوان این مبحث تا حد زیادی هدف این جلسه را برای شما روشن کرده باشد. View Binding در اندروید جایگزینی شایسته برای findViewById است که کار معرفی یک view در اکتیویتی یا فرگمنت را برای ما انجام می‌داد.

قبلا برای تعریف هر view لازم بود یکبار متد findViewById فراخوانی شود که علاوه بر افزایش حجم کدها، وقت زیادی را از برنامه نویس اندروید می‌گرفت. در این جلسه به بررسی قابلیت View Binding می‌پردازیم که ما را از شر findViewById راحت کرده و باعث افزایش سرعت کار و همچنین کاهش حجم کدهای پروژه می‌گردد. البته مزایای دیگری هم دارد که در ادامه جلسه به آن می‌پردازیم.

View Binding چیست؟

View Binding یکی از امکانات زیر مجموعه‌ی Jetpack است که در کنفرانس IO گوگل در سال ۲۰۱۹ معرفی و در سال ۲۰۲۰ در Android Studio 3.6 و به عبارت دیگر Gradle 3.6 امکان فعالسازی و استفاده از آن مهیا شد. بنابراین استفاده از این قابلیت تنها از این نسخه و به بالا امکان پذیر است.

اولین مزیت View Binding این است که برای فعالسازی روی اندروید استودیو نیازی به افزودن (import) یک کتابخانه اضافی به پروژه اندرویدی نیست و درون پلاگین Gradle اندروید استودیو تعبیه شده است. بنابراین صرفا لازم است در فایل گریدلِ پروژه آنرا فعال کنیم.

همانطور که در ابتدای جلسه اشاره شد، View Binding جایگزینی برای تعریف view ها به شیوه سنتی آن یعنی استفاده از findViewById به شمار می‌رود.

اگر در حالت عادی در یک لایه XML تعداد ۱۰ عدد view (مانند Button، TextView، EditText و...) داشته باشیم برای هرکدام و به صورت جداگانه باید با استفاده از findViewById آنها را در Activity یا Fragment تعریف کنیم. چیزی شبیه به اکتیویتی زیر:

```
package ir.android_studio.testapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private Button sendBtn;
    private TextView textView;
    private EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        sendBtn = findViewById(R.id.send_btn);
        textView = findViewById(R.id.txt_view);
        editText = findViewById(R.id.edit_text);

    }
}
```

علاوه بر نیاز به نوشتن و تکرار کدهای اضافی، ایرادات و باگ‌هایی این شیوه‌ی معرفی view ها دارد که با جایگزینی آن با View Binding این ایرادات مرتفع می‌گردد.

با استفاده از View Binding نیازی به تعریف view ها توسط برنامه نویس نبوده و فراخوانی view ها به صورت مستقیم توسط id (شناسه) آنها انجام می‌شود!

نکته: اگر قبلا اصطلاح Data Binding را در جایی دیده یا شنیده‌اید آنرا با View Binding اشتباه نگیرید. البته تا قبل از معرفی View Binding از Data Binding هم برای انجام این کار استفاده می‌شد اما از آنجایی که قابلیت‌های Data Binding به حذف findViewById محدود نمی‌شود، تیم توسعه اندروید

زیرمجموعه‌ای از آن را با نام View Binding معرفی کرد که به صورت تخصصی تنها یک وظیفه را برعهده داشته باشد.

ضمن اینکه در مستندات اندروید هم توصیه شده چنانچه صرفا به قابلیت تعریف view ها در پروژه خود نیاز داشته باشیم بجای Data Binding از View Binding استفاده کنیم. بکارگیری View Binding نسبت به Data Binding ساده‌تر بوده و کارایی (Performance) بهتری نیز به همراه خواهد داشت.

مقایسه View Binding در اندروید با سایر ابزارها و روش‌ها

برای تعریف view ها در اندروید چندین روش و کتابخانه وجود دارد که هرکدام مزایا و معایب مخصوص به خود را داشته و البته در نهایت به این نتیجه می‌رسیم که استفاده از View Binding در اندروید بهینه ترین روش فعلی خواهد بود.

در ادامه برای درک بهتر برتری View Binding نسبت به رقبا، به بررسی هرکدام می‌پردازیم:

findViewById

نیازی به معرفی نیست بنابراین به بررسی مزایا و معایب می‌پردازم:

عدم تاثیر در سرعت بیلد: شاید تنها مزیت آن را بتوان عدم تاثیر بر سرعت بیلد پروژه دانست چرا که هنگام اجرای برنامه کار می‌کند. اما این مزیت در برابر معایب متعدد آن اصلا قابل توجیه نیست.

کدهای اضافی: اولین امتیاز منفی این است که به ازاء هر view می‌بایست یک متغیر جداگانه ایجاد شود که طبیعتا حجم کدهای ما را افزایش خواهد داد.

Type safety نیست: واژه type به معنی "نوع" و safety به معنی "ایمنی" است. Type safety نبودن findViewById به این معنی است که در هنگام تعریف کردن یک view امکان بروز اشتباه در تعیین نوع آن وجود دارد.

برای مثال ممکن است view از جنس TextView باشد و ما اشتباها آن را در یک متغیر از جنس EditText تعریف کنیم.

Null safe نیست: اگر یک view موجود در layout فقط در شرایطی خاص در دسترس باشد ممکن است با یک خطای NullPointerException مواجه شویم. زیرا findViewById وضعیت نال بودن یا نبودن view را در شرایط مختلف بررسی نمی‌کند.

برای مثال حالتی را درنظر بگیرید که دو نسخه از activity_main.xml در پروژه داریم که یکی برای حالت عادی (صفحه عمودی یا portrait) و دیگری برای حالت افقی (landscape) استفاده می‌شود. چنانچه یک view در هردو layout مشترک نبوده و تنها در حالت portrait بکار رفته باشد هنگام قرار گرفتن دستگاه کاربر در حالت landscape می‌تواند سبب بروز خطای null شود.

کتابخانه ButterKnife

تا قبل از معرفی Data Binding توسط تیم اندروید، کتابخانه ButterKnife یکی از پرکاربردترین ابزار برای این منظور به شمار می‌رفت. البته کاربرد این کتابخانه محدود به bind (متصل) کردن view ها نمی‌شود و برای resource ها مانند رشته‌ها، رنگ‌ها و سایزها نیز کاربرد دارد. اما در این جلسه تنها قسمت مربوط به view ها مدنظر ماست.

