DataBinding - Architecture Component

CAF Study Week10

Introduction of Android Architecture Component

📌 Introduction

Google I/O 2017에서 새로운 라이브러리들을 Android Architecture Components(**AAC**)로 묶어서 발표하였습니다. Google은 AAC를 안드로이드 앱을 개발하면서 자주 만날 수 있는 문제들을 쉽게 해결할 수 있는 새로운 선택지로 설명하였습니다. 사실 이미 레거시가 많이 포함된 실무 프로젝트에 AAC를 적용하는 것은 어려움이 있습니다. 하지만 AAC에는 안드로이드와 아키텍쳐에 대한 고민이 많이 담겨있기 때문에, 필수로 공부해야 한다고 생각합니다.

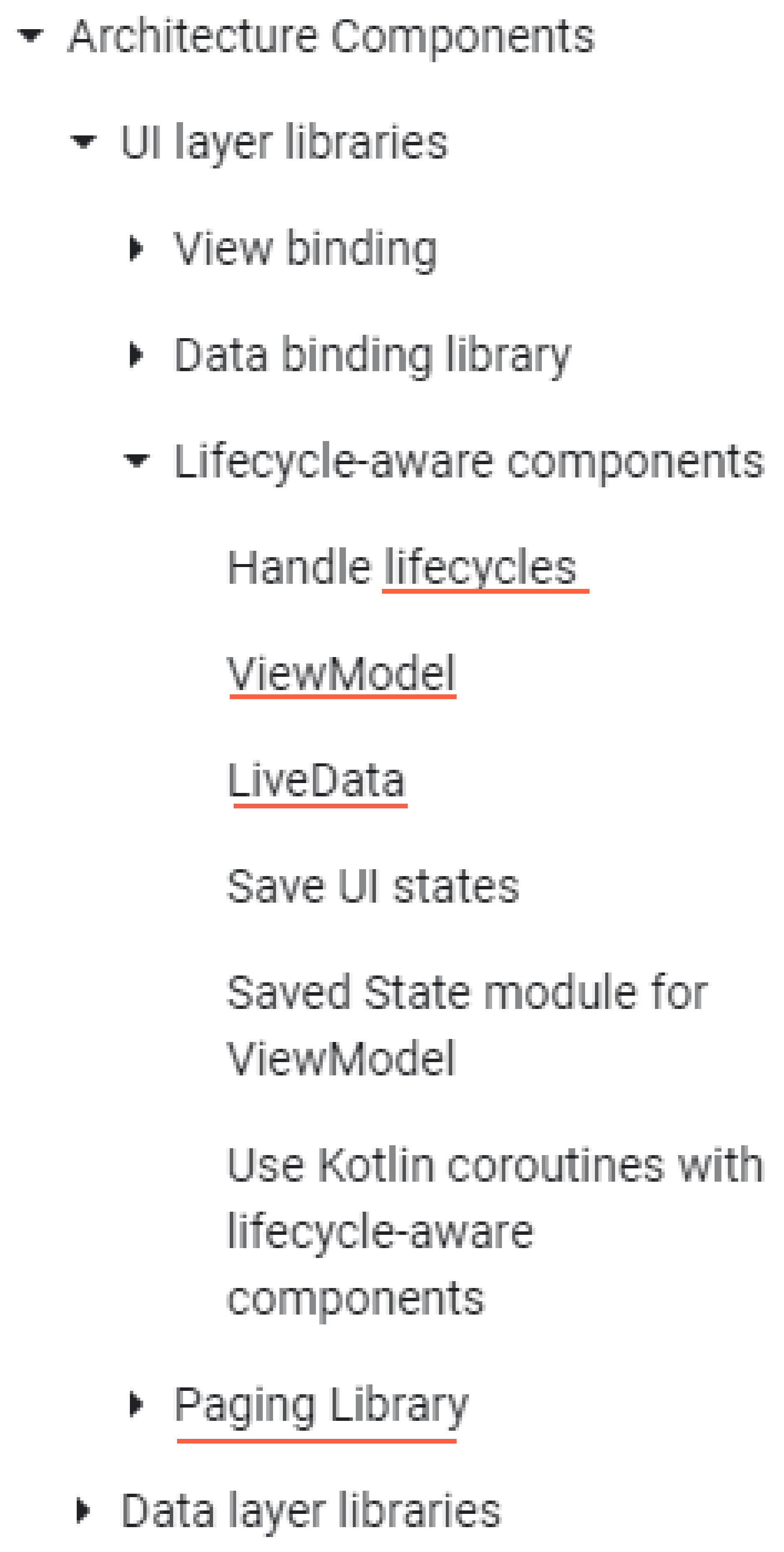
📌 왜 AAC를 만들었니?

안드로이드는 Activity, BroadcastReceiver, Service, ContentProvider 등 **여러 컴포넌트들이 있고, 생명주기가 다르게 얽혀있습니다**. 앱을 잘 만들기 위해서는 이러한 컴포넌트들을 부드럽게 연결해야 하는데, **생명주기를 학습하고 엉키지 않도록 고민**하는 것은 결국 개발자의 몫이였습니다. 구글은 이 고민을 줄이기 위해 SDK에서 제공하는 컴포넌트들에 대해 개발자들에게 더 가이드를 주기를 원했습니다. 그래서 Android Architecture Components(AAC)를 만들었습니다.

AAC는 Google I/O 2017에 발표한 4가지와 추후에 추가된 1개까지 총 5개의 라이브러리로 구성되어 있습니다.

1. Lifecycles (Easy handling lifecycles)
2. LiveData (Lifecycle aware observable)
3. ViewModel (Managing data in a lifecycle)
4. Room (Object Mapping for SQLite)
5. Paging (Gradually loading information)

📚 공식문서 목차



- Architecture Components
  - UI layer libraries
    - View binding
    - Data binding library
    - Lifecycle-aware components
      - Handle lifecycles
      - ViewModel
      - LiveData
      - Save UI states
      - Saved State module for ViewModel
      - Use Kotlin coroutines with lifecycle-aware components
    - Paging Library
  - Data layer libraries

https://medium.com/@maryangmin/android-architecture-components-%EC%86%8C%EA%B0%9C-1-8e04491be1f6

DataBinding - Architecture Component

ViewBinding

Introduction

View binding is a feature that allows you to more **easily** write code that interacts with views. Once view binding is enabled in a module, it **generates a binding class for each XML** layout file present in that **module**. An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.

Setup instructions

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

```
<LinearLayout
        ...
        tools:viewBindingIgnore="true" >
    ...
</LinearLayout>
```

특정 파일을 viewBinding에서 제외시키고 싶을 경우 위와 같이 'veiwBindingIgnore' 속성을 이용 할 수 있음

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

ViewBinding Usage

inflate() : This creates an instance of the **binding class** for the activity to use.
👉 activity가 사용할 binding class 객체를 생성해줌

layoutInflater 👉 xml을 view 객체로 변환시켜 준다

Pass the root view to **setContentView()** to make it the active view on the screen.
👉 setContentView()에 root view를 전달해줌 / screen에 view 띄워주는 메소드

The **inflate()** method requires you to pass in a layout inflater. If the layout has already been inflated, you can instead call the binding class's static **bind()** method.

