

Leveraging Android Databinding with Kotlin

What is Data Binding?



Data Binding

```
class RepoActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_repo)  
    }  
  
    override fun onResume() {  
        super.onResume()  
    }  
}
```

Data Binding

```
class RepoActivity : AppCompatActivity() {

    private lateinit var binding: ActivityRepoBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_repo)
    }

    override fun onResume() {
        super.onResume()
    }
}
```

Data Binding

```
internal class RepoActivity : AppCompatActivity() {

    private lateinit var binding: ActivityRepoBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_repo)
        binding.setLifecycleOwner(this)
    }

    override fun onResume() {
        super.onResume()
    }
}
```

Data Binding

```
internal class Repoactivity : AppCompatActivity() {

    private lateinit var binding: ActivityRepoBinding
    private lateinit var service: RepoService

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_repo)
        binding.setLifecycleOwner(this)

        service = RepoServiceFactory().get()
    }

    override fun onResume() {
        super.onResume()
        service.fetch { binding.items = it }
    }
}
```

Data Binding

```
class RepoActivity : AppCompatActivity() {

    private lateinit var binding: ActivityRepoBinding
    private lateinit var service: RepoService

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_home)
        binding.setLifecycleOwner(this)

        service = RepoServiceFactory().get()
    }

    override fun onResume() {
        super.onResume()
        service.fetch { binding.items = it }
    }
}
```

Delegated Properties

Delegated Properties

```
val value: String by lazy {  
    println("computed!")  
    "Hello"  
}
```

Delegated Properties

```
interface ReadOnlyProperty<in R, out T> {  
    operator fun getValue(thisRef: R, property: KProperty<*>): T  
}
```

```
interface ReadWriteProperty<in R, T> {  
    operator fun getValue(thisRef: R, property: KProperty<*>): T  
    operator fun setValue(thisRef: R, property: KProperty<*>, value: T)  
}
```

Delegated Properties

```
class ActivityBindingProperty<out T : ViewDataBinding>(  
    ) : ReadOnlyProperty<Activity, T> {  
  
    override operator fun getValue(thisRef: Activity, property: KProperty<*>): T {  
        TODO("not implemented")  
    }  
}
```

Delegated Properties

```
class ActivityBindingProperty<out T : ViewDataBinding>(  
    @LayoutRes private val resId: Int  
) : ReadOnlyProperty<Activity, T> {  
  
    private var binding: T? = null  
  
    override operator fun getValue(thisRef: Activity, property: KProperty<*>): T {  
        return binding ?: createBinding(thisRef).also { binding = it }  
    }  
  
    private fun createBinding(activity: Activity): T {  
        return DataBindingUtil.setContentView(activity, resId)  
    }  
}
```

ProTip: Extension Function! 🤘

```
class RepoActivity : AppCompatActivity() {

    private val binding by activityBinding<ActivityRepoBinding>(R.layout.activity_repo)

    private lateinit var service: RepoService

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding.setLifecycleOwner(this)

        service = RepoServiceFactory().get()
    }

    override fun onResume() {
        super.onResume()
        service.fetch { binding.items = it }
    }
}

fun <T : ViewDataBinding> FragmentActivity.activityBinding(
    @LayoutRes resId: Int
) = ActivityBindingProperty(resId)
```

Introducing

Android Architecture Components



Model-View-ViewModel

```
class RepoActivity : AppCompatActivity() {  
  
    private val binding by activityBinding<ActivityRepoBinding>(R.layout.activity_repo)  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        binding.lifecycleOwner(this)  
    }  
}
```

Model-View-ViewModel

```
class RepoActivity : AppCompatActivity() {

    private val binding by activityBinding<ActivityRepoBinding>(R.layout.activity_repo)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding.lifecycleOwner(this)
        binding.model = ViewModelProviders
            .of(this, RepoViewModelFactory())
            .get(RepoViewModel::class.java)
    }
}
```