کاهش حجم کدها: استفاده از این کتابخانه حجم کدها را نسبت به findViewById کاهش داده و به عبارتی کدهای تمیز تری می‌نویسیم. در این روش، به وسیله Annotation ها (حاشیه نویسی) می‌توان view های مدنظر را تعریف کرد. سه view ای که در قسمت قبل توسط findViewById تعریف شده بود را اینبار با استفاده از ButterKnife تعریف می‌کنم.

پس از اضافه کردن کتابخانه به پروژه، توسط annotation با نام @BindView هرکدام از view ها قابل تعریف است:

```
package ir.android_studio.testapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import butterknife.BindView;

public class MainActivity extends AppCompatActivity {

    @BindView(R.id.send_btn) Button sendBtn;
    @BindView(R.id.txt_view) TextView textView;
    @BindView(R.id.edit_text) EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```

Type safety نیست: مانند findViewById کتابخانه ButterKnife هم type safety نیست و id هر نوع view ای را می‌توان به هر نوع View در کلاس جاوا متصل کرد. بنابراین ممکن است در هنگام اجرا با ارور Exception برخورد کنیم.

کاهش سرعت بیلد: از آنجایی که در حین کامپایل پروژه یک پردازشگر Annotation برای تولید (generate) کدها اجرا می‌شود در سرعت بیلد تاثیر منفی می‌گذارد. هرچند این کاهش سرعت بیلد چیزی نیست که ما را از استفاده یک کتابخانه منصرف کند زیرا به قول معروف مزایای آن بر معایبش می‌چربد. البته نه الان که View Binding را داریم!

Data Binding

تفاوت اساسی Data Binding با دو مورد قبل در این است که پس از فعالسازی این قابلیت در پروژه اندرویدی، به ازاء هر layout یا سایر resource ها به صورت خودکار یک کلاس ایجاد (generate) می‌شود. در این کلاس‌ها تمامی عناصر به صورت خودکار تعریف و مقدار دهی شده و به سادگی می‌توانیم در اکتیویتی یا فرگمنت به آنها دسترسی داشته باشیم.

کاهش حجم کدها: فعالسازی و استفاده از Data Binding نسبت به دو گزینه قبل نیاز به نوشتن کد کمتری دارد در نتیجه کدهای تمیزتری خواهیم داشت.

کاهش سرعت بیلد: در کلاس‌های ساخته شده توسط Data Binding از Annotation ها برای تعریف منابع استفاده می‌شود بنابراین مانند ButterKnife این مساله در کاهش سرعت build شدن پروژه تاثیرگذار خواهد بود.

Type safety و Null safe است: هردو مورد در Data Binding صدق می‌کند و از این بابت جای نگرانی نداریم.

در خصوص Data Binding بیشتر از این به بیان جزئیات نمی‌پردازم.

View Binding

رسیدیم به هدف! همانطور که در ابتدای مبحث گفته شد View Binding به نوعی زیر مجموعه‌ی Data Binding محسوب می‌شود که کاربرد آن در حذف findViewById خلاصه شده و به همین جهت علاوه بر ساده تر شدن فرآیند، نسبت به Data Binding خروجی بهینه‌تری را در اختیار ما قرار می‌دهد.

کاهش حجم کدها: با فعالسازی و استفاده از View Binding حجم کد مورد نیاز برای استفاده از view ها به حداقل ممکن می‌رسد و در بین این ۴ گزینه بالاترین امتیاز را به خود اختصاص داده است. تنها با نوشتن id هر view در اکتیویتی یا فرگمنت به view مربوطه دسترسی خواهیم داشت!

افزایش سرعت بیلد: شیوه کار View Binding هم مانند Data Binding است و با فعالسازی آن به ازاء هر layout یک کلاس ایجاد شده و تمامی view های آن درون کلاس به صورت خودکار تعریف می‌شود. اما در اینجا خبری از پردازش annotation ها نیست و در نتیجه سرعت بیلد به نسبت Data Binding بالاتر خواهد بود.

Type safety و Null safe است: این دو مورد را از پدر خودش یعنی Data Binding به ارث برده و با یکدیگر مشترک هستند.

حالا که گزینه‌های مختلف را بررسی کردیم و فهمیدیم بهترین گزینه در حال حاضر View Binding است در ادامه جلسه این قابلیت را در قالب یک پروژه بررسی و تمرین می‌کنیم.

ساخت پروژه View Binding و بررسی آن

ابتدا طبق مبحث آموزش ساخت پروژه در اندروید استودیو یک پروژه اندرویدی با نام ViewBinding می‌سازم. اکتیویتی را از نوع Empty Activity و زبان را Java انتخاب کردم.

در ابتدا و قبل از هرچیز لازم است قابلیت View Binding را در پروژه خود در اندروید استودیو فعال کنیم.

فعالسازی View Binding در اندروید استودیو

همانطور که قبلا اشاره شد این قابلیت به صورت پیش فرض در گریدل نسخه ۳.۶ به بالا تعبیه شده و برای فعالسازی آن نیازی به اضافه کردن کتابخانه به پروژه نیست.

برای فعالسازی کافیست بلاک زیر را درون بلاک android در فایل build.gradle (project) اضافه و سپس پروژه را Sync کنم:

```
buildFeatures {
    viewBinding = true
}
```

نکته: چنانچه از اندروید استودیو ۳.۶ و تا قبل از ۴.۰ استفاده می‌کنید فعالسازی View Binding به صورت زیر انجام می‌شود:

```
viewBinding {
    enabled = true
}
```

البته واضح است که استفاده از نسخه‌های قدیمی اندروید استودیو توصیه نمی‌شود.

پیاده سازی View Binding در اکتیویتی

همانطو که قبلا گفته شد، با فعالسازی View Binding به ازاء هر layout موجود در پروژه یک کلاس جداگانه ایجاد می‌شود. نام هر کلاس از نام layout مربوط به آن گرفته می‌شود که البته به صورت camel case نوشته شده و پسوند Binding هم به انتهای آن اضافه می‌شود.

برای مثال در این پروژه ما یک layout با نام activity_main.xml داریم. کلاسی که برای این layout ایجاد می‌شود ActivityMainBinding.java نام دارد. به عنوان یک مثال دیگر چنانچه در آینده یک

layout با نام activity_map.xml به پروژه اضافه کنیم، کلاس ایجاد شده ActivityMapBinding.java نام خواهد داشت.

در این کلاس به ازاء هر view (ویجت) ای که در activity_main.xml وجود داشته و یک id به آن اختصاص داده شده باشد، یک متغیر ایجاد می‌شود و ما در اکتیویتی به آن دسترسی خواهیم داشت.

در layout پیش فرض پروژه یک View Group از جنس ConstraintLayout و یک TextView وجود دارد که البته برای TextView شناسه (id) به صورت پیش فرض تعریف نشده. بنابراین برای آنکه به این view دسترسی داشته باشم یک id به آن اختصاص می‌دهم:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
      android:id="@+id/txt_view"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="parent" />

  </androidx.constraintlayout.widget.ConstraintLayout>
```

در مرحله بعد لازم است یک آبجکت (شیء) از کلاس ActivityMainBinding داخل اکتیویتی تعریف کنیم تا به view ها به صورت مستقیم دسترسی داشته باشیم.

```
activity_main.xml   MainActivity.java
1   package ir.android_studio.viewbinding;
2
3   import ...
5
6   public class MainActivity extends AppCompatActivity {
7
8       private ActivityMai
9           ActivityMainBinding ir.android_studio.viewbinding.databinding
10      @Ove  Press Enter to insert, Tab to replace
11      protected void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.activity_main);
14      }
15  }
                                                www.android-studio.ir
```

مشاهده می‌کنید کلاس ActivityMainBinding توسط اندروید استودیو شناسایی می‌شود. یعنی کلاس قبلا ساخته شده و قابل استفاده است.

MainActivity.java

```
package ir.android_studio.viewbinding;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

import ir.android_studio.viewbinding.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding mainBinding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mainBinding = ActivityMainBinding.inflate(getLayoutInflater());

        setContentView(R.layout.activity_main);


    }
}
```

نام دلخواه mainBinding را برای کلاس انتخاب کردم. سپس درون متد onCreate و قبل از setContentView توسط متد inflate آنرا مقداردهی کردم.

حالا به View Group اصلی layout و تمام view هایی که قبلا به layout اضافه شده و یا بعد از این اضافه شود دسترسی خواهیم داشت. البته مجدد تاکید می‌کنم view هایی که دارای ویژگی id باشند. بجز عنصر اصلی یا ریشه‌ی layout که نیازی به id نداشته و در متغیری با نام root ذخیره شده و توسط getRoot در دسترس است. در اینجا عنصر ریشه ما یک ConstraintLayout است که قبلا در جلسه آموزش کار با ConstraintLayout با آن آشنا شدیم.

ابتدا لازم است getRoot را به متد setContentView پاس بدهیم تا به اکتیویتی اعلام شود از طریق آبجکت Binding ای که ساخته‌ایم به layout دسترسی داشته باشد. getRoot را جایگزین R.layout.activity_main می‌کنم:

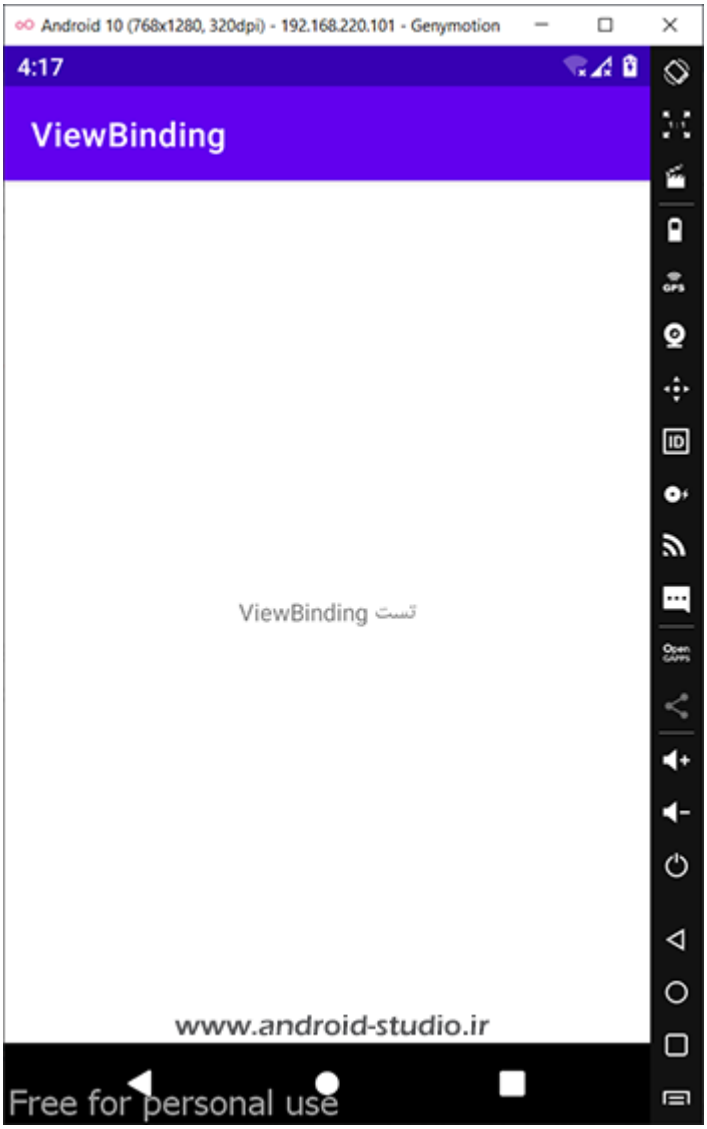
```
setContentView(mainBinding.getRoot());
```

حالا برای تست View Binding با استفاده از متد setText یک متن را روی txt_view چاپ می‌کنم:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mainBinding = ActivityMainBinding.inflate(getLayoutInflater());

    setContentView(mainBinding.getRoot());

    mainBinding.txtView.setText("تست ViewBinding");

