Pitfalls when mixing Kotlin & Java

Bob Dahlberg
Hedy Tech

Bob Dahlberg

Hedy Tech
Challenge Accepted

```
Who's Bob?
- Building software
- CrossFit Instructor
- Father of one
- Arranging meetups
- Loves a good challenge!
```

Background

Android

Been an Android developer since Android Cupcake (2009).

Only Java

Background

Android
Scala

```
Learned Scala and tried to build
Android with it..
```

Background

Android
Scala

Learned Scala and tried to build Android with it..

It’s plausible.

Background

Android
Scala

Learned Scala and tried to build Android with it..

It’s plausible.

But you rather die a little than trying to mix it with Java.

Background

Android
Scala
Kotlin

```
Found Kotlin M12 in 2015 and it
blended almost seamlessly with
Java.
```

Background

Android
Scala
Kotlin

Found Kotlin M12 in 2015 and it blended almost seamlessly with Java.

A few years later even Google understood it and made it an official language on Android.

Background

Android
Scala
Kotlin

Found Kotlin M12 in 2015 and it blended almost seamlessly with Java.

A few years later even Google understood it and made it an official language on Android.

But there are a few things to keep in mind when mixing.

The goal should be to move
completely to Kotlin.

All the way

Blending

Although blending Java and Kotlin works great you can’t write idiomatic Java nor Kotlin code when blending.

You should choose.

All the way

Blending

Although blending Java and Kotlin works great you can’t write idiomatic Java nor Kotlin code when blending.

~~You should choose.~~
You have to choose!

All the way

Blending
Choose Java

- If you’re fairly new to Kotlin and wicked at Java.
- Don’t know how to write idiomatic Kotlin code.
- Can’t invest in learning in the short run or want to slowly adapt to it.

All the way

Blending
Choose Java
Choose Kotlin

- Already know how to write idiomatic Kotlin code.
- Can invest in learning.
- If it doesn’t mean to re-write the entire app upfront.

All the way

Blending
Choose Java
Choose Kotlin
Be pragmatic

The idioms should be a goal to strive for.

Not enforced in every change.

All the way

Blending
Choose Java
Choose Kotlin
Be pragmatic
Permanent mixing

If you have a great separation of concern and code, a modular architecture, you can have idiomatic Java and idiomatic Kotlin mixed permanently.

But it’s a rare case.

Don’t go all the way!

Pace yourself

Auto convert

```
Don’t use it, please!
```

Pace yourself

Auto convert

```
Don’t use it, please!
Only one file at the time!
```

Pace yourself

Auto convert
Explanation mark bonanza

```
// Converted from Java
player!!.prepare(audioRenderer)
if (!stream!!.atLiveEdge()) {
    player!!.seekTo(stream!!.getPosition())
}
player!!.setTrack(liveQuality)
```

Pace yourself

Auto convert
Explanation mark bonanza
Kotlin Friendly

```
If you've written your Java code
so that it can be easily
interpreted, the converted
Kotlin code is better.
```

Pace yourself

Auto convert
Explanation mark bonanza
Kotlin Friendly

```
If you've written your Java code
so that it can be easily
interpreted, the converted
Kotlin code is better.

final
```

Pace yourself

Auto convert
Explanation mark bonanza
Kotlin Friendly

```
If you've written your Java code
so that it can be easily
interpreted, the converted
Kotlin code is better.

final
@NotNull
```

Pace yourself

Auto convert
Explanation mark bonanza
Kotlin Friendly

If you've written your Java code so that it can be easily interpreted, the converted Kotlin code is better.

```
final
@NotNull
@Nullable
```

Pace yourself

Auto convert
Explanation mark bonanza
Kotlin Friendly
Refactor the converted code

```
// Converted from Java
player!!.prepare(audioRenderer)
if (!stream!!.atLiveEdge()) {
    player!!.seekTo(stream!!.getPosition())
}
player!!.setTrack(liveQuality)
```

