Data binding

in a Kotlin world

Lisa Wray
@lisawrayz

Data binding

in a Kotlin world

Lisa Wray
@lisawrayz

Data binding

in a Kotlin world

Lisa Wray
@lisawrayz

Less code is better code

1. Quick tour of The Best Parts™ of data binding
2. Kotlin tips and tricks
3. High level advice

What is data binding?

Declarative, functional code

- Code generation + Android's XML layouts
- A "glue" layer replacing the boilerplate of connecting views and models
- Generates Java code, but works with Kotlin today

Data binding

Works fine with:

- Dagger
- RxJava
- Kotlin
- Architecture Components
- Butterknife, Kotlin Android Extensions (really)

Doesn’t work with:

- Anko
- Litho

Getting started

Now with more Kotlin

build.gradle (module)

```
android {
    databinding {
        enabled = true;
    }
}
```

@lisawrayz

Kotlin

build.gradle (module)

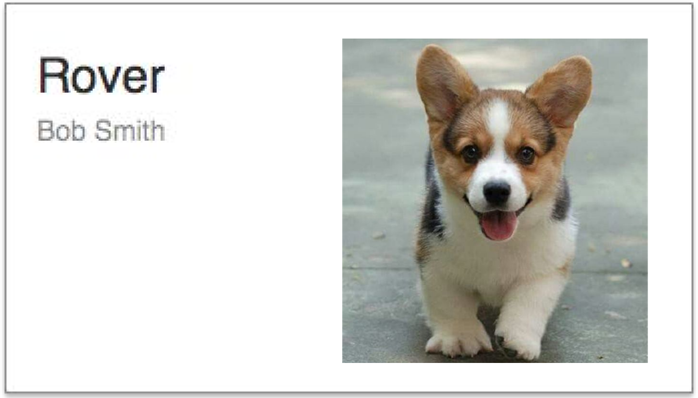
```
apply plugin: 'kotlin-kapt'

dependencies {
    kapt "com.android.databinding:compiler:$tools_version"
}
```

Just the basics

Example: view a dog

Normally:
3 findViewById()
3 setter calls


Rover
Bob Smith

Dog.kt

```kotlin
class Dog(val dogName: String, val ownerName: String,
    val imageUrl: String)
```

@lisawrayz

activity_dog.xml

```
<layout>
   <data>
       <variable
           name="dog"
           type="com.xwray.doglist.model.Dog" />
   </data>

   <android.support.constraint.ConstraintLayout ...>
       <ImageView
           app:image="@{dog.imageUrl}" />

       <TextView ..
           android:text="@{dog.dogName}"
           tools:text="Name"  />

       <TextView
           android:text="@{dog.ownerName}"
           tools:text="Owner name"  />
   </android.support.constraint.ConstraintLayout>
</layout>
```

@lisawrayz

activity_dog.xml

```xml
<layout>
   <data>
       <variable
           name="dog"
           type="com.xwray.doglist.model.Dog"  />
   </data>

   <android.support.constraint.ConstraintLayout ...>
       <ImageView
           app:image="@{dog.imageUrl}"  />

       <TextView ..
           android:text="@{dog.dogName}"
           tools:text="Name"  />

       <TextView
           android:text="@{dog.ownerName}"
           tools:text="Owner name"  />
   </android.support.constraint.ConstraintLayout>
</layout>
```

@lisawrayz

activity_dog.xml

```
<layout>
   <data>
       <variable
           name="dog"
           type="com.xwray.doglist.model.Dog"  />
   </data>

   <android.support.constraint.ConstraintLayout ...>
       <ImageView
           app:image="@{dog.imageUrl}"  />

       <TextView ..
           android:text="@{dog.dogName}"
           tools:text="Name"  />

       <TextView
           android:text="@{dog.ownerName}"
           tools:text="Owner name"  />
   </android.support.constraint.ConstraintLayout>
</layout>
```

@lisawrayz

DogActivity.kt

```
class DogActivity : BaseActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = DataBindingUtil.setContentView<ActivityDogBinding>(this,
          R.layout.activity_dog)

        binding.dog = myDog
    }
}
```