}
```

نکته: نحوه نامگذاری id ها توسط View Binding به صورت camel case است. بنابراین همانطور که در کد فوق مشاهده می‌کنید txtView مربوط به txt_view موجود در layout است.

پروژه را روی امولاتور (شبیه ساز) اندرویدی اجرا می‌کنم:



طبق تصویر فوق هم layout نمایش داده شد و هم متنی که در setText قرار داده بودیم.

برای تمرین و آشنایی بیشتر، یک Button به layout اضافه کرده و در اکتیویتی یک setOnClickListener برای آن تعریف می‌کنم. قبلا در آموزش کار با رویدادها در اندروید با این متد آشنا شده‌ایم. خط مربوط به setText را به درون رویداد مربوط به دکمه منتقل می‌کنم تا بعد از کلیک روی دکمه اجرا شود:

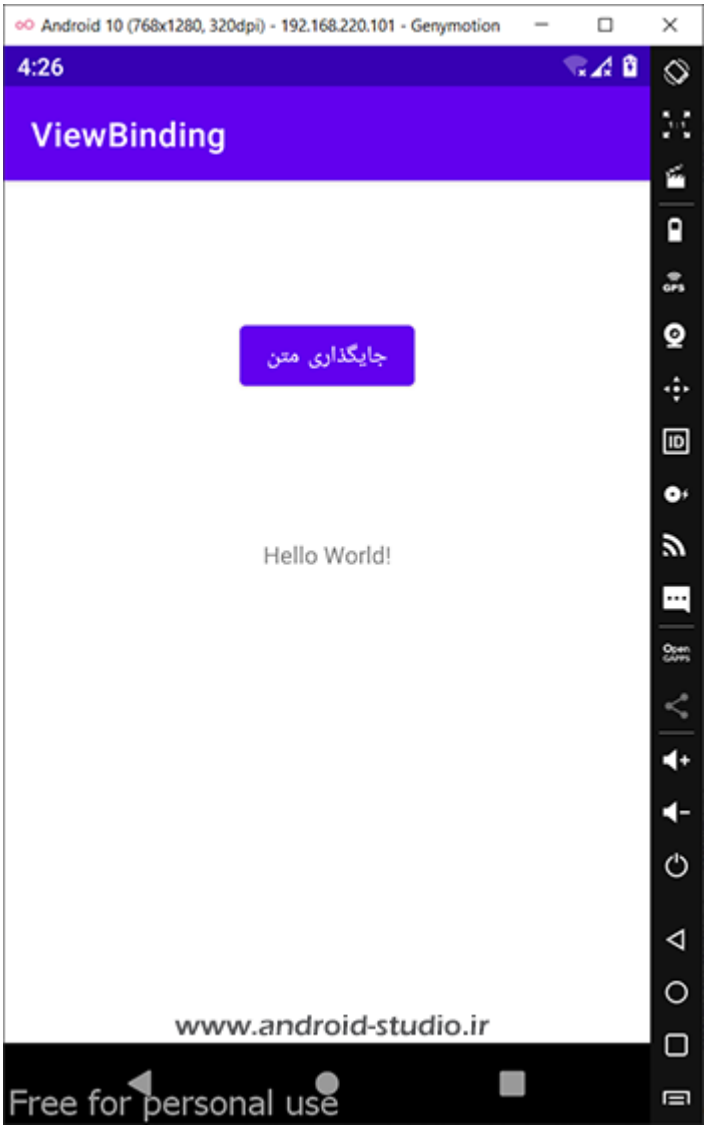
activity_main.xml

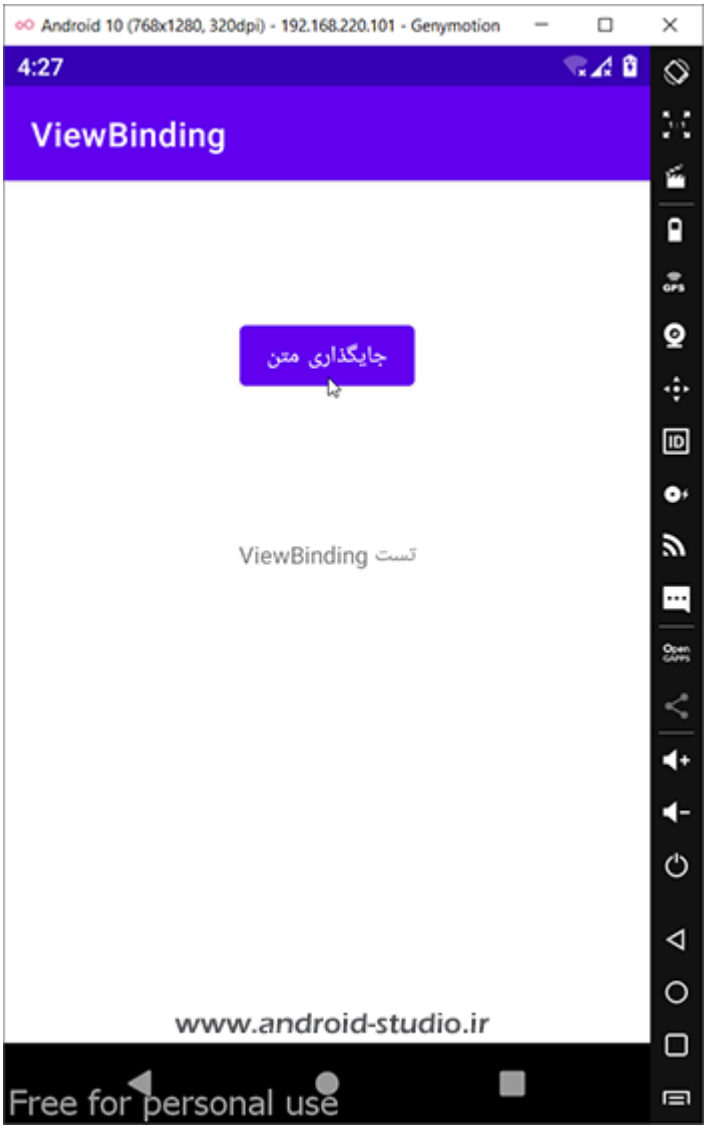
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/set_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="80dp"
        android:text="جایگذاری متن"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/txt_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/set_btn" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.java