Activity

```kotlin
private lateinit var binding: ResultProfileBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ResultProfileBinding.inflate(layoutInflater)
    val view = binding.root
    setContentView(view)
}
```

Fragment onViewCreated

```kotlin
class BindFragment : Fragment(R.layout.fragment_blank) {

    // Scoped to the lifecycle of the fragment's view (between onCreateView and onDestroyView)
    private var fragmentBlankBinding: FragmentBlankBinding? = null

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val binding = FragmentBlankBinding.bind(view)
        fragmentBlankBinding = binding
        binding.textViewFragment.text = getString(string.hello_from_vb_bindfragment)
    }
```

Fragment onCreateView

```kotlin
private var _binding: ResultProfileBinding? = null
// This property is only valid between onCreateView and
// onDestroyView.
private val binding get() = _binding!!

override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    _binding = ResultProfileBinding.inflate(inflater, container, false)
    val view = binding.root
    return view
}

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}
```

DataBinding - Architecture Component

Provide hints for different configurations

When you declare views across multiple configurations, occasionally it makes sense to use a different view type depending on the particular layout. For example:

```
# in res/layout/example.xml
<TextView android:id="@+id/user_bio" />

# in res/layout-land/example.xml
<EditText android:id="@+id/user_bio" />
```

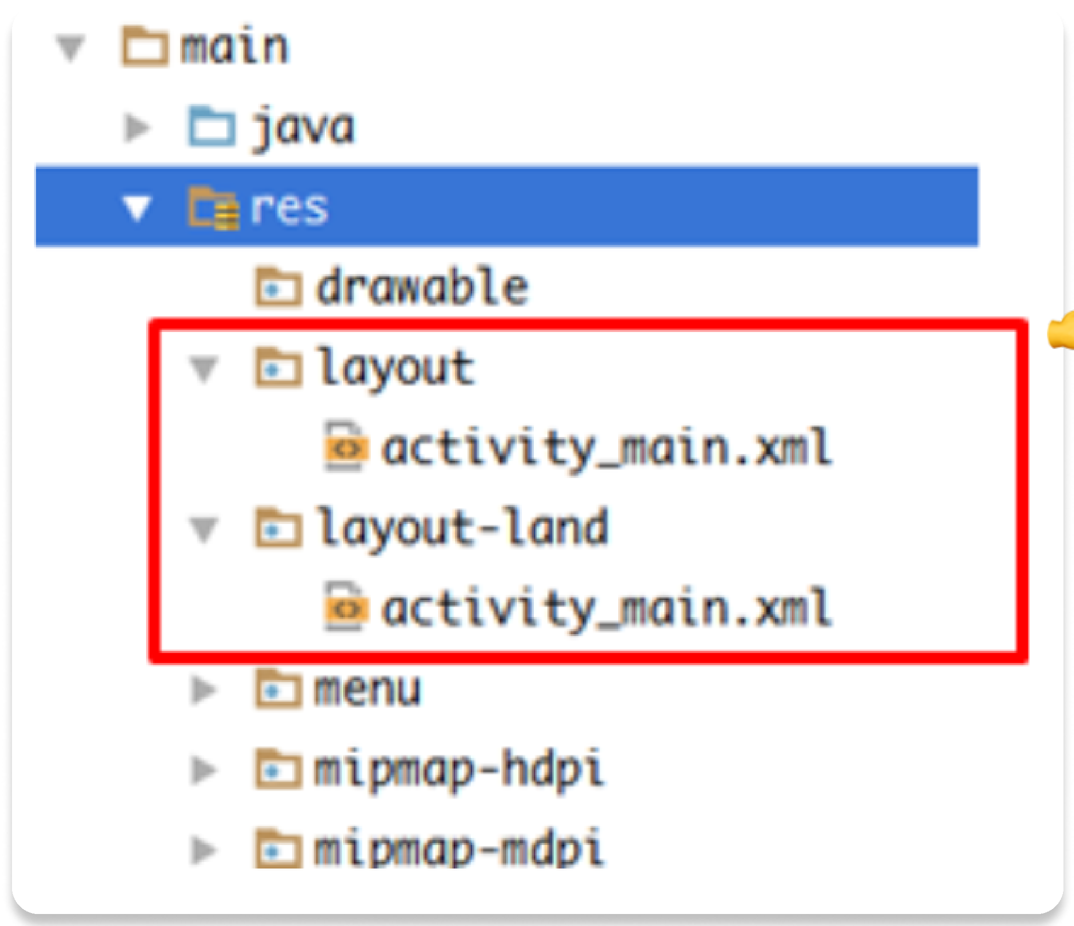
In this case, you might expect the generated class to expose a field `userBio` of type `TextView`, because `TextView` is the common base class. Unfortunately, due to technical limitations, the view binding code generator is unable to make this determination and simply generates a `View` field instead. This would require casting the field later with `binding.userBio as TextView`.

To work around this limitation, view binding supports a `tools:viewBindingType` attribute, allowing you to tell the compiler what type to use in the generated code. In the above example, you can use this attribute to make the compiler generate the field as a `TextView`:

```
# in res/layout/example.xml (unchanged)
<TextView android:id="@+id/user_bio" />

# in res/layout-land/example.xml
<EditText android:id="@+id/user_bio" tools:viewBindingType="TextView" />
```

1. 한국어 버전으로 보면 이 내용 없음
2. ViewBinding을 쓸 일이 거의 없어서 굳이 알필요는 없을 듯 함
3. 왼쪽 내용 요약하자면, 같은 layout에서 **다른 형식의 view를, 같은 id로 사용**을 하려고 한다면, **viewBindingType**을 지정해줘야 한다.
4. 이 예시를 통해 추가로 알아갈 수 있는 지식은 **layout-land** 키워드이다.

layout-landscape의 약자로써 가로모드 뷰를 나타냄



https://loveiskey.tistory.com/199

layout(-port)

port 가 생략되어 있는거라고 생각하기!

뭐.. 다들 아실거라 생각합니다^_^

Portrait : 세로모드
LandScape : 가로모드

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

DataBinding - Architecture Component

findViewById vs ViewBinding

findViewById와의 차이점

1. Null safety

Binding Class를 만들 때, id를 가진 값에 대해서 reference를 생성하기 때문에, id값에 직접 접근하는 findViewById와 다르게 NPE(Null Point Exception)이 일어날 걱정이 없다. 앞 화면에서 얘기했듯이 portrait / landscape 모드 중 단일 layout에서 사용되는 id의 경우에는 binding class에 @Nullable 어노테이션으로 표시된다.

Since view binding creates direct references to views, there's no risk of a null pointer exception due to an invalid view ID. Additionally, when a view is only present in some configurations of a layout, the field containing its reference in the binding class is marked with @Nullable.