ProTip: Extension Function! 🤘

```
class RepoActivity : AppCompatActivity() {

    private val binding by activityBinding<ActivityRepoBinding>(R.layout.activity_repo)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding.setLifecycleOwner(this)
        binding.model = getViewModel(RepoViewModelFactory())
    }
}

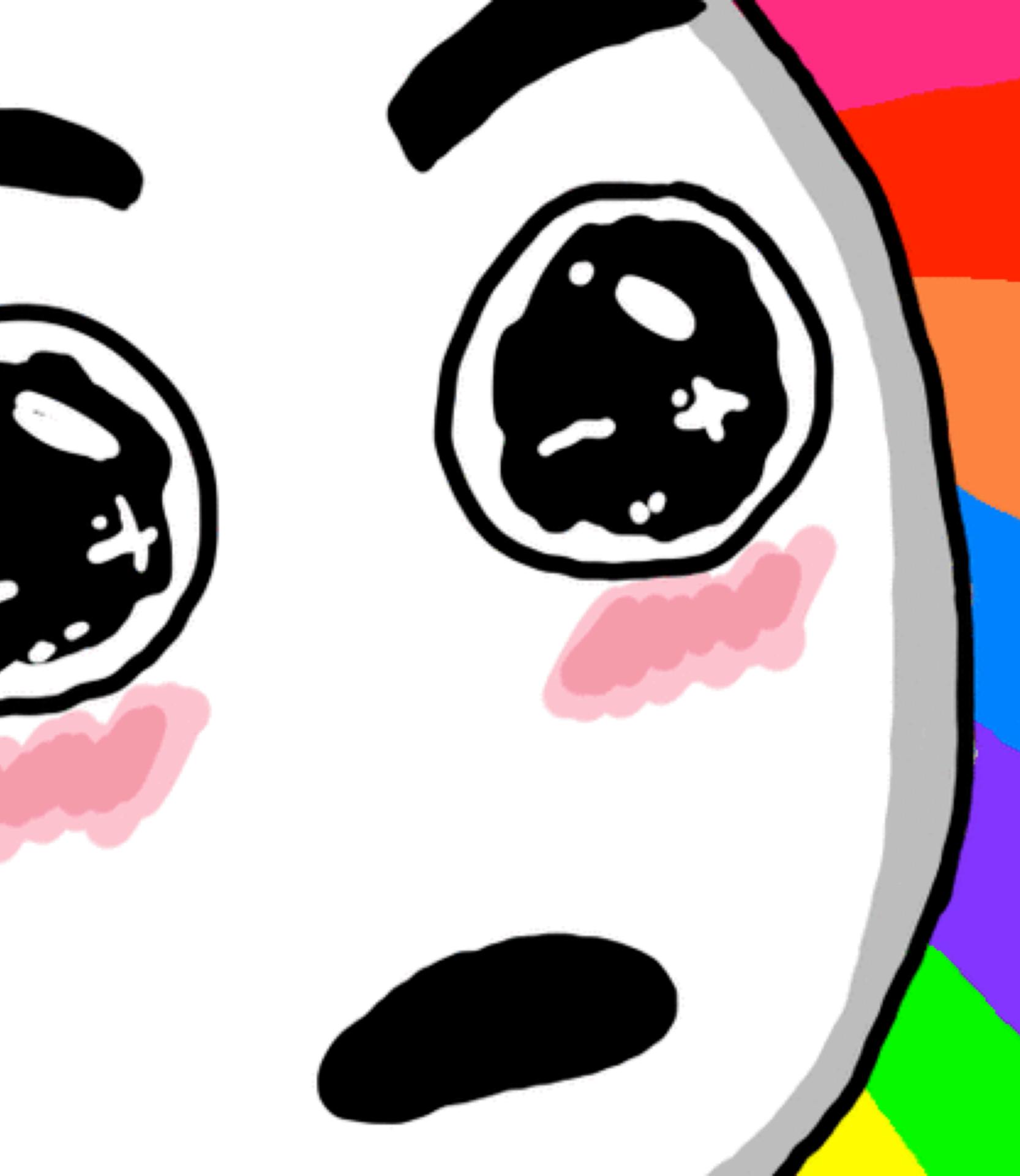
inline fun <reified T : ViewModel> FragmentActivity.getViewModel(
    factory: ViewModelProvider.Factory = ViewModelProvider.NewInstanceFactory()
) = ViewModelProviders.of(this, factory).get(T::class.java)
```