Pace yourself

Auto convert
Explanation mark bonanza
Kotlin Friendly
Refactor the converted code

```
// Converted from Java
player!!.prepare(audioRenderer)
if (!stream!!.atLiveEdge()) {
    player!!.seekTo(stream!!.getPosition())
}
player!!.setTrack(liveQuality)

// Kotlin - null safe
player?.prepare(audioRenderer)
if (stream?.atLifeEdge() == false) {
  player?.seekTo(stream.position)
}
player?.track = liveQuality
```

Pace yourself

Auto convert
Explanation mark bonanza
Kotlin Friendly
Refactor the converted code

```
// Converted from Java
player!!.prepare(audioRenderer)
if (!stream!!.atLiveEdge()) {
    player!!.seekTo(stream!!.getPosition())
}
player!!.setTrack(liveQuality)

// Kotlin - null safe
player?.prepare(audioRenderer)
if (stream?.atLifeEdge() == false) {
  player?.seekTo(stream.position)
}
player?.track = liveQuality

// Kotlin - null safe
player?.let {
  it.prepare(audioRenderer)
  if (stream?.atLifeEdge() == false) {
    it.seekTo(stream.position)
  }
  it.track = liveQuality
}
```

Where to start?

Path of least resistance

Hit it were it don’t hurt

```
Proof of Concept
Prove that your project works
with Kotlin and that you and the
team understand how to mix it.
```

Path of least resistance

Hit it were it don’t hurt
Isolated

`Tests`

Path of least resistance

Hit it were it don’t hurt
Isolated
Small

`POJOs`

Path of least resistance

Hit it were it don’t hurt
Isolated
Small
Independent

`Utilities & Helpers`

Preparations

Decide upfront

How should you handle…

Decide upfront

How should you handle…

```
static java code
```

Decide upfront

How should you handle…

```
static java code
null
```

Decide upfront

How should you handle…

```
static java code
null
extensions
```

Decide upfront

How should you handle…

```
static java code
null
extensions
visibility modifiers
```

Decide upfront

How should you handle…

```
static java code
null
extensions
visibility modifiers
POJOs
```

Decide upfront

How should you handle…

```
static java code
null
extensions
visibility modifiers
POJOs
factories
```

Decide upfront

How should you handle…

```
static java code
null
extensions
visibility modifiers
POJOs
factories
singeltons
```

Decide upfront

How should you handle…
Differences

```
Read up on the big differences
and decide how you will tackle
those challenges.
```

When you need more

Bits & Bytes

Byte code

In IntelliJ there’s a tool to see the byte code generated from your code.

Bits & Bytes

Byte code
Decompile

In IntelliJ there’s a tool to see the byte code generated from your code.

And if you decompile that you’ll see the equivalent java code.

Bits & Bytes

Byte code
Decompile
Try

In IntelliJ there’s a tool to see the byte code generated from your code.

And if you decompile that you’ll see the equivalent java code.

Try with a simple data class.

Let’s see some code

Code

typealias

```
typealias Degrees = Double
typealias Farenheit = Double

class Example {
  fun convert(degrees:Celcius): Farenheit {
    return degrees * 1.8 + 32
  }
}
```

Code

typealias

```
typealias Degrees = Double
typealias Farenheit = Double

class Example {
  fun convert(degrees:Celcius): Farenheit {
    return degrees * 1.8 + 32
  }
}
```

```
// java signature
convert(double degrees) double
```

Code

typealias
function parameters

```
typealias Task = () -> Unit

fun execute(task:Task) {
  task()
}
```

Code

typealias
function parameters

```
typealias Task = () -> Unit

fun execute(task:Task) {
  task()
}
```

```
// kotlin
execute {
  println("Anonymous task")
}
```

Code

typealias
function parameters

```kotlin
typealias Task = () -> Unit

fun execute(task:Task) {
  task()
}
```

```kotlin
// kotlin
execute {
  println("Anonymous task")
}
```

```java
// Java 6
execute(
  new Function0<Unit>() {
    @Override
    public Unit invoke() {
      System.out.println("Anonymous task");
      return Unit.INSTANCE;
    }
  }
);
```

Code

typealias
function parameters

```
typealias Task = () -> Unit

fun execute(task:Task) {
  task()
}
```

```
// kotlin
execute {
  println("Anonymous task")
}
```

