IMMOBILIEN SCOUT 24

Leveraging Android Databinding with Kotlin

What is Data Binding?

Data Binding Beta

2015

Data Binding v2

Data Binding v2

Incremental class generation ⚡

Data Binding v2

Incremental class generation ⚡

Pre-compilation class generation 💪

Data Binding

```
class RepoActivity : AppCompatActivity() {

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

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)
  }
}
```

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)
  }
}
```

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_home)
    binding.setLifecycleOwner(this)
  }
}
```

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)

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

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

Introducing

Android Architecture Components

Model-View-ViewModel


View
ViewModel
Model

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.setLifecycleOwner(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.setLifecycleOwner(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)
```

Model-View-ViewModel

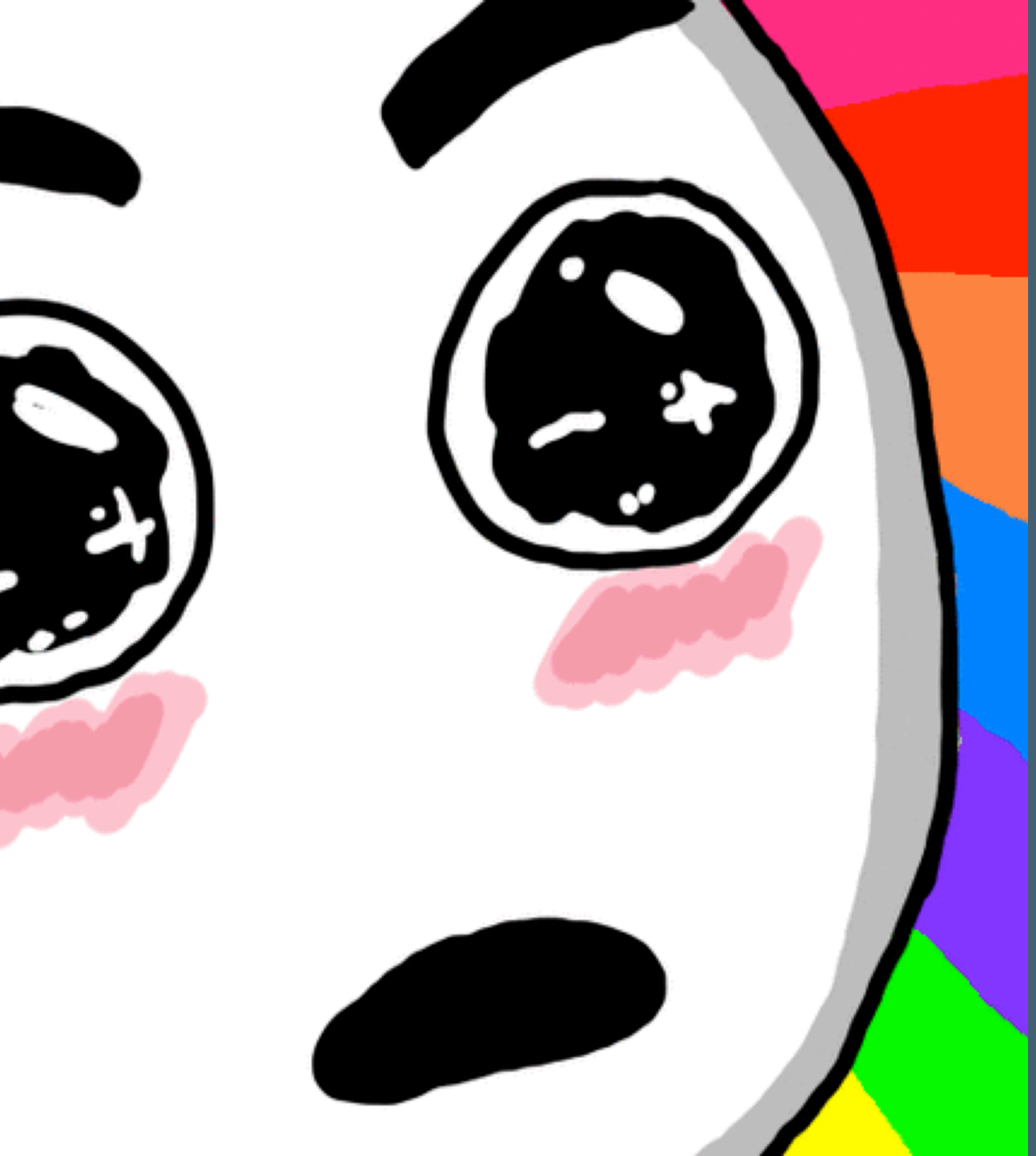
```
class RepoActivity : AppCompatActivity() {

  private val binding by activityBinding<ActivityRepoBinding>(R.layout.activity_repo)
  private val model by lazy { getViewModel<RepoViewModel>(RepoViewModelFactory()) }