@lisawrayz

DogActivity.kt

```
class DogActivity : BaseActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = DataBindingUtil.setContentView<ActivityDogBinding>(this,
          R.layout.activity_dog)

        binding.dog = myDog
    }
}
```

You can still have ids, if you want to

```
<layout>
   <android.support.constraint.ConstraintLayout ...>
       <ImageView
           android:id="@+id/image"
           app:image="@{dog.imageUrl}"  />

       <TextView
           android:id="@+id/name"
           android:text="@{dog.dogName}"
           tools:text="Name"  />

       <TextView
           android:id="@+id/ownerName"
           android:text="@{dog.ownerName}"
           tools:text="Owner name"  />
   </android.support.constraint.ConstraintLayout>
</layout>
```

@lisawrayz

You can still have ids, if you want to

```
<layout>
   <android.support.constraint.ConstraintLayout ...>
       <ImageView
           android:id="@+id/image"
           app:image="@{dog.imageUrl}"  />

       <TextView
           android:id="@+id/name"
           android:text="@{dog.dogName}"
           tools:text="Name"  />

       <TextView
           android:id="@+id/owner_name"
           android:text="@{dog.ownerName}"
           tools:text="Owner name"  />
   </android.support.constraint.ConstraintLayout>
</layout>
```

You can still have ids, if you want to

Ids generated in camelCase

```
binding.name
binding.ownerName
binding.image
```

DogActivity.kt

```
class DogActivity : BaseActivity() {

    var binding: ActivityDogBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView<ActivityDogBinding>(this,
          R.layout.activity_dog)

        binding.dog = myDog
    }
}
```

@lisawrayz

DogActivity.kt

```kotlin
class DogActivity : BaseActivity() {

    var binding: ActivityDogBinding? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView<ActivityDogBinding>(this,
          R.layout.activity_dog)

        binding?.dog = myDog
    }
}
```

@lisawrayz

DogActivity.kt

```kotlin
class DogActivity : BaseActivity() {

    lateinit var binding: ActivityDogBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView<ActivityDogBinding>(this,
          R.layout.activity_dog)

        binding.dog = myDog
    }
}
```

@lisawrayz

DogActivity.kt

```
class DogActivity : BaseActivity() {

   val binding: ActivityDogBinding by lazy {
       DataBindingUtil.setContentView<ActivityDogBinding>(this,
            R.layout.activity_dog)
   }

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

       binding.dog = myDog
   }
}
```

@lisawrayz

DogActivity.kt

```
class DogActivity : BaseActivity() {

   val binding: ActivityDogBinding by SetContentView(R.layout.activity_dog)

}
```

Delegates.kt

```
class SetContentView<in R : Activity, out T : ViewDataBinding>(
        @LayoutRes private val layoutRes: Int) {

    operator fun getValue(thisRef: Activity, property: KProperty<*>): T {
        return DataBindingUtil.setContentView<T>(thisRef, layoutRes)
    }

}
```

@lisawrayz

DogActivity.kt

```kotlin
val binding: ActivityDogBinding by SetContentView(R.layout.activity_dog)
```

Delegates.kt

```kotlin
class SetContentView<in R : Activity, out T : ViewDataBinding>(
       @LayoutRes private val layoutRes: Int) {

   private var value : T? = null

   operator fun getValue(thisRef: Activity, property: KProperty<*>): T {

       value = value ?: DataBindingUtil.setContentView<T>(thisRef, layoutRes)
       return value
   }
}
```

@lisawrayz

DogActivity.kt

```
val binding: ActivityDogBinding by contentView(R.layout.activity_dog)
```

Delegates.kt

```
fun <R : Activity, T : ViewDataBinding> contentView(@LayoutRes layoutRes: Int):
        SetContentView<R, T> {

    return SetContentView(layoutRes)

}
```

@lisawrayz

Multiple binding variables

```
<layout>
   <data>
       <variable
           name="dog"
           type="com.xwray.doglist.model.Dog" />
       <variable
           name="cat"
           type="com.xwray.doglist.model.Cat" />
       <variable
           name="subscription"
           type="com.xwray.doglist.model.Subscription" />
   </data>
...
```

@lisawrayz

Multiple binding variables

```
binding.dog = myDog
binding.cat = myCat
binding.subscription = Subscription.SUBSCRIBED
```

