Pragmatic Kotlin ❤

Practical Tips to migrate your Android App to Kotlin

By Ravindra Kumar @ravidsrk

About Me

- Ravindra Kumar **@ravidsrk**
- Android Developer **@Fueled**
- Speaker at **Droidcon In, Jsfoo, TiConf, Devfest**
- Creator of **AndroidStarters.com** and **KotlinExtensions.com**
- Open source contributor **@ravidsrk**
- Author of **Android Testing Guide**

Agenda

- Steps to Convert
- Common converter Issues
- Takeaways
- Eliminate all **!!** from your Kotlin code
- Kotlin Extensions
- Kotlin Android Extensions

Steps to Convert

Once you learn basics syntax of **Kotlin**

1. Convert files, one by one, via "**⌥⇧⌘K**", make sure tests still pass

Steps to Convert

Once you learn basics syntax of **Kotlin**

1. Convert files, one by one, via "**⌥⇧⌘K**", make sure tests still pass
2. Go over the Kotlin files and make them more idiomatic.

Steps to Convert

Once you learn basics syntax of **Kotlin**

1. Convert files, one by one, via "⌥⇧⌘K", make sure tests still pass
2. Go over the Kotlin files and make them more idiomatic.
3. Repeat step 2 until you convert all the files.

Steps to Convert

Once you learn basics syntax of **Kotlin**

1. Convert files, one by one, via "**⌥⇧⌘K**", make sure tests still pass
2. Go over the Kotlin files and make them more idiomatic.
3. Repeat step 2 until you convert all the files.
4. Ship it.

Common Converter Issues

- **TypeCasting** for the sake of Interoperability.
- **Companion** will add extra layer.
- If java method starting with **getFoo()**, converter looks for property with the name **foo**.
- Generics are hard to get it right on the first go.
- No argument captor.
- **git diff** If two developers are working on same

TypeCasting **for the sake of** Interoperability

Here is the Java class:

```
public class DemoFragment extends BaseFragment implements DemoView {

    @Override
    public void displayMessageFromApi(String apiMessage) {
      ...
    }
}
```

TypeCasting **for the sake of** Interoperability

```
// Kotlin class
class DemoResponse {
    @SerializedName("message") var message: String? = null
}
```

TypeCasting **for the sake of** Interoperability

```
// Kotlin class
class DemoResponse {
    @SerializedName("message") var message: String? = null
}

// Typecasting to String
mainView?.displayMessageFromApi(demoResponse.message)
```

TypeCasting **for the sake of** Interoperability

```
// Kotlin class
class DemoResponse {
    @SerializedName("message") var message: String? = null
}

// Typecasting to String
mainView?.displayMessageFromApi(demoResponse.message) // Compiler Error
```

TypeCasting **for the sake of** Interoperability

```
// Kotlin class
class DemoResponse {
    @SerializedName("message") var message: String? = null
}

// Typecasting to String
mainView?.displayMessageFromApi(demoResponse.message as String)
```

Companion **will add extra layer**

Here is Java class:

```
public class DetailActivity extends BaseActivity implements DetailMvpView{
    public static final String EXTRA_POKEMON_NAME = "EXTRA_POKEMON_NAME";

    public static Intent getStartIntent(Context context, String pokemonName) {
        Intent intent = new Intent(context, DetailActivity.class);
        intent.putExtra(EXTRA_POKEMON_NAME, pokemonName);
        return intent;
    }
}
```

Companion **will add extra layer**

Converted Kotlin class:

```
class DetailActivity : BaseActivity(), DetailMvpView {
    companion object {
        val EXTRA_POKEMON_NAME = "EXTRA_POKEMON_NAME"

        fun getStartIntent(context: Context, pokemonName: String): Intent {
            val intent = Intent(context, DetailActivity::class.java)
            intent.putExtra(EXTRA_POKEMON_NAME, pokemonName)
            return intent
        }
    }
}
```

Companion **will add extra layer**

```
public class MainActivity extends BaseActivity implements MainMvpView {
  private void pokemonClicked(Pokemon pokemon) {
      startActivity(DetailActivity.Companion.getStartIntent(this, pokemon))
  }
}
```

Add @JvmStatic

Converted Kotlin class:

```
class DetailActivity : BaseActivity(), DetailMvpView {
    companion object {
        val EXTRA_POKEMON_NAME = "EXTRA_POKEMON_NAME"

        @JvmStatic
        fun getStartIntent(context: Context, pokemonName: String): Intent {
            val intent = Intent(context, DetailActivity::class.java)
            intent.putExtra(EXTRA_POKEMON_NAME, pokemonName)
            return intent
        }
    }
}
```

Add @JvmStatic

```
public class MainActivity extends BaseActivity implements MainMvpView {
  private void pokemonClicked(Pokemon pokemon) {
      startActivity(DetailActivity.getStartIntent(this, pokemon))
  }
}
```