Code

typealias
function parameters

```
typealias Task = () -> Unit

fun execute(task:Task) {
  task()
}
```

```
// kotlin
execute {
  println("Anonymous task")
}
```

Code

typealias
function parameters

```kotlin
typealias Task = () -> Unit

fun execute(task:Task) {
  task()
}
```

```kotlin
// kotlin
execute {
  println("Anonymous task")
}
```

```java
// Java 8
execute(() -> {
  System.out.println("Anonymous task");
  return Unit.INSTANCE;
});
```

Code

typealias
function parameters
static-ish

```
class Example {
  companion object {

    val TAG: String = "Example"
  }
}

// From Kotlin
Example.TAG
```

Code

typealias
function parameters
static-ish

```
class Example {
  companion object {

    val TAG: String = "Example"
  }
}

// From Kotlin
Example.TAG
```

```
Example.Companion.getTAG();
```

Code

typealias
function parameters
static-ish

```kotlin
class Example {
  companion object {
    @JvmStatic
    val TAG: String = "Example"
  }
}

// From Kotlin
Example.TAG
```

Code

typealias
function parameters
static-ish

```kotlin
class Example {
  companion object {
    @JvmStatic
    val TAG: String = "Example"
  }
}

// From Kotlin
Example.TAG
```

```
Example.getTAG();
```

Code

typealias
function parameters
static-ish

```
class Example {
  companion object {

    const val TAG: String = "Example"
  }
}

// From Kotlin
Example.TAG
```

Code

typealias
function parameters
static-ish

```kotlin
class Example {
  companion object {

    const val TAG: String = "Example"
  }
}

// From Kotlin
Example.TAG
```

```
Example.TAG;
```

Code

typealias
function parameters
static-ish

```
class Example {
  companion object {
    const val TAG: String = "Example"

    @JvmField
    var state: String = "Lorem"

    @JvmStatic
    var status: String = "Ipsum"

    @JvmStatic
    fun validate(value:Example): Boolean {...}
  }
}
```

Code

typealias
function parameters
static-ish
singelton

```
object Example {

  const val TAG: String = "Example"

  @JvmField
  var state: String = "Lorem"

  @JvmStatic
  var status: String = "Ipsum"

  @JvmStatic
  fun validate(value:Example): Boolean {...}

}
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values

```kotlin
// FileName StringExtensions.kt
fun String.toList(separator: String = " "): List<String> {...}

"Split me up".toList()             // List("Split","me","up")
"com.example.kotlin".toList(".")   // List("com","example","kotlin")
```

```kotlin
StringExtensionsKt.toList("Split me up", " "); // No defaults...
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values

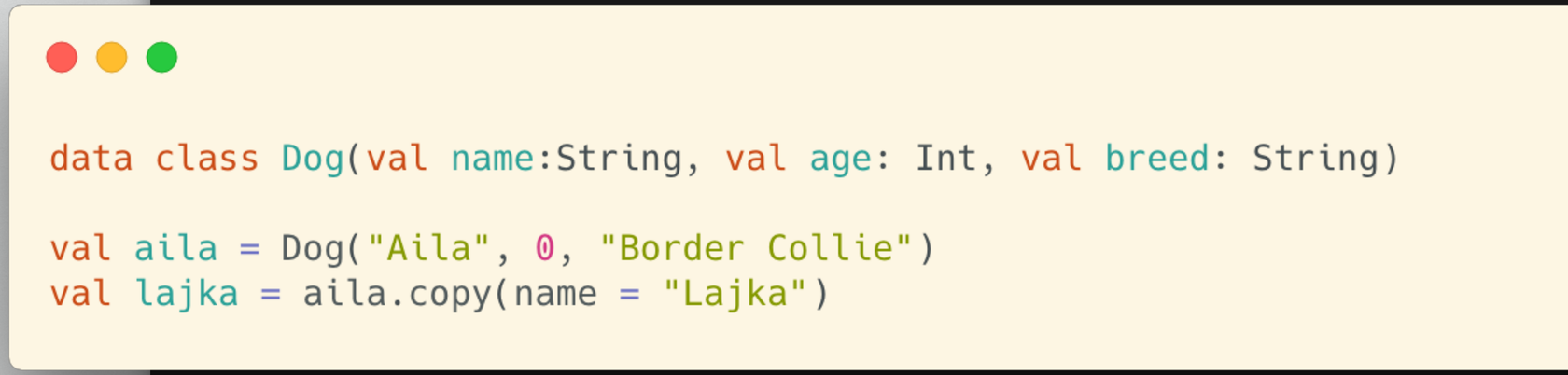
```kotlin
// FileName StringExtensions.kt
@file:JvmName("StringExt")
fun String.toList(separator: String = " "): List<String> {...}

"Split me up".toList()             // List("Split","me","up")
"com.example.kotlin".toList(".")   // List("com","example","kotlin")
```

```kotlin
StringExt.toList("Split me up", " "); // But still no defaults...
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes



```
data class Dog(val name:String, val age: Int, val breed: String)

val aila = Dog("Aila", 0, "Border Collie")
val lajka = aila.copy(name = "Lajka")
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes

```
data class Dog(val name:String, val age: Int, val breed: String)

val aila = Dog("Aila", 0, "Border Collie")
val lajka = aila.copy(name = "Lajka")
```

```
Dog aila = new Dog("Aila", 0, "Border Collie");
Dog lajka = aila.copy("Lajka", 0, "Border Colie");
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes

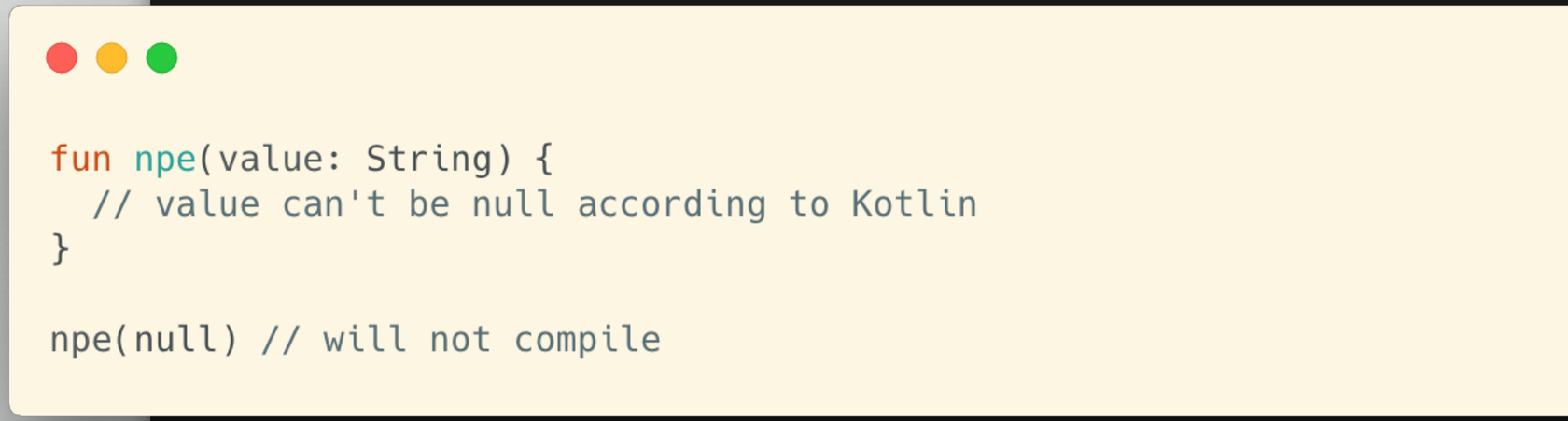
```
data class Dog(val name:String, val age: Int, val breed: String)

val aila = Dog("Aila", 0, "Border Collie")
val lajka = aila.copy(name = "Lajka")
```

```
Dog aila = new Dog("Aila", 0, "Border Collie");
//Dog lajka = aila.copy("Lajka", 0, "Border Colie");
Dog lajka = new Dog("Lajka", 0, "Border Colie");
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null



```
fun npe(value: String) {
  // value can't be null according to Kotlin
}

npe(null) // will not compile
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null