```
package ir.android_studio.viewbinding;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

import ir.android_studio.viewbinding.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding mainBinding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mainBinding = ActivityMainBinding.inflate(getLayoutInflater());

        setContentView(mainBinding.getRoot());

        mainBinding.setBtn.setOnClickListener(view -> {

            mainBinding.txtView.setText("تست ViewBinding");

        });

    }
}
```

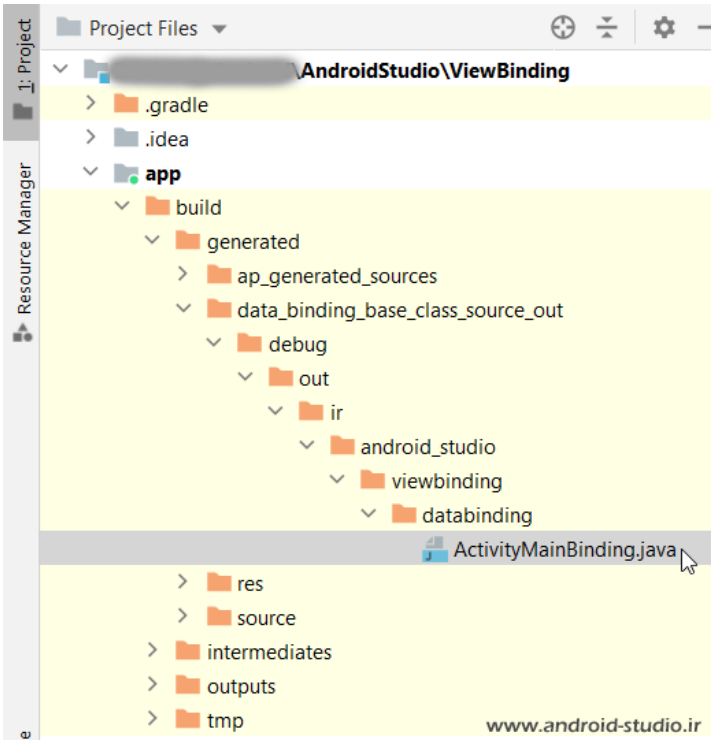
مجدد پروژه را اجرا کرده و روی دکمه جایگذاری متن کلیک می‌کنم:


Android 10 (768x1280, 320dpi) - 192.168.220.101 - Genymotion
4:26
ViewBinding
جایگذاری متن
Hello World!
www.android-studio.ir
Free for personal use



دسترسی به کلاس‌های View Binding

اگر مایل بودید به محتوای کلاس‌های ساخته شده توسط View Binding دسترسی داشته باشید، بعد از بیلد شدن پروژه (یعنی هنگام اجرای پروژه روی دیوایس یا گزینه Rebuild Project در تب Build و یا گرفتن خروجی APK از پروژه) در مسیر زیر فایل کلاس‌ها در دسترس هستند:



برای نمایش محتوای دایرکتوری build لازم است نحوه نمایش پروژه در حالت Project و یا Project Files قرار گیرد. البته در پوشه‌ای که پروژه ذخیره شده هم می‌توان به فایل‌ها دسترسی داشت.

نکته: در صورت عدم نمایش کلاس‌های مربوط به View Binding در قسمت نمایش ساختار پروژه در اندروید استودیو و البته بعد از بیلد شدن پروژه، راست کلیک کرده و Reload from Disk کنید تا محتوای پروژه بروز شود.

محتوای فعلی کلاس ActivityMainBinding.java پروژه من به اینصورت است:

ActivityMainBinding.java

```
// Generated by view binder compiler. Do not edit!
package ir.android_studio.viewbinding.databinding;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.constraintlayout.widget.ConstraintLayout;
```

```
import androidx.viewbinding.ViewBinding;
import ir.android_studio.viewbinding.R;
import java.lang.NullPointerException;
import java.lang.Override;
import java.lang.String;

public final class ActivityMainBinding implements ViewBinding {
  @NonNull
  private final ConstraintLayout rootView;

  @NonNull
  public final Button setBtn;

  @NonNull
  public final TextView txtView;

  private ActivityMainBinding(@NonNull ConstraintLayout rootView, @NonNull Button setBtn,
      @NonNull TextView txtView) {
    this.rootView = rootView;
    this.setBtn = setBtn;
    this.txtView = txtView;
  }

  @Override
  @NonNull
  public ConstraintLayout getRoot() {
    return rootView;
  }

  @NonNull
  public static ActivityMainBinding inflate(@NonNull LayoutInflater inflater) {
    return inflate(inflater, null, false);
  }

  @NonNull
  public static ActivityMainBinding inflate(@NonNull LayoutInflater inflater,
      @Nullable ViewGroup parent, boolean attachToParent) {
    View root = inflater.inflate(R.layout.activity_main, parent, false);
    if (attachToParent) {
      parent.addView(root);
    }
    return bind(root);
  }

  @NonNull
  public static ActivityMainBinding bind(@NonNull View rootView) {
    // The body of this method is generated in a way you would not otherwise write.
    // This is done to optimize the compiled bytecode for size and performance.
    int id;
    missingId: {
      id = R.id.set_btn;
      Button setBtn = rootView.findViewById(id);
      if (setBtn == null) {
        break missingId;
      }

      id = R.id.txt_view;
      TextView txtView = rootView.findViewById(id);
      if (txtView == null) {
```

```
        break missingId;
      }

      return new ActivityMainBinding((ConstraintLayout) rootView, setBtn, txtView);
    }
    String missingId = rootView.getResources().getResourceName(id);
    throw new NullPointerException("Missing required view with ID: ".concat(missingId));
  }
}
```

تذکر: همانطور که در کامنت ابتدای کلاس ذکر شده این کلاس توسط کامپایلر View Binding ساخته شده و نباید به هیچ عنوان به صورت دستی ویرایش شود.

ملاحظه می‌کنید برای همه متغیرها انوتیشین @NonNull قید شده یعنی هیچکدام از آیتم‌ها تحت هیچ شرایطی نال نخواهند شد.

اما فرض کنید در پروژه خود یک activity_main.xml دیگر هم داشته باشیم که مربوط به طراحی صفحه در حالت Landscape (افقی) باشد و در این حالت یکی از view ها را حذف کرده باشیم. یا بلعکس یک view در این حالت اضافه کرده باشیم که قرار نیست در layout اصلی یعنی حالت عمودی وجود داشته باشد. در اینصورت بجای @NonNull انوتیشین @Nullable جایگزین خواهد شد که نشان می‌دهد این view می‌تواند در شرایطی نال باشد. بنابراین وضعیت نال مدیریت شده و به اصطلاح Null safe هستند.

به متد انتهای کلاس دقت کنید:

```
@NonNull
public static ActivityMainBinding bind(@NonNull View rootView) {
  // The body of this method is generated in a way you would not otherwise write.
  // This is done to optimize the compiled bytecode for size and performance.
  int id;
  missingId: {
    id = R.id.set_btn;
    Button setBtn = rootView.findViewById(id);
    if (setBtn == null) {
      break missingId;
    }

    id = R.id.txt_view;
    TextView txtView = rootView.findViewById(id);
    if (txtView == null) {
      break missingId;
    }

    return new ActivityMainBinding((ConstraintLayout) rootView, setBtn, txtView);
```

```
  }
  String missingId = rootView.getResources().getResourceName(id);
  throw new NullPointerException("Missing required view with ID: ".concat(missingId));
}
```

حدس می‌زنم انتظارش را نداشتید اینجا با findViewById مواجه شوید! اما واقعیت جز این نیست :)

View Binding از متد findViewById برای اتصال view ها به اکتیویتی استفاده می‌کند و انجام این مرحله تکراری و زجر آور را از روی دوش برنامه نویس برمی‌دارد. علاوه بر آن، در خصوص type و null هم ما را ایمن نگه می‌دارد.

View Binding و layout های include شده

اگر بخاطر داشته باشید در جلسه آموزش کار با Navigation Drawer با استفاده از تگ include یک layout را درون layout اصلی برنامه اضافه کردیم.

دسترسی به view های layout ای که در layout یک اکتیویتی include شده امکان پذیر است. یک layout جدید با نام layout_bottom.xml به پروژه اضافه کرده و یک TextView با شناسه btm_txt داخل آن تعریف می‌کنم:

layout_bottom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/btm_txt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="متن پایینی"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

سپس layout را درون activity_main.xml توسط تگ include اضافه می‌کنم. دقت داشته باشید برای دسترسی به layout ضمیمه شده و view های درون آن توسط View Binding حتما برای تگ include هم باید یک id تعریف شود. من شناسه include_layout را انتخاب کردم:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/set_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="80dp"
        android:text="جایگذاری متن"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/txt_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="252dp"
        android:text="Hello World!"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/set_btn" />

    <include
        layout="@layout/layout_bottom"
        android:id="@+id/include_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/txt_view"
        app:layout_constraintVertical_bias="0.706"
        tools:layout_editor_absoluteX="-16dp" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

حالا برای دسترسی به btm_txt در layout_bottom به اینصورت عمل می‌کنیم:

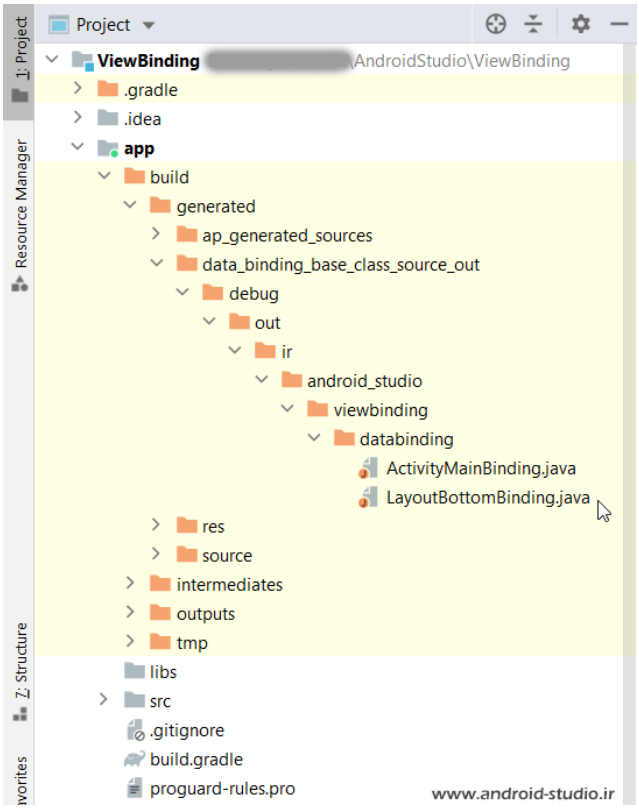
```
mainBinding.includeLayout.btmTxt
```

includeLayout همان شناسه include_layout است.

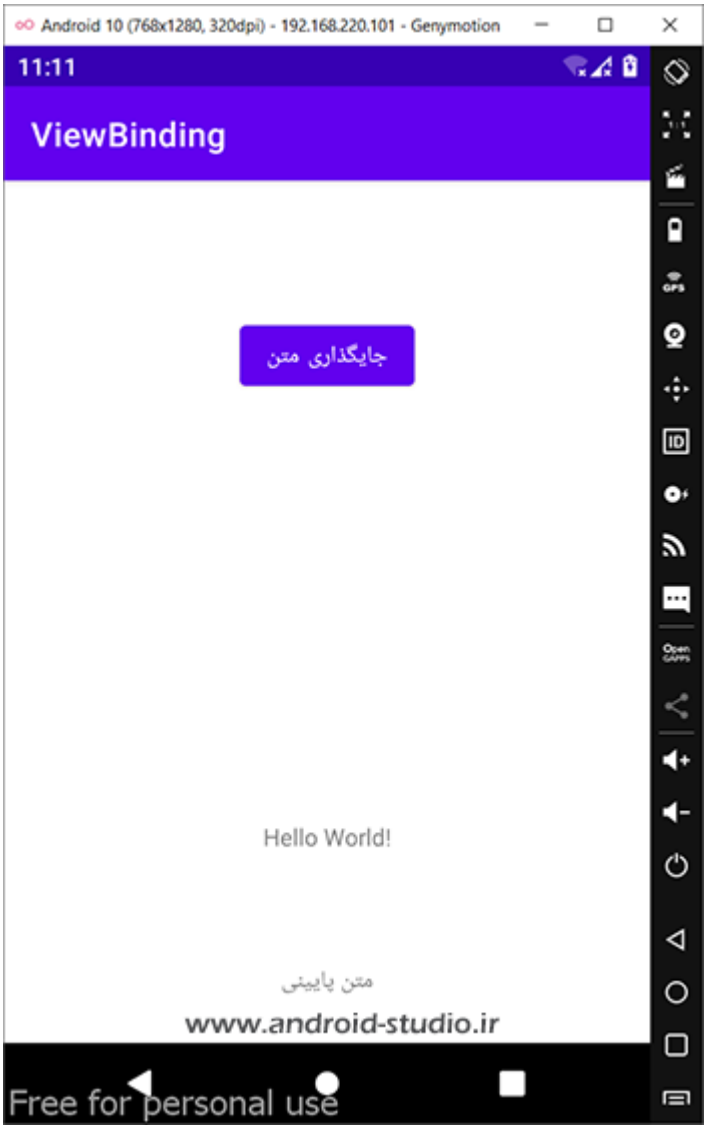
برای مثال متد setText را برای این TextView استفاده می‌کنم:

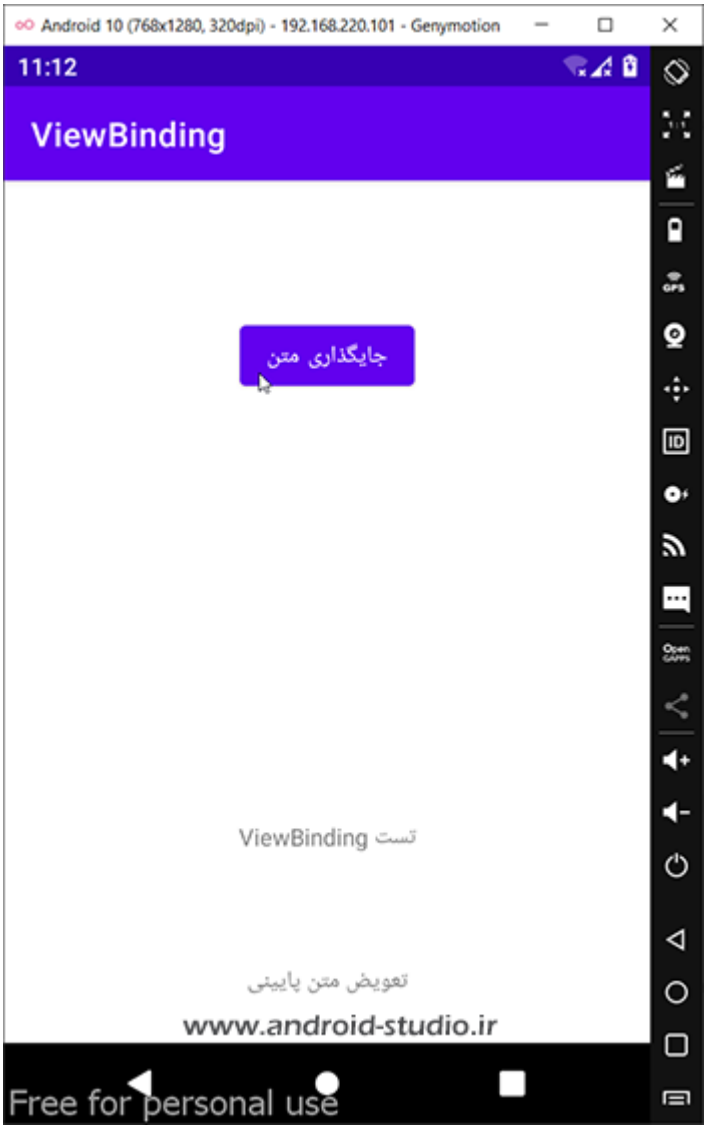
```
mainBinding.setBtn.setOnClickListener(view -> {