@lisawrayz

Multiple binding variables

```
binding.apply {
    dog = aDog
    cat = currentUser
    subscription = Subscription.SUBSCRIBED
}
```

Property references

If you like Kotlin, you may like this

```
android:text="@{dog.owner.name}"
```

@lisawrayz

NPE safety

Defaults (not just safe calls)

`@{dog.name}` `(String)` → null

`@{dog.age}` `(int)` → 0

```
dog?.nam        t {
    textV        xt = it
}
```

```
var name: String = null;
if (dog != null && dog.name != null) {
    name = dog.name
}
textView.text = name
```

@lisawrayz

New XML attributes

```
app:image="@{dog.imageUrl}"
```

- Automatic setters
- Renamed setters (provided!)
- Custom bindings

Automatic setters in XML

No attribute needed -- only a public setter

```
<android.support.v4.widget.DrawerLayout
    app:scrimColor="@{@color/scrim}"
    app:drawerListener="@{fragment.drawerListener}"/>
```

$$\texttt{scrimColor} \xrightarrow{\qquad\qquad} \texttt{setScrimColor(resolvedColor)}$$

@lisawrayz

Renamed setters in XML

Already implemented for Android framework

```
@BindingMethods({
        @BindingMethod(type = "android.widget.ImageView",
                       attribute = "android:tint",
                       method = "setImageTintList"),
})
```

`tint` $\longrightarrow$ `setImageTintList()`

@lisawrayz

Custom binding adapters

My favorite!

```
<ImageView
    android:id="@+id/image"
    app:image="@{dog.imageUrl}" />
```

@lisawrayz

Custom binding adapters

```
@BindingAdapter("image")
fun loadImage(imageView: ImageView, imageUrl: String?) {
   Picasso.with(imageView.context).load(imageUrl)
               .fit()
               .centerCrop()
               .into(imageView)
}
```

Custom binding adapters

```
class BindingAdapters {
   companion object {
       @JvmStatic @BindingAdapter("image")
       fun loadImage(imageView: ImageView, url: String?) {
           // load image here
       }
   }
}
```

@lisawrayz

Resources in expressions

Aka, no more @dimen/a_plus_b

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

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

android:text="@{@plurals/banana(bananaCount)}"

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

Expression language

But … don't use it!

- Mathematical `+ - / * %`
- String concatenation `+`
- Logical `&& ||`
- Binary `& | ^`
- Unary `+ - ! ~`
- Shift `>> >>> <<`
- Comparison `== > < >= <=`
- `instanceof`
- Grouping `()`

- Literals - character, String, numeric, `null`
- Cast
- Method calls
- Field access
- Array access `[]`
- Ternary operator `?:`
- `Null coalescing operator ??`

@lisawrayz

Bind a ViewModel

Don’t put business logic in your XML

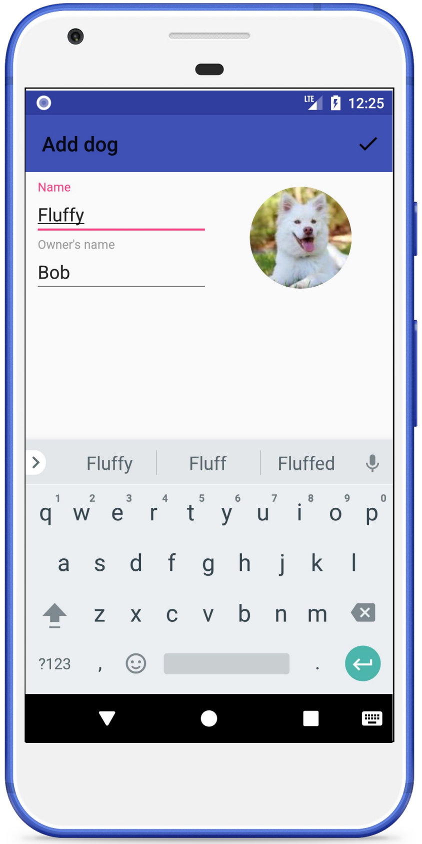
- Code in XML is ugly & can’t be tested
- Write Kotlin, not data binding expressions
- ViewModel + unit tests = ❤️