2. Type safety

Binding Class의 각 field(생성자)는 XML에서의 view type에 기반하여 만들어지기 때문에, class cast exception이 일어날 위험성이 없다~
만약에 코드단에서 ~~멍청이처럼~~ 강제캐스팅을 잘못? 실수로? 하게되어 type이 안맞는 상황이 벌어진다 하더라도, runtime이 아니라 compile 할때 에러가 난다~

The fields in each binding class have types matching the views they reference in the XML file. This means that there's no risk of a class cast exception. These differences mean that incompatibilities between your layout and your code will result in your build failing at compile time rather than at runtime.

DataBinding - Architecture Component

dataBinding vs **ViewBinding**

DataBinding 과의 차이점

📌 장점

ViewBinding, Data Binding 둘다 view reference를 참조할 수 있는 Binding Class를 만듦. 그런데 view binding은 좀 간단한 케이스에서 사용하길 권장하고 있고, 그럴 경우에 데이터 바인딩과 비교해서 아래와 같은 이점을 가진다.
View binding and data binding both generate binding classes that you can use to reference views directly. However, view binding is intended to handle simpler use cases and provides the following benefits over data binding:

- **Faster compilation:** View binding requires no annotation processing, so compile times are faster.

- **Ease of use:** DataBinding 처럼 <layout> 태그같은거 안써도 돼서 쉽다. 사용하고 싶은 모듈의 build.gradle 파일에 viewbinding 설정해주면, 모든 레이아웃에서 바로 사용 가능하다 → 당연한 소리 ㅋㅋ

- View binding does not require specially-tagged XML layout files, so it is faster to adopt in your apps. Once you enable view binding in a module, it applies to all of that module's layouts automatically.

📌 단점

- Layout variables & Layout Expressions 사용 불가
- two-way dataBinding 안된다~!

```
<data>
    <variable name="user" type="com.example.User"/>
</data>
```

```
android:text="@{String.valueOf(index + 1)}"
android:visibility="@{age > 13 ? View.GONE : View.VISIBLE}"
android:transitionName='@{"image_" + id}'
```

- View binding doesn't support layout variables or layout expressions, so it can't be used to declare dynamic UI content straight from XML layout files.
- View binding doesn't support two-way data binding.

DataBinding - Architecture Component

dataBinding vs **ViewBinding**

DataBinding 과의 차이점

결론

viewBinding, dataBinding 둘의 장점을 살려서 필요할 때 각각 적용해서 알잘딱깔센 하자~!!

너무 DataBinding만 고수할 필요 없다는 말입니다 😎

Because of these considerations, it is best in some cases to use both view binding and data binding in a project.
You can use data binding in layouts that require advanced features and use view binding in layouts that do not.

DataBinding 공식 문서 부분에도, 그저 findViewById()를 대체하는 기능으로 databinding을 사용하고 있으면
더 좋은 성능을 가지는 viewBinding을 사용하라고 권하고 있음~!

📌 장점

ViewBinding, Data Binding 둘다 view reference를 참조할 수 있는 Binding Class를 만듦. 그런데 view binding은 좀 간단한 케이스에서 사용하길 권장하고 있고, 그럴 경우에 데이터 바인딩과 비교해서 아래와 같은 이점을 가진다.

View bin[illegible] ded to handle simpler use cases and provides the following benefits over data binding:

- **Faster compilation:** View bindin[illegible]
- **Ease of use:** DataBinding 처럼 <layout> 태그같은거 안써도 돼서 쉽다. 사용하고 싶은 모듈의 build.gradle 파일에 viewbinding 설정해주면, 모든 레이아웃에서 바로 사용 가능하다 → 당연한 소리 ㅋㅋ

- View binding does not require specially-tagged XML layout files, so it is faster to adopt in your apps. Once you enable view binding in a module, it applies to all of that module's layouts automatically.

📌 단점

- Layout variables & Layout Expressions 사용 불가
- two-way dataBinding 안된다~!

```
<data>
    <variable name="user" type="com.example.User"/>
</data>
```

```
[illegible]ing.valueOf(index + 1)}"
android:visibility="@{age > 13 ? View.GONE : View.VISIBLE}"
android:transitionName='@{"image_" + id}'
```

- View binding doesn't support layout variables or layout expressions, so it can't be used to declare dynamic UI content straight from XML layout files.
- View binding doesn't support two-way data binding.

DataBinding

ViewBinding 은 이쯤하고
DataBinding 을 본격적으로 알아보자~!!

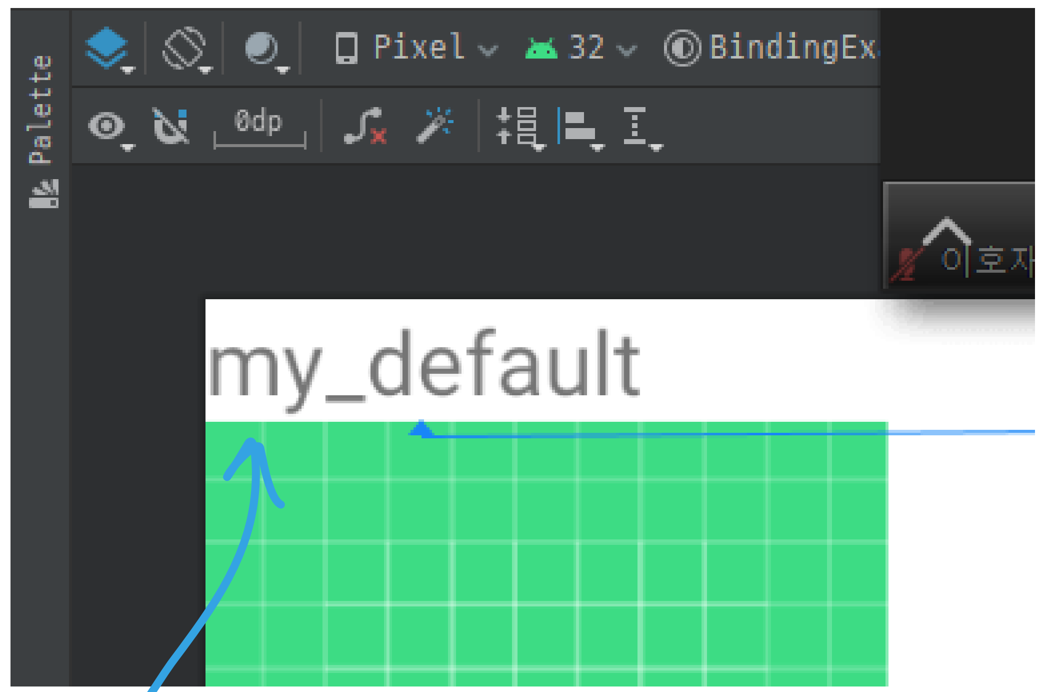
DataBinding Overview

1. **GetStarted** : Build.gralde 세팅, 개념
2. **Layouts and binding expressions** : <layout>, <data> 태그 등.. 뭐 그런 얘기임
3. **Work with observable data objects**
4. **Generated binding classes**
5. **Binding adapters**
6. **Bind layout views to Architecture Components**
7. **Two-way data binding**