    mainBinding.txtView.setText("تست ViewBinding");
    mainBinding.includeLayout.btmTxt.setText("تعویض متن پایینی");

});
```

برای layout_bottom.xml هم یک کلاس ایجاد شده که LayoutBottomBinding.java نام دارد اما به دلیل اینکه در اینجا layout را درون یک layout دیگر include کرده‌ایم که قبلا یک آبجکت از آن در اکتیویتی ساخته شده، بدون نیاز به ساخت آبجکت جدید از layout زیرمجموعه، می‌توان به آن دسترسی داشت.



با اجرای پروژه و کلیک روی دکمه جایگذاری متن، متن این TextView هم تغییر می‌کند:


Android 10 (768x1280, 320dpi) - 192.168.220.101 - Genymotion
11:11
ViewBinding
جایگذاری متن
Hello World!
متن پایینی
www.android-studio.ir
Free for personal use



غیر فعال کردن View Binding برای یک لایه (layout) خاص

ممکن است در پروژه خود برای یک یا چند layout نیازی به ساخت کلاس View Binding نداشته باشیم. با توجه به اینکه افزایش تعداد کلاس‌ها در نهایت موجب کاهش سرعت بیلد و همچنین افزایش حجم پروژه می‌شود لذا می‌طلبد برای لایه‌هایی که به صورت ایستا (static) هستند و کاری با view های داخل آن نداریم، قابلیت View Binding را غیر فعال کنیم.

برای انجام این کار کافیست خط زیر به تگ layout ریشه آن اضافه شود:

```
tools:viewBindingIgnore="true"
```

برای مثال من یک لایه با نام layout_top.xml در پروژه ایجاد کرده‌ام و با توجه به ایستا بودن محتوای آن قصد دارم View Binding را روی این فایل غیر فعال کنم:

layout_top.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    tools:viewBindingIgnore="true">

</androidx.constraintlayout.widget.ConstraintLayout>
```

واژه ignore به معنی "نادیده گرفتن" و "رد کردن" است بنابراین چنانچه برای این ویژگی مقدار true تعیین شود، قابلیت View Binding برای layout مدنظر نادیده گرفته خواهد شد.