Observable data

The real power of data binding

Observable ViewModel

Example: Add a dog



@lisawrayz

Observable fields

- ObservableField
- ObservableBoolean
- ObservableByte
- ObservableChar
- ObservableShort
- ObservableInt
- ObservableLong
- ObservableFloat
- ObservableDouble
- ObservableParcelable

```
private class Dog {
   val name = ObservableField<String>()
   val ownerName = ObservableField<String>()
   val age = ObservableInt()
}

dog.name.set("Fluffy")
dog.name.get()

android:text=”@{dog.name}”
```

@lisawrayz

Observable models (BaseObservable)

```
class AddDogViewModel : BaseObservable() {

    var dogName = ""
        @Bindable get
        set(value) {
            if (field != value) {
                field = value
                notifyPropertyChanged(BR.dogName)
            }
        }
}
```

@lisawrayz

Observable models (BaseObservable)

```
class AddDogViewModel : BaseObservable() {

   var dogName = ""
       @Bindable get
       set(value) {
           if (field != value) {
               field = value
               notifyPropertyChanged(BR.dogName)
           }
       }
}
```

@lisawrayz

Observable models (BaseObservable)

```
class AddDogViewModel : BaseObservable() {

   var dogName = ""
       @Bindable get
       set(value) {
           if (field != value) {
               field = value
               notifyPropertyChanged(BR.dogName)
           }
       }
}
```

@lisawrayz

Observable models

```
class AddDogViewModel : BaseObservable() {

   var dogName: String by Delegates.observable("") {
           prop, old, new ->
       notifyPropertyChanged(BR.dogName)
   }
       @Bindable get

}
```

@lisawrayz

Observable models

```
class AddDogViewModel : BaseObservable() {

    @get:Bindable
    var dogName: String by Delegates.observable("") {
            prop, old, new ->
        notifyPropertyChanged(BR.dogName)
    }
}
```

@lisawrayz

AddDogViewModel.kt

```kotlin
class AddDogViewModel : BaseObservable() {

    @get:Bindable
    var dogName by BindableDelegate("", BR.dogName)

}
```

Delegates.kt

```kotlin
class BindableDelegate<in R : BaseObservable, T : Any>(private var value: T,
        private val bindingRes: Int) {

    operator fun getValue(thisRef: R, property: KProperty<*>): T = value

    operator fun setValue(thisRef: R, property: KProperty<*>, value: T) {
        this.value = value
        thisRef.notifyPropertyChanged(bindingRes)
    }
}
```

@lisawrayz

AddDogViewModel.kt

```kotlin
class AddDogViewModel : BaseObservable() {

   @get:Bindable
   var dogName by BindableDelegate("", BR.dogName)

}
```

Delegates.kt

```kotlin
class BindableDelegate<in R : BaseObservable, T : Any>(private var value: T,
       private val bindingEntry: Int) {

   operator fun getValue(thisRef: R, property: KProperty<*>): T = value

   operator fun setValue(thisRef: R, property: KProperty<*>, value: T) {
       this.value = value
       thisRef.notifyPropertyChanged(bindingEntry)
   }
}
```

@lisawrayz

AddDogViewModel.kt

```kotlin
class AddDogViewModel : BaseObservable() {

   @get:Bindable
   var dogName by bindable("", BR.dogName)

}
```

Delegates.kt

```kotlin
fun <R : BaseObservable, T : Any> bindable(value: T, bindingRes: Int):
       BindableDelegate<R, T> {
   return BindableDelegate(value, bindingRes)
}
```

@lisawrayz

AddDogViewModel.kt

```kotlin
class AddDogViewModel : BaseObservable() {

    @get:Bindable
    var dogName by bindString(BR.dogName)

}
```

Delegates.kt

```kotlin
fun <R : BaseObservable, String> bindString(value: String = "", bindingRes: Int):
       BindableDelegate<R, String> {
   return BindableDelegate(value, bindingRes)
}
```

@lisawrayz

Dependent properties

```kotlin
class AddDogViewModel : BaseObservable() {

    @get:Bindable
    var dogName by bindString(BR.dogName)

    val submitEnabled: Boolean
        @Bindable("dogName")
        get() = !dogName.isNullOrEmpty()

}
```