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

    binding.setLifecycleOwner(this)
    binding.model = model

    // val items = model.items.get()
  }
}
```

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) { /* ... */ }
}
```



LiveData Data Binding

LiveData Data Binding

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

    val items = MutableLiveData<List<String>>()

    fun query(value: String) { /* ... */ }
}
```

ProTip: Extension Functions! 👍

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

  val items = mutableLiveData<List<String>>()

  fun query(value: String) { /* ... */ }
}

fun <T> ViewModel.mutableLiveDataOf() = MutableLiveData<T>()
```

ViewModel

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

  val items = mutableLiveDataOf<List<String>>()

  fun query(value: String) {
    service.fetch(value) {
      items.value = it
    }
  }
}
```

ViewModel

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

  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
    }
  }
}
```

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 &amp;&amp; !model.loading ? View.VISIBLE : View.GONE}"/>

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

</layout>
```


Too Complicated!
1
ピタゴラ

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

  val items = mutableLiveDataOf<List<String>>()
  val loading = mutableLiveDataOf<Boolean>()
  val empty = mutableLiveDataOf<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 ObservableViewModel : 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 = mutableLiveDataOf<Boolean>()

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

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

@Bindable

Delegated Properties

Property Delegate

ObservableProperty

@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)
  }
}
```

ProTip: Extension Function! 👍

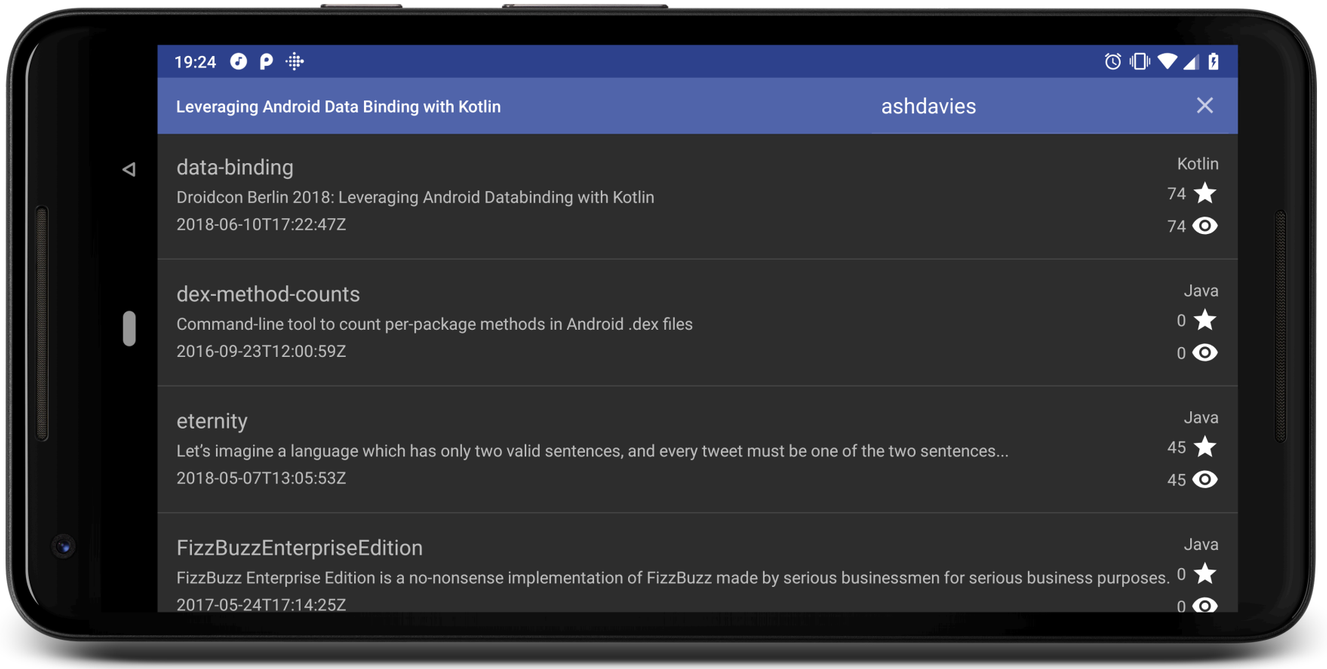
@Bindable

```
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)
}
```

🎉 Awesome! 🎉

Data Binding Sample

github.com/ashdavies/data-binding



Android Data Binding with Kotlin

bit.ly/2yU8qUz

Cheers! 🍻

_ashdavies