Add @JvmStatic

```
public class MainActivity extends BaseActivity implements MainMvpView {
  private void pokemonClicked(Pokemon pokemon) {
      startActivity(DetailActivity.getStartIntent(this, pokemon))
  }
}
```

Remember: **you do not need to stress about migrating the entire codebase.*

Method names starting with get

Here is the Java class:

```
public interface DemoService {
    @GET("posts")
    Observable<PostResponse> getDemoResponse();

    @GET("categories")
    Observable<CategoryResponse> getDemoResponse2();
}
```

Method names starting with get

```
interface DemoService {
    @get:GET("posts")
    val demoResponse: Observable<PostResponse>

    @get:GET("categories")
    val demoResponse2: Observable<CategotyResponse>
}
```

Expecting methods **demoResponse** and **demoResponse2**, They are being interpreted as getter methods, this will cause lots of issues.

No ArgumentCaptor

If you are using Mockito’s ArgumentCaptor you will most probably get following error

```
java.lang.IllegalStateException: classCaptor.capture() must not be null
```

No ArgumentCaptor

If you are using Mockito’s ArgumentCaptor you will most probably get following error

```
java.lang.IllegalStateException: classCaptor.capture() must not be null
```

The return value of **classCaptor.capture()** is null, but the signature of **SomeClass#someMethod(Class, Boolean)** does not allow a *null* argument.

No ArgumentCaptor

If you are using Mockito's ArgumentCaptor you will most probably get following error

```
java.lang.IllegalStateException: classCaptor.capture() must not be null
```

The return value of **classCaptor.capture()** is null, but the signature of **SomeClass#someMethod(Class, Boolean)** does not allow a *null* argument.

mockito-kotlin library provides supporting functions to solve this problem

Key Takeaways

- **annotationProcessor** must be replaced by **kapt** in build.gradle

Key Takeaways

- **annotationProcessor** must be replaced by **kapt** in build.gradle
- Configure tests to mock final classes

Key Takeaways

- **annotationProcessor** must be replaced by **kapt** in build.gradle
- Configure tests to mock final classes
- If you are using android **data-binding**, include:

```
kapt com.android.databinding:compiler:3.0.0
```

Key Takeaways

- **annotationProcessor** must be replaced by **kapt** in build.gradle
- Configure tests to mock final classes
- If you are using android **data-binding**, include:

```
kapt com.android.databinding:compiler:3.0.0
```

- **@JvmField** to rescue while using ButterKnife **@InjectView** and Espresso **@Rule**

Eliminate all !! from your Kotlin **code**

1. Use **val** instead of **var**
2. Use **lateinit**
3. Use **let** function
4. User **Elivis** operator

Use val **instead of** var

- Kotlin makes you think about immutability on the language level and that’s great.
- *var* and *val* mean "writable" and "read-only"
- If you use them as immutables, you don’t have to care about nullability.

Use lateinit

```
private var adapter: RecyclerAdapter<Droids>? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    mAdapter = RecyclerAdapter(R.layout.item_droid)
}

fun updateTransactions() {
    adapter!!.notifyDataSetChanged()
}
```

Use lateinit

```
private lateinit var adapter: RecyclerAdapter<Droids>

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    mAdapter = RecyclerAdapter(R.layout.item_droid)
}

fun updateTransactions() {
    adapter?.notifyDataSetChanged()
}
```

Use let function

```
private var photoUrl: String? = null

fun uploadClicked() {
    if (photoUrl != null) {
        uploadPhoto(photoUrl!!)
    }
}
```

Use let function

```
private var photoUrl: String? = null

fun uploadClicked() {
    photoUrl?.let { uploadPhoto(it) }
}
```

User Elivis operator

Elvis operator is great when you have a fallback value for the null case. So you can replace this:

```
fun getUserName(): String {
    if (mUserName != null) {
        return mUserName!!
    } else {
        return "Anonymous"
    }
}
```

User Elivis **operator**

Elvis operator is great when you have a fallback value for the null case. So you can replace this:

```
fun getUserName(): String {
    return mUserName ?: "Anonymous"
}
```

Kotlin Extensions

```
Toast.makeText(context, "Hello #BlrKotlin :)", Toast.LENGTH_LONG).show()
```

Kotlin Extensions

```
Toast.makeText(context, "Hello #BlrKotlin :)", Toast.LENGTH_LONG).show()
```

```
/**
 * Extension method to show toast for Context.
 */
fun Context?.toast(@StringRes textId: Int, duration: Int = Toast.LENGTH_LONG) =
 this?.let { Toast.makeText(it, textId, duration).show() }
```

Kotlin Extensions

```
Toast.makeText(context, "Hello #BlrKotlin :)", Toast.LENGTH_LONG).show()
```

```
/**
 * Extension method to show toast for Context.
 */
fun Context?.toast(@StringRes textId: Int, duration: Int = Toast.LENGTH_LONG) =
 this?.let { Toast.makeText(it, textId, duration).show() }
```