@lisawrayz

Two-way binding

add_dog_activity.xml

```
<android.support.design.widget.TextInputEditText
   android:hint="@string/dog_name"
   android:maxLines="1"
   android:text="@={viewModel.dogName}" />
```

add_dog_activity.xml

```xml
<android.support.design.widget.TextInputEditText
    android:hint="@string/dog_name"
    android:maxLines="1"
    android:text="@={viewModel.dogName}" />
```

@lisawrayz

add_dog_activity.xml

```
editText.addTextChangedListener(object : TextWatcher {
   override fun afterTextChanged(editable: Editable?) {
       viewModel.dogName = editable?.toString()
   }

   override fun beforeTextChanged(p0: CharSequence?,
       p1: Int, p2: Int, p3: Int) {

   }

   override fun onTextChanged(p0: CharSequence?,
       p1: Int, p2: Int, p3: Int) {

   }
})
```

@lisawrayz

Event listeners

Listener objects

Meh

```
android:onClickListener="@{callbacks.clickListener}"


android:onClick="@{callbacks.clickListener}"
```

Method references

```
<EditText
   android:afterTextChanged="@{callbacks::nameChanged}" .../>

public class Callbacks {
   public void nameChanged(Editable editable) {
       //...
   }
}
```

Lambda expressions

```
<EditText
  android:afterTextChanged="@{(e)->callbacks.textChanged(user, e)}"
  ... />

public class Callbacks {
  public void textChanged(User user, Editable editable) {
      //...
  }
}
```

Lambda expressions

```
<EditText
  android:afterTextChanged="@{()->callbacks.textChanged(user)}"
  ... />

public class Callbacks {
  public void textChanged(User user) {
      //...
  }
}
```

@lisawrayz

Method reference vs lambda

- Method references: evaluated at binding time
- Lambda expressions: evaluated when the event occurs
- (FYI) standard android:onClick() uses reflection

Where data binding shines

Partial updates to UI

- Data binding uses bitwise flags to mark fields “dirty”
- Only changed fields are re-bound

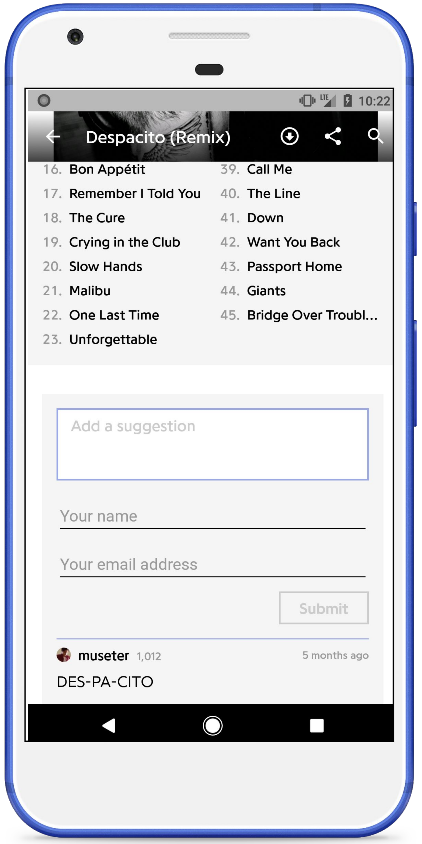
Related UI components

Ex: sign up forms, content input forms

- Components depend on each others’ state?
- Use a state machine to model view state?
- Consider data binding

Related components

Bonus: In a RecyclerView!


10:22

Despacito (Remix)

16. Bon Appétit
17. Remember I Told You
18. The Cure
19. Crying in the Club
20. Slow Hands
21. Malibu
22. One Last Time
23. Unforgettable
39. Call Me
40. The Line
41. Down
42. Want You Back
43. Passport Home
44. Giants
45. Bridge Over Troubl...