DataBinding

Android Studio support for data binding

```
<TextView android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{user.firstName, default=my_default}"/>
```



tools 대신 사용 가능

Binding Data

If you are using data binding items inside a Fragment, ListView, or RecyclerView adapter, you may prefer to use the inflate() methods of the bindings classes or the DataBindingUtil class, as shown in the following code example:

```
val listItemBinding = ListItemBinding.inflate(layoutInflater, viewGroup, false)
// or
val listItemBinding = DataBindingUtil.inflate(layoutInflater, R.layout.list_item, viewGroup, false)
```

DataBinding - Architecture Component

DataBinding Expression Language

Common Features

- Mathematical `+ - / * %`
- String concatenation `+`
- Logical `&& ||`
- Binary `& | ^`
- Unary `+ - ! ~`
- Shift `>> >>> <<` signed, unsigned
- Comparison `== > < >= <=` (Note that `<` needs to be escaped as `&lt;`)
- `instanceof`
- Grouping `()`
- Literals - character, String, numeric, `null`
- Cast
- Method calls
- Field access
- Array access `[]`
- Ternary operator `?:`

Note: For the XML to be syntactically correct, you have to escape the < characters. For example: instead of List<String> you have to write List&lt;String>.

```
<data>
    <import type="android.util.SparseArray"/>
    <import type="java.util.Map"/>
    <import type="java.util.List"/>
    <variable name="list" type="List&lt;String>"/>
    <variable name="sparse" type="SparseArray&lt;String>"/>
    <variable name="map" type="Map&lt;String, String>"/>
    <variable name="index" type="int"/>
    <variable name="key" type="String"/>
</data>
```

Null coalescing operator

The null coalescing operator (`??`) chooses the left operand if it isn't `null` or the right if the former is `null`.

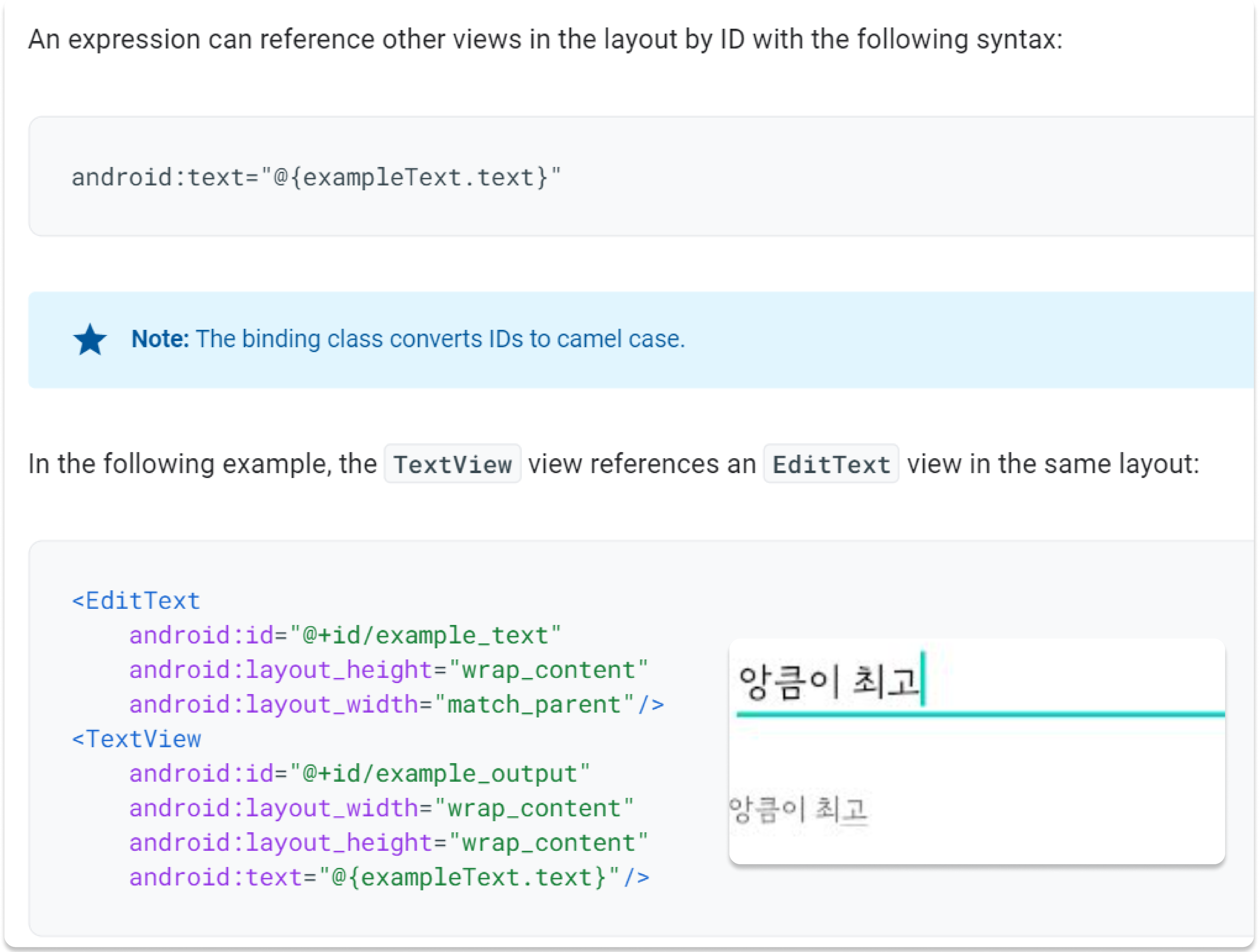
```
android:text="@{user.displayName ?? user.lastName}"
```

This is functionally equivalent to:

```
android:text="@{user.displayName != null ? user.displayName : user.lastName}"
```