ViewModel

```
class RepoViewModel(service: RepoService) : ViewModel {  
  
    val items: ObservableField<List<String>> = ObservableField()  
  
    fun query(value: String) {  
        service.fetch(value, items::set)  
    }  
}
```

Observable

```
class RepoViewModel(service: RepoService) : ViewModel {  
  
    val items: ObservableField<List<String>> = ObservableField()  
  
    fun query(value: String) {  
        service.fetch(value, items::set)  
    }  
}
```



LiveData Data Binding

LiveData Data Binding

```
class RepoViewModel(service: RepoService) : ViewModel {  
  
    val items = MutableLiveData<List<String>>()  
  
    fun query(value: String) {  
        service.fetch(value, items::set)  
    }  
}
```

ProTip: Extension Functions!

```
class RepoViewModel(service: RepoService) : ViewModel {  
  
    val items = mutableLiveData<List<String>>()  
  
    fun query(value: String) {  
        service.fetch(value, items::setValue)  
    }  
}  
  
fun <T> ViewModel.mutableLiveDataOf() = MutableLiveData<T>()
```

ViewModel

```
class RepoViewModel(private val service: RepoService) : ViewModel {  
  
    val items = MutableLiveData<List<String>>()  
    val loading = MutableLiveData<Boolean>()  
  
    fun query(value: String) {  
        loading.value = true  
  
        service.fetch(value) {  
            loading.value = false  
            items.value = it  
        }  
    }  
}
```

Layout Expressions

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>

        <import type="android.view.View"/>

        <variable
            name="model"
            type="io.ashdavies.databinding.RepoViewModel"/>

    </data>

    <ProgressBar...
        android:visibility="@{model.loading ? View.VISIBLE : View.GONE}" />

</layout>
```

Layout Expressions

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>

        <import type="android.view.View"/>

        <variable
            name="model"
            type="io.ashdavies.databinding.RepoViewModel"/>

    </data>

    <TextView...
        android:text="@string/activity_repo_empty"/>

    <ProgressBar...
        android:visibility="@{model.loading ? View.VISIBLE : View.GONE}"/>

</layout>
```

Layout Expressions

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>

        <import type="android.view.View"/>

        <variable
            name="model"
            type="io.ashdavies.databinding.RepoViewModel"/>

    </data>

    <TextView...
        android:text="@string/activity_repo_empty"
        android:visibility="@{model.items.count == 0 ? View.VISIBLE : View.GONE}"/>

    <ProgressBar...
        android:visibility="@{model.loading ? View.VISIBLE : View.GONE}"/>

</layout>
```

Layout Expressions

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>

        <import type="android.view.View"/>

        <variable
            name="model"
            type="io.ashdavies.databinding.RepoViewModel"/>

    </data>

    <TextView...
        android:text="@string/activity_repo_empty"
        android:visibility="@{model.items.count == 0 && !model.loading ? View.VISIBLE : View.GONE}"/>

    <ProgressBar...
        android:visibility="@{model.loading ? View.VISIBLE : View.GONE}"/>

</layout>
```



Too Complicated!

```
class RepoViewModel(private val service: RepoService) : ViewModel {

    val items = MutableLiveData<List<String>>()
    val loading = MutableLiveData<Boolean>()
    val empty = MutableLiveData<Boolean>()

    fun query(value: String) {
        loading.value = true
        empty.value = true

        service.fetch(value) {
            items.value = it
            empty.value = it.isEmpty()
            loading.value = false
        }
    }
}
```

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>

        <import type="android.view.View"/>

        <variable
            name="model"
            type="io.ashdavies.databinding.RepoViewModel"/>

    </data>

    <TextView...
        android:text="@string/activity_repo_empty"
        android:visibility="@{model.empty ? View.VISIBLE : View.GONE}"/>

    <ProgressBar...
        android:visibility="@{model.loading ? View.VISIBLE : View.GONE}"/>

</layout>
```

@BindingAdapter

@BindingAdapter

```
@BindingAdapter("goneUnless")
fun goneUnless(view: View, visible: Boolean) {
    view.visibility = if (visible) View.VISIBLE else View.GONE
}
```

ProTip: Extension Property!