Add a suggestion

Your name

Your email address

Submit

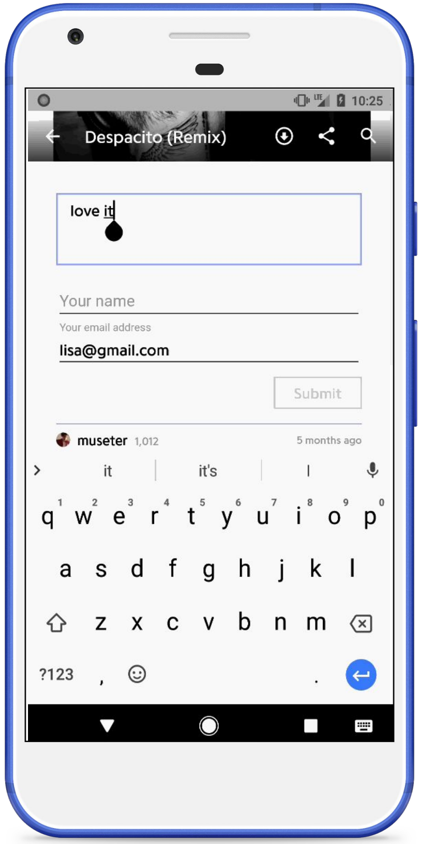
museter 1,012 5 months ago

DES-PA-CITO

@lisawrayz

Related components

Bonus: In a RecyclerView!


10:25

Despacito (Remix)

love it

Your name

Your email address

lisa@gmail.com

Submit

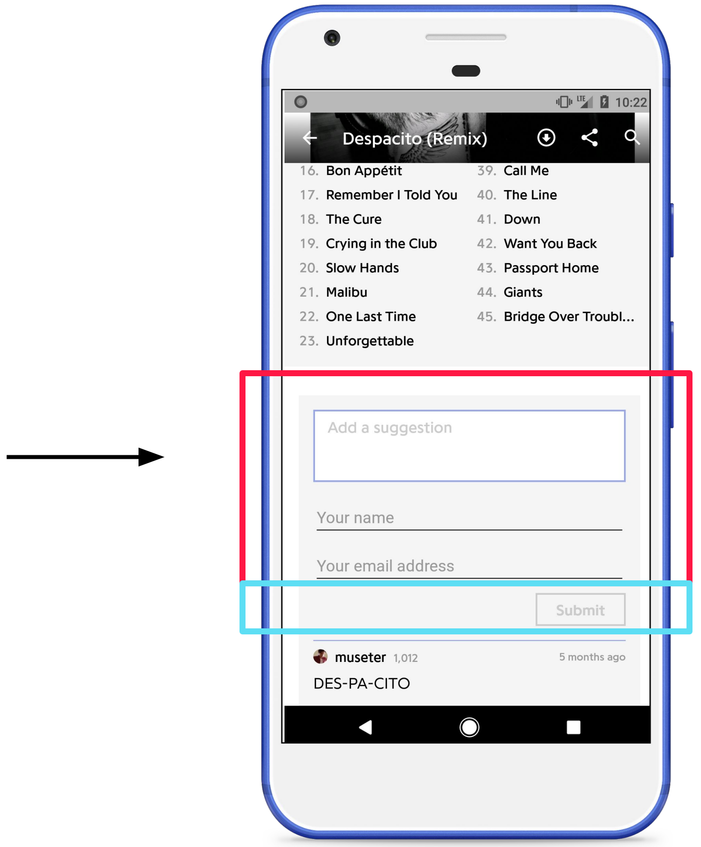
museter 1,012 5 months ago

@lisawrayz

Related components

Bonus: In a RecyclerView!

Two different RV items,
bound to the same ViewModel


10:22

Despacito (Remix)

16. Bon Appétit
17. Remember I Told You
18. The Cure
19. Crying in the Club
20. Slow Hands
21. Malibu
22. One Last Time
23. Unforgettable
39. Call Me
40. The Line
41. Down
42. Want You Back
43. Passport Home
44. Giants
45. Bridge Over Troubl...

museter 1,012 5 months ago

DES-PA-CITO

@lisawrayz

Encapsulation of view components

Alternative to custom views

```
<layout>
   <data>
       <variable name="dog" type="com.xwray.doglist.model.Dog"/>
       <variable name="secondDog" type="com.xwray.doglist.model.Dog"/>
   </data>
   <LinearLayout>
       <include layout="@layout/dog"
           bind:dog="@{dog}"/>
       <include layout="@layout/dog"
           bind:dog="@{secondDog}"/>
   </LinearLayout>
</layout>
```

@lisawrayz

Animations

Binding adapters

```
@BindingAdapter("animatedVisibility")
fun setVisibility(view: View, visibility: Int) {
    // animate here
}
```