DataBinding Expression Language

View Reference



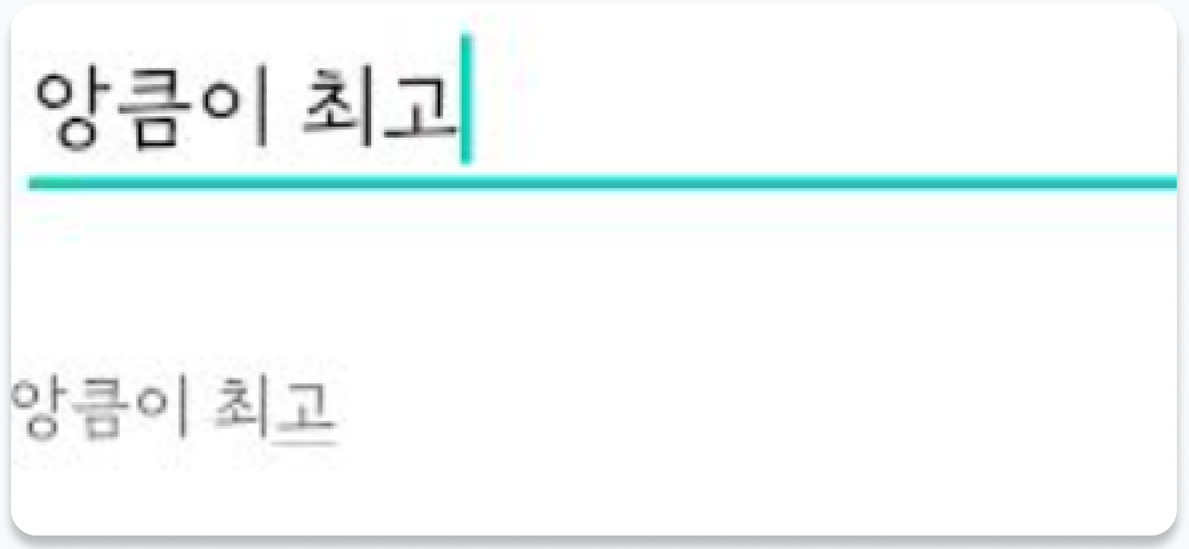
An expression can reference other views in the layout by ID with the following syntax:

```
android:text="@{exampleText.text}"
```

★ **Note:** The binding class converts IDs to camel case.

In the following example, the `TextView` view references an `EditText` view in the same layout:

```
<EditText
    android:id="@+id/example_text"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"/>
<TextView
    android:id="@+id/example_output"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{exampleText.text}"/>
```



앙큼이 최고

앙큼이 최고

Collections
String Literals
Resources

(https://developer.android.com/topic/libraries/data-binding/expressions#collections)

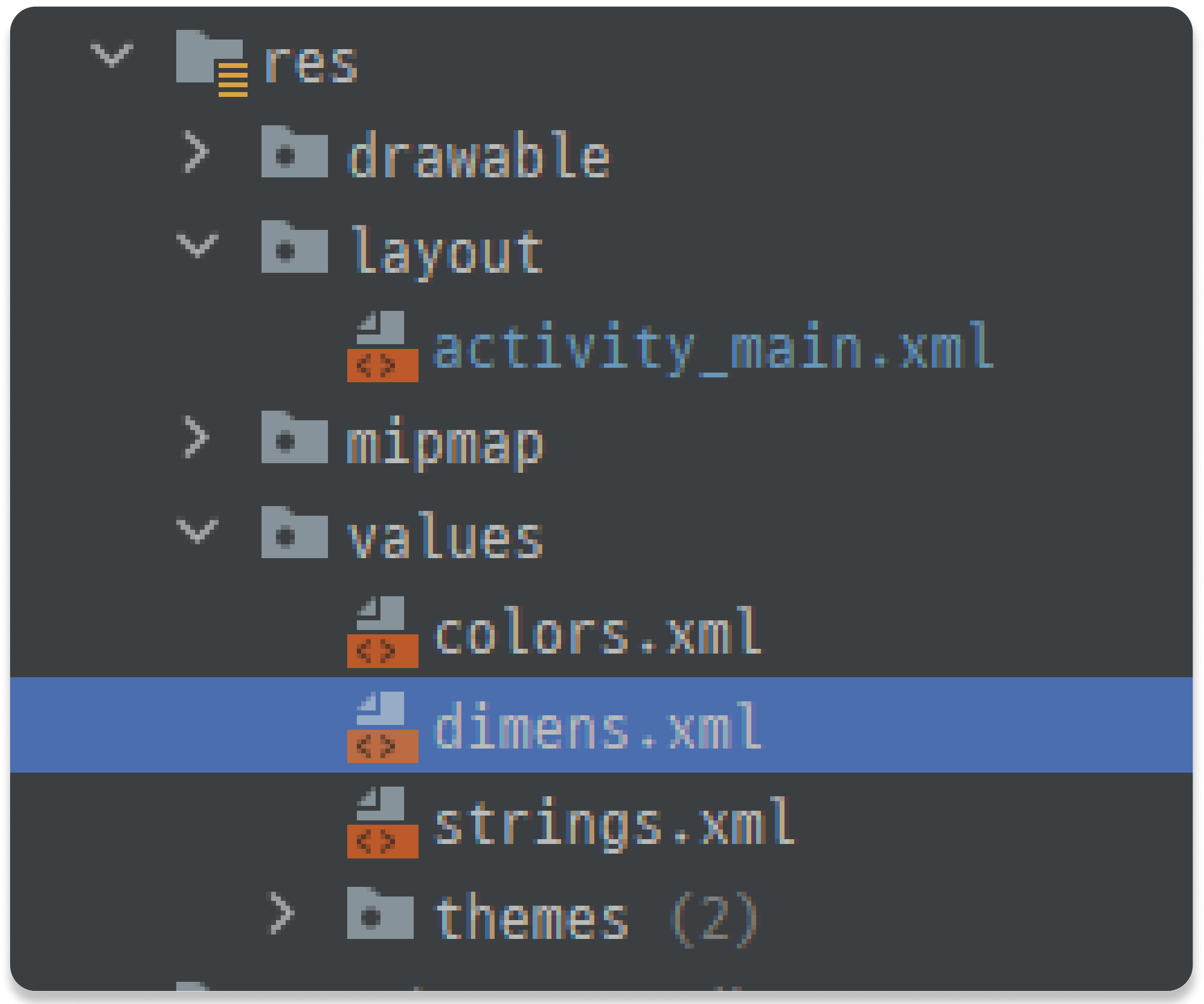
DataBinding Expression Language

Resources(1)