Check http://kotlinextensions.com/

Kotlin Android Extensions

- Goodbye **findViewById**
- Using **Parcelize** annotation for Parcelable

Kotlin Android Extensions

- Goodbye **findViewById**
- Using **Parcelize** annotation for Parcelable

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
```

Goodbye findViewById

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

    <TextView
        android:id="@+id/welcomeMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Hello World!"/>

</FrameLayout>
```

Goodbye findViewById

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    TextView welcomeMessageView = findViewById(R.id.welcomeMessage),

    welcomeMessageView.text = "Hello BlrKotlin!"
}
```

Goodbye findViewById

```
import kotlinx.android.synthetic.main.activity_main.*
```

Goodbye findViewById

```
import kotlinx.android.synthetic.main.activity_main.*

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    welcomeMessage.text = "Hello BlrKotlin!"
}
```

Using Parcelize annotation for Parcelable

Here is the Java Class:

```
public class MyParcelable implements Parcelable {
     private int mData;
     public int describeContents() { return 0; }
     public void writeToParcel(Parcel out, int flags) { out.writeInt(mData); }
     public static final Parcelable.Creator<MyParcelable> CREATOR
             = new Parcelable.Creator<MyParcelable>() {

         public MyParcelable createFromParcel(Parcel in) { return new MyParcelable(in); }

         public MyParcelable[] newArray(int size) { return new MyParcelable[size];}
     };

     private MyParcelable(Parcel in) { mData = in.readInt(); }
 }
```

Using Parcelize annotation for Parcelable

Converted Kotlin Class:

```
data class MyParcelable(var data: Int): Parcelable {
    override fun describeContents() = 1
    override fun writeToParcel(dest: Parcel, flags: Int) { dest.writeInt(data)}

    companion object {
        @JvmField
        val CREATOR = object : Parcelable.Creator<MyParcelable> {
            override fun createFromParcel(source: Parcel): MyParcelable {
                val data = source.readInt(); return MyParcelable(data)
            }
            override fun newArray(size: Int) = arrayOfNulls<MyParcelable>(size)
        }
    }
}
```

Using Parcelize annotation for Parcelable

To use **@Parcelize** we need to set experimental flag in **build.gradle**

```
androidExtensions {
    experimental = true
}
```

Using Parcelize annotation for Parcelable

To use **@Parcelize** we need to set experimental flag in **build.gradle**

```
androidExtensions {
    experimental = true
}

@Parcelize
class MyParcelable(val data: Int): Parcelable
```

Final tip

Final tip

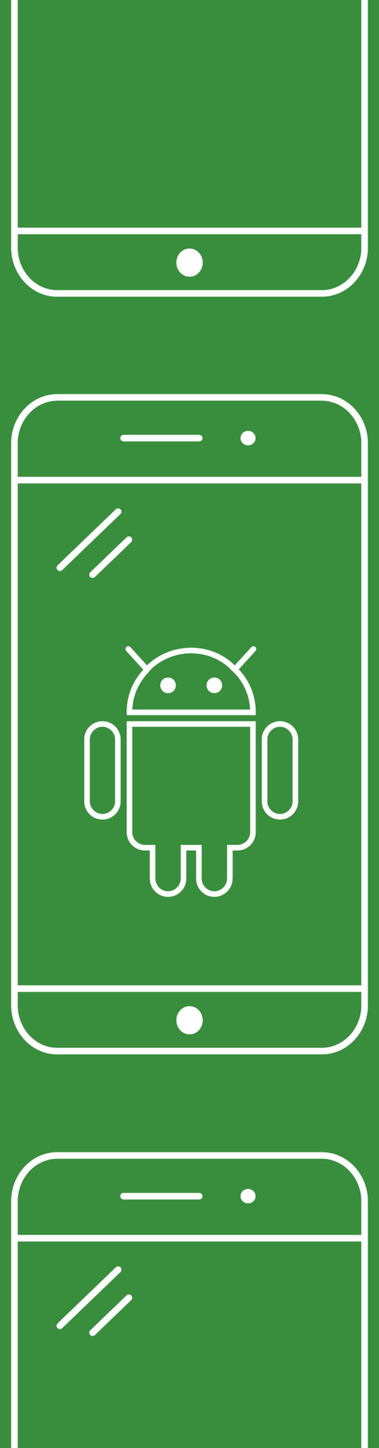
Don’t try to learn the whole language at once

Questions?

Thank You

Android Testing Guide

- Everything to start writing tests for Android App.
- 75% discount on my upcoming book use BLRKOTLIN
- https://leanpub.com/android-testing/c/BLRKOTLIN



Android Testing Guide

Practical tips and techniques for testing real-world androidapplications.

Ravindra Kumar