Animations

View transitions

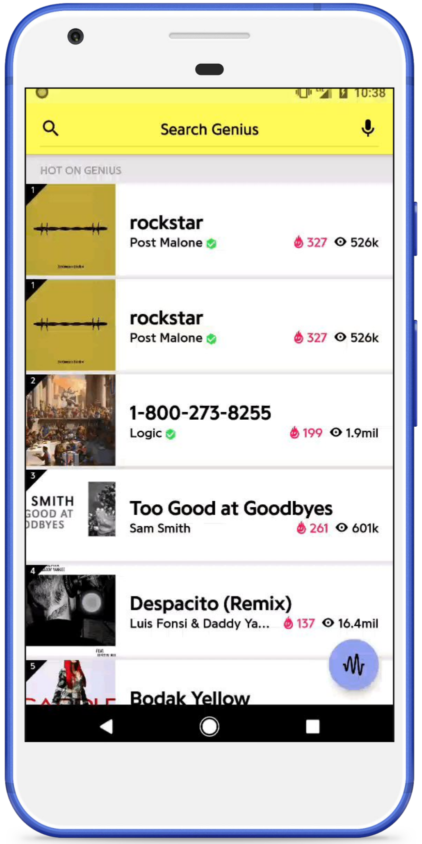
```
binding.addOnRebindCallback(object : OnRebindCallback<ViewDataBinding>() {
   override fun onPreBind(binding: ViewDataBinding?): Boolean {
       TransitionManager.beginDelayedTransition(
               binding.root as ViewGroup)
       return super.onPreBind(binding)
   }
})
```

George Mount:
https://medium.com/google-developers/android-data-binding-animations-55f6b5956a64

Animations

Activity or fragment transitions

```
android:transitionName="@{“song” + song.id}"
```



@lisawrayz

Plays well with others

RxJava

Can I use data binding with RxJava? Sure

`io.reactivex.Observable`   `android.databinding.Observable`

- Rx Observable is different from data binding Observable
- Rx will not handle partial updates for you
- Data binding does not deal with threading
- Data binding isn’t an event bus for UI events

@lisawrayz

RxJava

Can I use data binding with RxJava? Sure

`io.reactivex.Observable`   `android.databinding.Observable`

- Rx Observable is different from data binding Observable
- Rx will not handle partial updates for you
- Data binding does not deal with threading
- Data binding isn't an event bus for UI events

Other view binding frameworks

Can I use it with Butterknife? Sure

… with Kotlin Android Extensions?

Data binding is overkill *just* for view references
If you already use another view binding framework, it’s ok
No conflicts

@lisawrayz

A note: Kotlin Android Extensions

Performance issues in Views (not Activities)

- View lookups are cached in Activities and Fragments
- NOT cached elsewhere
  - RecyclerView ViewHolders
  - Custom views
  - Fixed in 1.1.4 but experimental

https://github.com/Kotlin/KEEP/blob/master/proposals/android-extensions-entity-caching.md

Dagger

Can I use data binding with Dagger? Sure

- Cascading errors: An error in one framework can cause the other to fail in annotation processing, printing tons of Dagger errors
    - Not unique to data binding
- Injection is possible (`DataBindingComponent` interface)

Jacob Tabak, Droidcon NYC 2016, "Advanced Data Binding in Practice"
https://www.youtube.com/watch?v=u8d_zXukB2w

Architecture Components

Can I use data binding with LiveData? Sure

- Can't extend both `BaseObservable` and `ViewModel`
- Alternatives:
  - Use `ObservableFields`
  - Implement `Observable` yourself
- "GithubBrowserSample": Arch Comp + Dagger2 + data binding

https://github.com/googlesamples/android-architecture-components/tree/ma
ster/GithubBrowserSample

What’s the catch?

Error handling

Errors in gradle console

- Data binding errors are printed out in gradle build console with kotlin
- Doesn’t work with Instant Apps
- Adds complexity to build process
- But … Google has committed to support

What’s the future of data binding?

Special thanks to Yigit Boyar, George Mount, Roberto Orgiu, Nate Ebel, Danny Preussler, and Aidan McWilliams!

Questions?

Lisa Wray
@lisawrayz