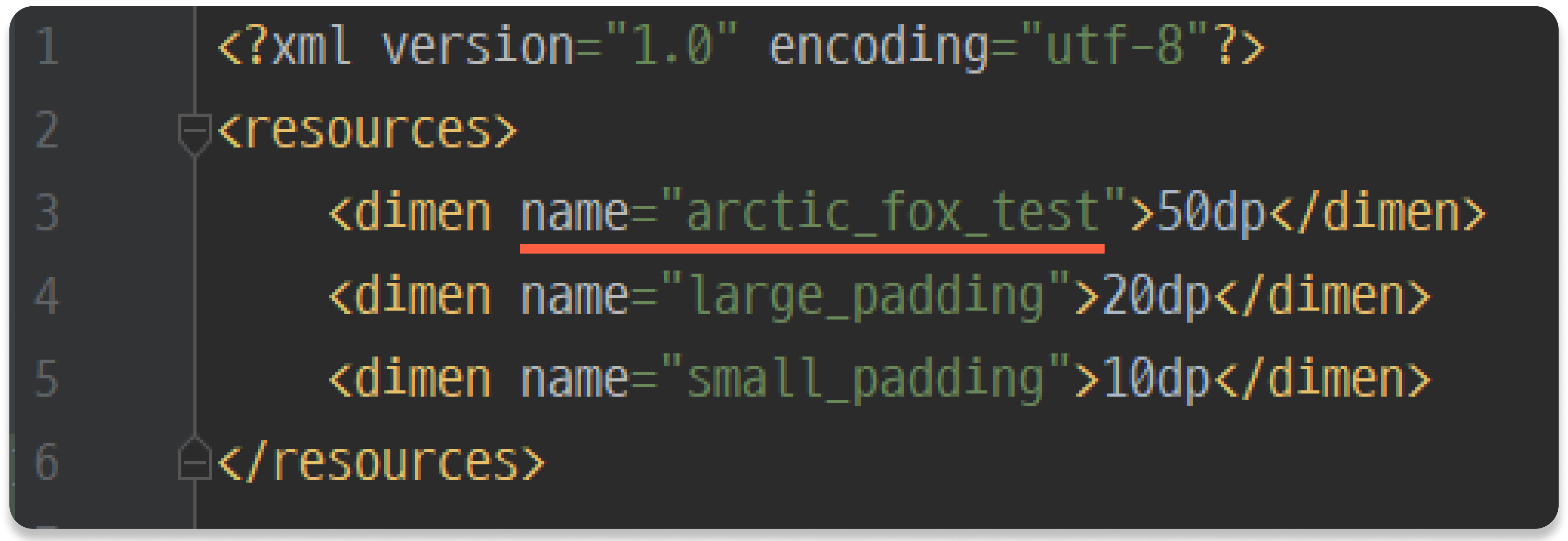
An expression can reference app resources with the following syntax:

```
android:padding="@{large? @dimen/largePadding : @dimen/smallPadding}"
```

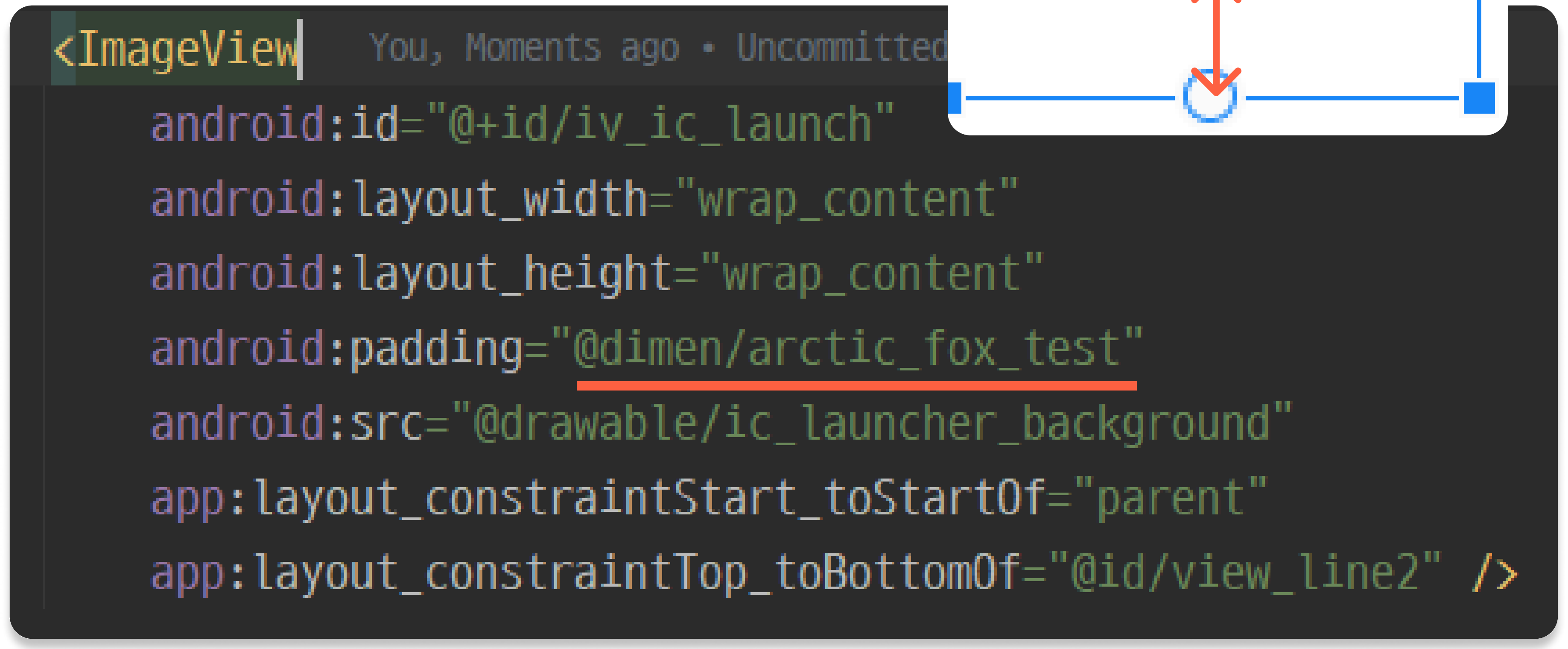
📚 알쓸신잡 Time - **diemns.xml**



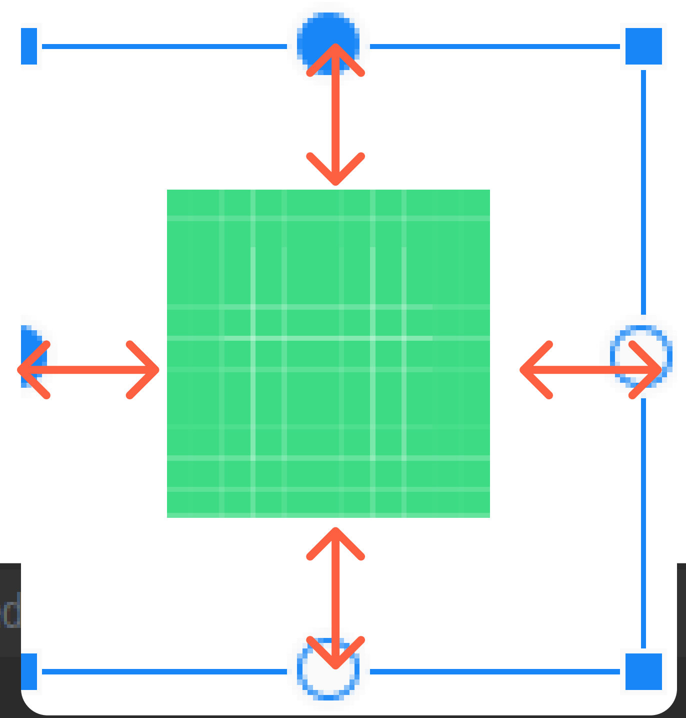
```
res
  drawable
  layout
    activity_main.xml
  mipmap
  values
    colors.xml
    dimens.xml
    strings.xml
  themes (2)
```