```
@set:BindingAdapter("isVisible")
inline var View.isVisible: Boolean
    get() = visibility == View.VISIBLE
    set(value) {
        visibility = if (value) View.VISIBLE else View.GONE
    }
```

@BindingAdapter

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>

        <import type="android.view.View"/>

        <variable
            name="model"
            type="io.ashdavies.databinding.RepoViewModel"/>

    </data>

    <TextView...
        android:text="@string/activity_repo_empty"
        app:isVisible="@{model.empty}"/>

    <ProgressBar...
        app:isVisible="@{model.loading}"/>

</layout>
```



@Bindable

@Bindable

ObservableViewModel

@Bindable

PropertyChangeRegistry

@Bindable

```
abstract class BaseObservableViewModel : ViewModel(), Observable {

    private val callbacks: PropertyChangeRegistry = PropertyChangeRegistry()

    override fun addOnPropertyChangedCallback(callback: Observable.OnPropertyChangedCallback) {
        callbacks.add(callback)
    }

    override fun removeOnPropertyChangedCallback(callback: Observable.OnPropertyChangedCallback) {
        callbacks.remove(callback)
    }

    fun notifyChange() {
        callbacks.notifyCallbacks(this, 0, null)
    }

    fun notifyPropertyChanged(fieldId: Int) {
        callbacks.notifyCallbacks(this, fieldId, null)
    }
}
```

@Bindable

```
class RepoViewModel(private val service: RepoService) : ObservableViewModel {  
  
    val items = mutableLiveDataOf<List<String>>()  
    val loading = mutableLiveDataOf<Boolean>()  
  
    fun query(value: String) {  
        loading.value = true  
  
        service.fetch(value) {  
            loading.value = false  
            items.value = it  
        }  
    }  
}
```

@Bindable

```
class RepoViewModel(private val service: RepoService) : ObservableViewModel {

    @get:Bindable
    var items: List<String> = emptyList()
        private set(value) {
            notifyPropertyChanged(BR.items)
            field = value
        }

    val loading = MutableLiveData<Boolean>()

    fun query(value: String) {
        loading.value = true

        service.fetch(value) {
            loading.value = false
            items = it
        }
    }
}
```



@Bindable



Delegated Properties

@Bindable

```
class BindableProperty<T>(  
    initial: T, private val observable: ObservableViewModel, private val id: Int  
) : ObservableProperty<T>(initial) {  
  
    override fun beforeChange(  
        property: KProperty<*>, oldValue: T, newValue: T  
    ): Boolean = { /* ... */ }  
  
    override fun afterChange(  
        property: KProperty<*>, oldValue: T, newValue: T  
    ) { /* ... */ }  
}
```

@Bindable

```
class BindableProperty<T>(  
    initial: T, private val observable: ObservableViewModel, private val id: Int  
) : ObservableProperty<T>(initial) {  
  
    override fun beforeChange(  
        property: KProperty<*>, oldValue: T, newValue: T  
    ): Boolean = oldValue != newValue  
  
    override fun afterChange(  
        property: KProperty<*>, oldValue: T, newValue: T  
    ) { /* ... */ }  
}
```