پیاده سازی View Binding روی فرگمنت

قبلا در مبحث فرگمنت‌ها در اندروید با کاربرد Fragment آشنا شدیم. استفاده از View Binding در فرگمنت تفاوت زیادی با اکتیویتی ندارد.

یک فرگمنت با نام TestFragment به پروژه اضافه می‌کنم. سپس در layout فرگمنت یعنی fragment_text.xml یک TextView با شناسه fragment_txt تعریف می‌کنم:

fragment_test.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".TestFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="متن پیش فرض فرگمنت"
        android:id="@+id/fragment_txt"/>

</FrameLayout>
```

سپس در کلاس فرگمنت مانند آنچه قبلا در اکتیویتی انجام شد یک آبجکت از کلاسی که View Binding برای layout فرگنت ایجاد کرده می‌سازم:

TextFragment.java

```
package ir.android_studio.viewbinding;

import android.os.Bundle;
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import ir.android_studio.viewbinding.databinding.FragmentTestBinding;

public class TestFragment extends Fragment {

    private FragmentTestBinding frgBinding;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        //return inflater.inflate(R.layout.fragment_test, container, false);

        frgBinding = FragmentTestBinding.inflate(inflater, container, false);

        return frgBinding.getRoot();

    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();

        frgBinding = null;

    }
}
```

مطابق کد فوق ابتدا از کلاس FragmentTestBinding یک نمونه با نام frgBinding ساخته‌ام. سپس inflate اصلی فرگمنت را حذف (کامنت) کرده و inflate مربوط به View Binding را مانند اکتیویتی و البته اندکی تفاوت بجای آن تعریف کرده‌ام. در خط بعد هم متد getRoot برگردانده یا return شد.

همچنین متد onDestroyView که مربوط به چرخه حیات فرگمنت هست را اضافه و درون آن آبجکت ساخته شده از View Binding را null می‌کنم تا هنگام حذف فرگمنت از اکتیویتی، این آبجکت نال شود.

یک متد setText برای TextView فرگمنت در onCreateView و قبل از دستور return تعریف می‌کنم:

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {

    //return inflater.inflate(R.layout.fragment_test, container, false);

    frgBinding = FragmentTestBinding.inflate(inflater, container, false);

    frgBinding.fragmentTxt.setText("متن جدید فرگمنت");

    return frgBinding.getRoot();

}
```

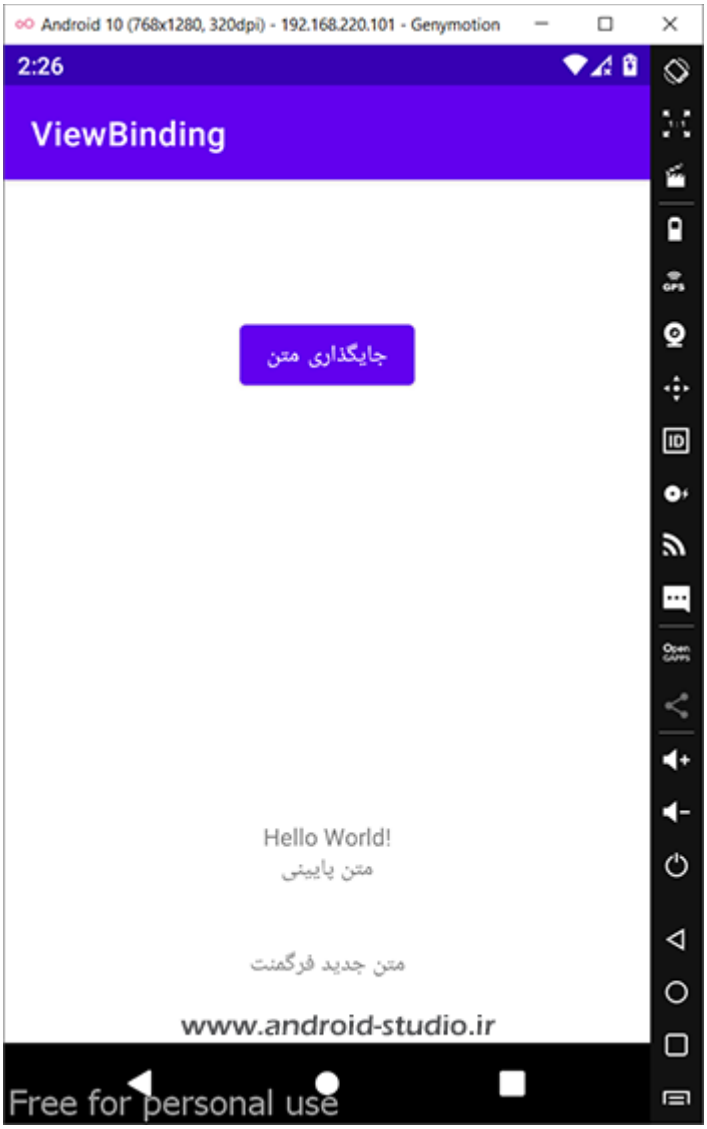
در نهایت، فرگمنت را در activity_main.xml تعریف می‌کنم. از اختصاص id به تگ fragment فراموش نکنید!

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/set_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="80dp"
        android:text="متن جایگذاری"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/txt_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="252dp"
        android:text="Hello World!"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/set_btn" />

    <include
        android:id="@+id/include_layout"
        layout="@layout/layout_bottom"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/txt_view"
        tools:layout_editor_absoluteX="-16dp" />
```

```
    <fragment
        android:id="@+id/fragment_one"
        android:name="ir.android_studio.viewbinding.TestFragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/include_layout" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

با اجرای مجدد پروژه می‌بینیم که متن موجود در setText روی TextView جایگزین متن پیش فرض آن شده:



این جلسه هم به پایان رسید.

موفق و پیروز باشید.

مطالعه بیشتر:

https://developer.android.com/topic/libraries/view-binding

https://developer.android.com/topic/libraries/data-binding

توجه: سورس پروژه درون پوشه Exercises قرار دارد

با ارائه انتقادات و پیشنهادات خود، ما را در ارائه آموزش‌های بهتر یاری فرمائید.
این فایل رایگان بوده و انتشار آن (بدون دخل و تصرف) مانعی ندارد.

www.android-studio.ir