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="arctic_fox_test">50dp</dimen>
    <dimen name="large_padding">20dp</dimen>
    <dimen name="small_padding">10dp</dimen>
</resources>
```

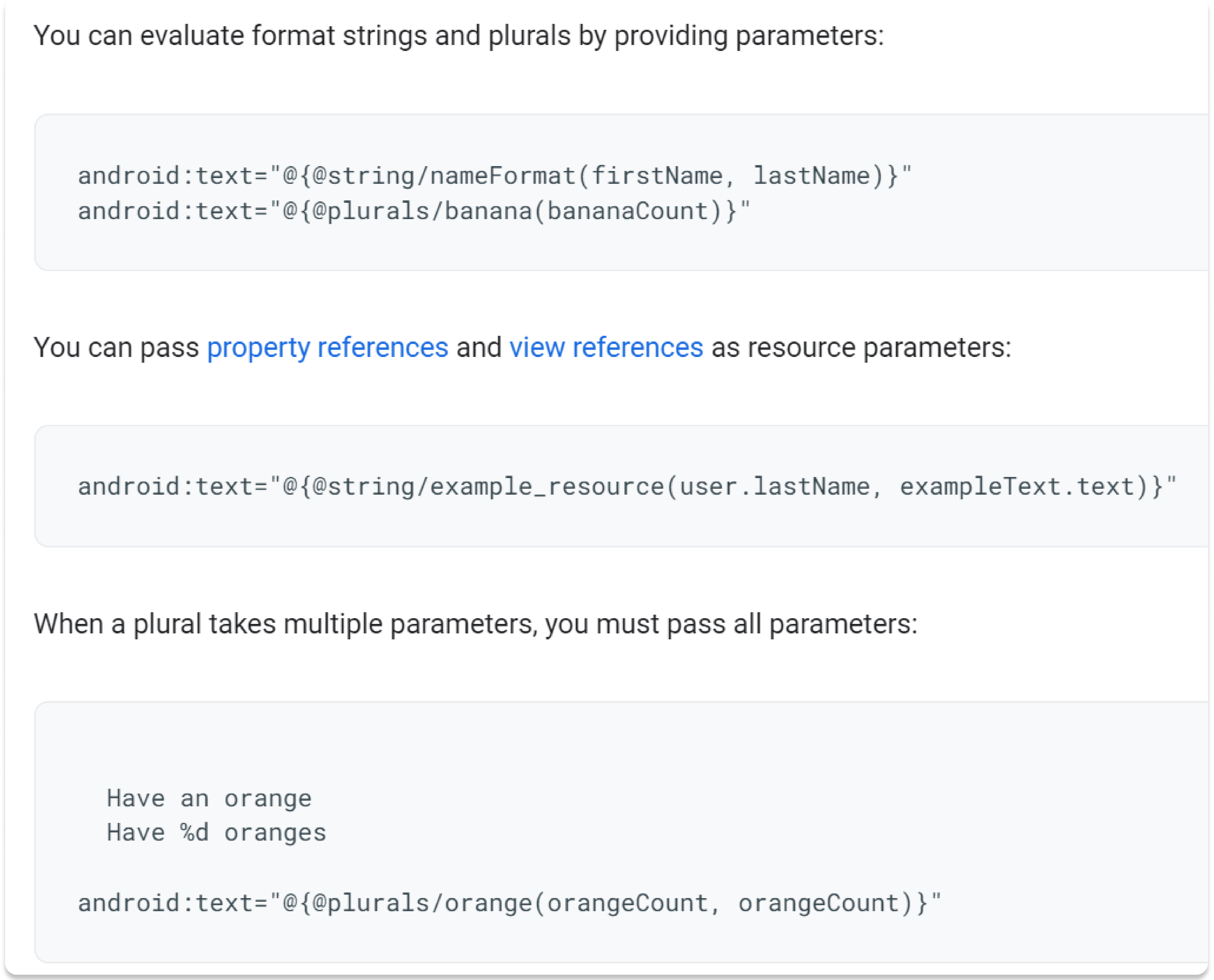


```xml
<ImageView    You, Moments ago • Uncommitted
    android:id="@+id/iv_ic_launch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="@dimen/arctic_fox_test"
    android:src="@drawable/ic_launcher_background"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/view_line2" />
```



DataBinding Expression Language

Resources(2)



You can evaluate format strings and plurals by providing parameters:

```
android:text="@{@string/nameFormat(firstName, lastName)}"
android:text="@{@plurals/banana(bananaCount)}"
```

You can pass property references and view references as resource parameters:

```
android:text="@{@string/example_resource(user.lastName, exampleText.text)}"
```

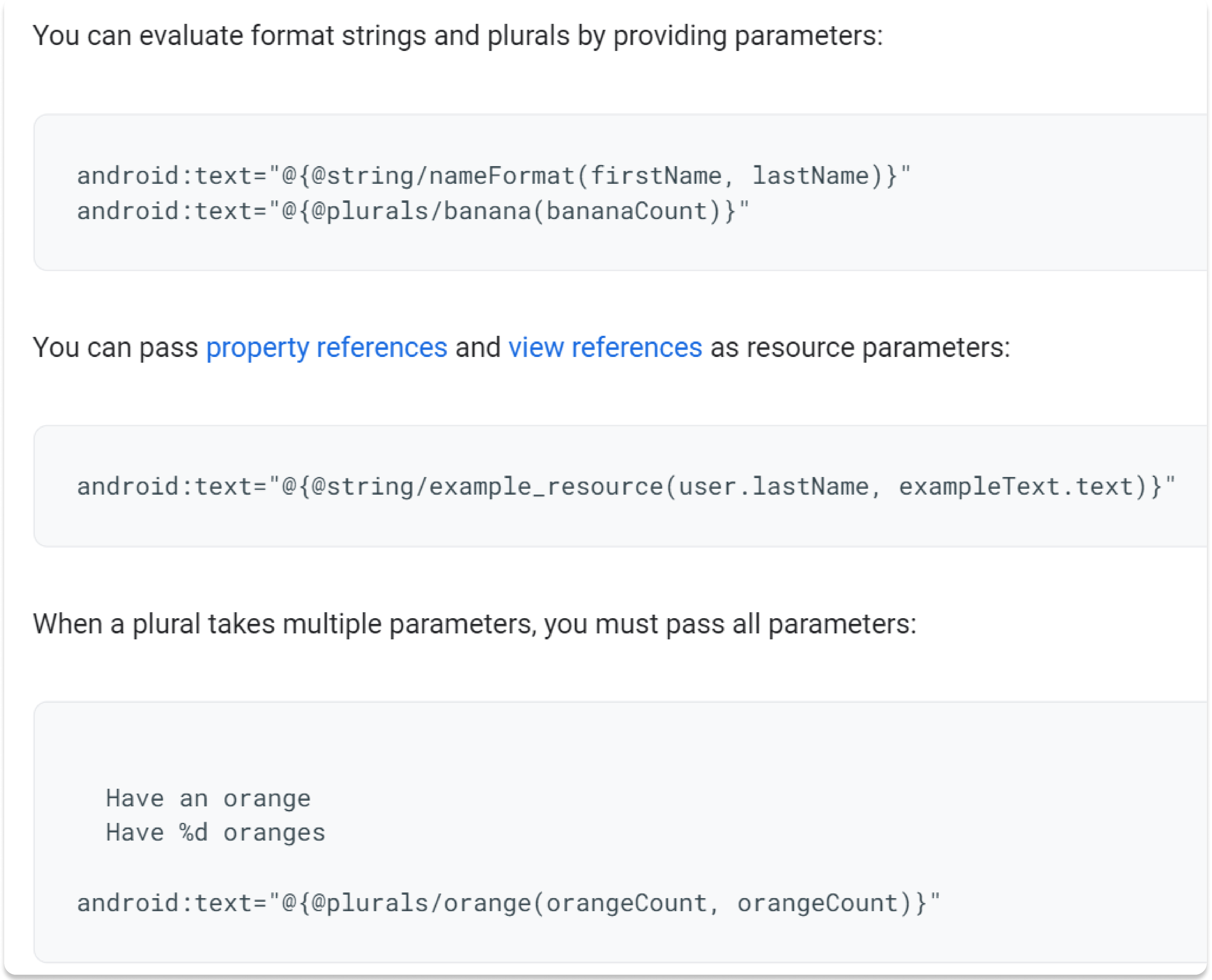
When a plural takes multiple parameters, you must pass all parameters:

```


  Have an orange
  Have %d oranges

android:text="@{@plurals/orange(orangeCount, orangeCount)}"
```

DataBinding - Architecture Component

DataBinding Expression Language

Resources(2)



You can evaluate format strings and plurals by providing parameters:

```
android:text="@{@string/nameFormat(firstName, lastName)}"
android:text="@{@plurals/banana(bananaCount)}"
```

You can pass property references and view references as resource parameters:

```
android:text="@{@string/example_resource(user.lastName, exampleText.text)}"
```

When a plural takes multiple parameters, you must pass all parameters:

```

  Have an orange
  Have %d oranges

android:text="@{@plurals/orange(orangeCount, orangeCount)}"
```

오빠들 갑자기 이해하기 귀찮아진거 다 알고있음ㅋㅋ😏😏

여기서 챙겨가야 하는 point는 2개임! (dataBinding은 거들뿐..)

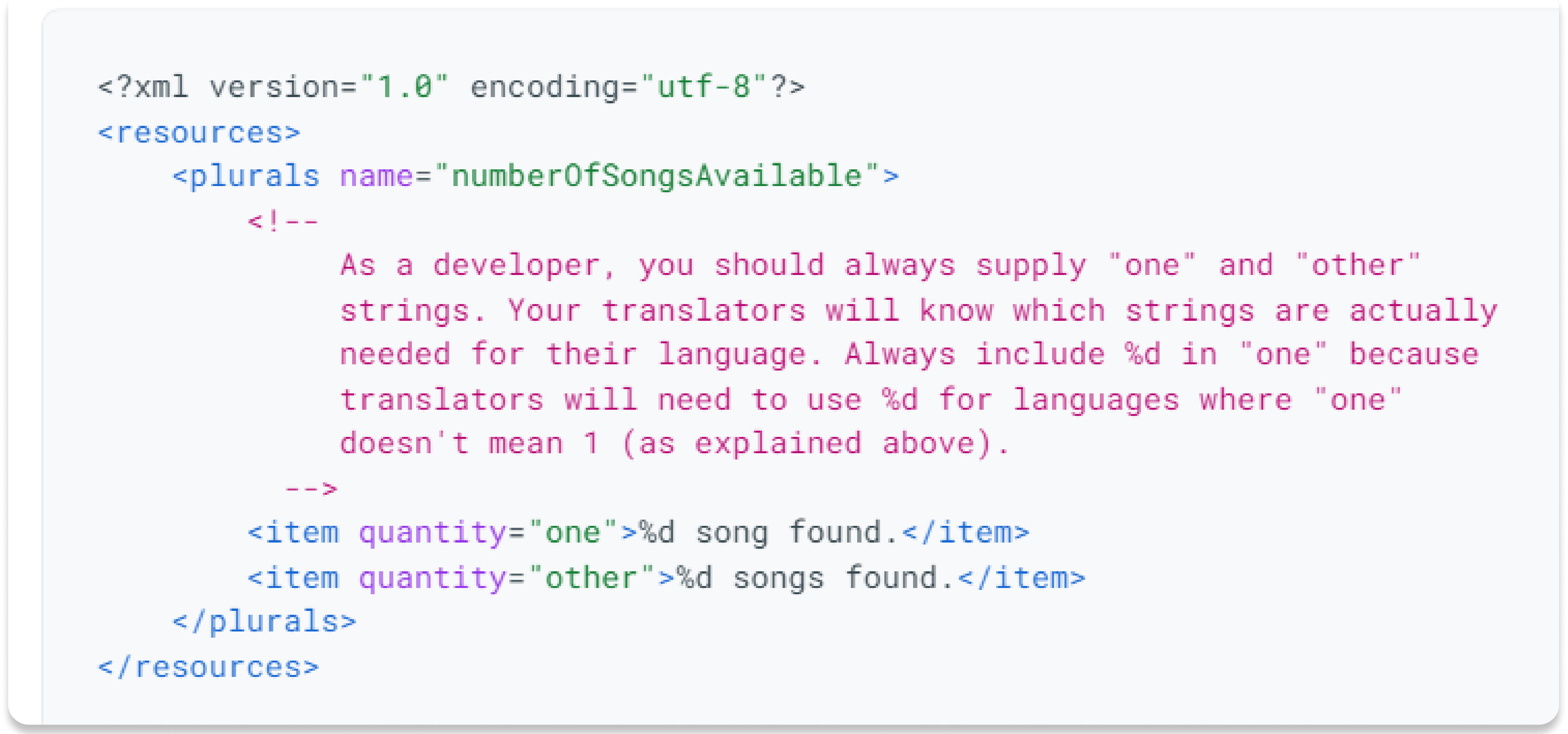
1. String에 **formant** 지정하는 법 (C언어 같음)
2. **Plurals** 사용하는 법

📚 **String Foramt (Android Developer)**

https://developer.android.com/guide/topics/resources/string-resource#formatting-strings

%1$s is a string and %2$d is a decimal number

📚 **Plurals**



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <!--
             As a developer, you should always supply "one" and "other"
             strings. Your translators will know which strings are actually
             needed for their language. Always include %d in "one" because
             translators will need to use %d for languages where "one"
             doesn't mean 1 (as explained above).
          -->
        <item quantity="one">%d song found.</item>
        <item quantity="other">%d songs found.</item>
    </plurals>
</resources>
```

DataBinding Expression Language

아 윌 비 백...