@Bindable

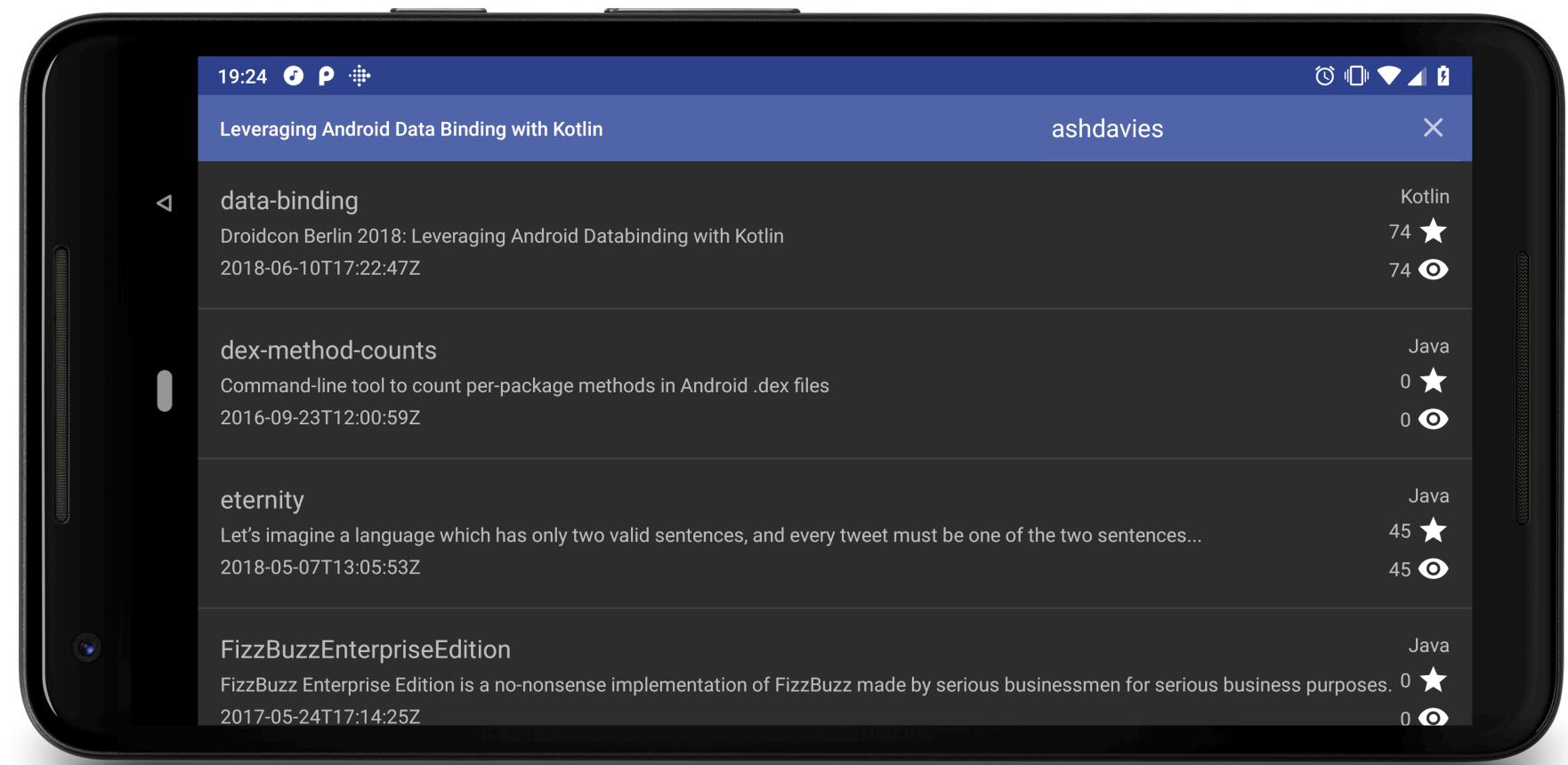
```
class BindableProperty<T>(  
    initial: T, private val observable: ObservableViewModel, private val id: Int  
) : ObservableProperty<T>(initial) {  
  
    override fun beforeChange(  
        property: KProperty<*>, oldValue: T, newValue: T  
    ): Boolean = oldValue != newValue  
  
    override fun afterChange(property: KProperty<*>, oldValue: T, newValue: T) {  
        observable.notifyPropertyChanged(id)  
    }  
}
```

@Bindable

```
class RepoViewModel(private val service: RepoService) : ObservableViewModel() {  
  
    @get:Bindable  
    var items by bindable<List<String>>(emptyList(), BR.items)  
        private set  
  
    val items = mutableLifeDataOf<List<String>>()  
  
    init {  
        service.fetch(items::setValue)  
    }  
}  
  
fun <T> ObservableViewModel.bindable(initial: T, id: Int): BindableProperty<T> {  
    return BindableProperty(initial, this, id)  
}
```

Data Binding Sample

github.com/ashdavies/data-binding



Android Data Binding with Kotlin



bit.ly/2yU8qUz

Cheers! 🍻



_ashdavies