```kotlin
fun npe(value: String) {
  // value can't be null according to Kotlin
}

npe(null) // will not compile
```

```java
// compiles and is guaranteed to throw a IllegalArgumentException
npe(null);
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null

```kotlin
fun npe(value: String?) {
  // handle null in this scope instead
}

npe(null) // compiles
```

```kotlin
// compiles and null is handled in the Kotlin function
npe(null);
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing

```
data class KSample(val first: String, val second: String)
val (k1, k2) = KSample("one", "two")
println("First: $k1 and second: $k2") // First: one and second: two
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing

```kotlin
data class KSample(val first: String, val second: String)
val (k1, k2) = KSample("one", "two")
println("First: $k1 and second: $k2") // First: one and second: two
```

```java
public class JSample {
  private final String first;
  private final String second;

  public Sample(String first, String second) {
    this.first = first;
    this.second = second;
  }
}
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing

```kotlin
data class KSample(val first: String, val second: String)
val (k1, k2) = KSample("one", "two")
println("First: $k1 and second: $k2") // First: one and second: two
```

```java
public class JSample {
  private final String first;
  private final String second;

  public Sample(String first, String second) {
    this.first = first;
    this.second = second;
  }
}
```

```kotlin
val (j1, j2) = JSample("one", "two")  // Won't compile..
println("First: $k1 and second: $k2")
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing

```kotlin
data class KSample(val first: String, val second: String)
val (k1, k2) = KSample("one", "two")
println("First: $k1 and second: $k2") // First: one and second: two
```

```java
public class JSample {
  private final String first;
  private final String second;

  public Sample(String first, String second) {
    this.first = first;
    this.second = second;
  }

  public String component1() { return first; }
  public String component2() { return second; }
}
```

```kotlin
val (j1, j2) = JSample("one", "two")
println("First: $k1 and second: $k2") // First: one and second: two
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing

```kotlin
data class KSample(val first: String, val second: String)
val (k1, k2) = KSample("one", "two")
println("First: $k1 and second: $k2") // First: one and second: two
```

```java
public class JSample {
  private final String first;
  private final String second;

  public Sample(String first, String second) {
    this.first = first;
    this.second = second;
  }

  public String component1() { return first; }
  public String component2() { return second; }
}
```

```kotlin
val (j1, j2) = JSample(null, null)
j1.length // NPE...
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing
more conventions

```
data class Dog(val name: String, val age: Int, val breed: String) {
    operator fun compareTo(other: Dog) = age.compareTo(other.age)
}
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing
more conventions

```
data class Dog(val name: String, val age: Int, val breed: String) {
    operator fun compareTo(other: Dog) = age.compareTo(other.age)
}
```

```
val aila = Dog("Aila", 0, "Border Collie")
val lajka = aila.copy("Lajka", 3)

println(aila > lajka) // false
println(aila < lajka) // true
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing
more conventions

```kotlin
data class Dog(val name: String, val age: Int, val breed: String) {
    operator fun compareTo(other: Dog) = age.compareTo(other.age)
}
```

```kotlin
val aila = Dog("Aila", 0, "Border Collie")
val lajka = aila.copy("Lajka", 3)

println(aila > lajka) // false
println(aila < lajka) // true
```

```java
Dog aila = new Dog("Aila", 0, "Border Collie");
Dog lajka = new Dog("Lajka", 3, "Border Collie");
System.out.println(aila.compareTo(lajka)); // -1
```

Code

typealias
function parameters
static-ish
singelton
extensions & default values
data classes
null
deconstructing
more conventions

```kotlin
data class Dog(val name: String, val age: Int, val breed: String) {
    operator fun compareTo(other: Dog) = age.compareTo(other.age)
}
```

```kotlin
val aila = Dog("Aila", 0, "Border Collie")
val lajka = aila.copy("Lajka", 3)

println(aila > lajka) // false
println(aila < lajka) // true
```

```java
Dog aila = new Dog("Aila", 0, "Border Collie");
Dog lajka = new Dog("Lajka", 3, "Border Collie");
System.out.println(aila.compareTo(lajka)); // -1
```

```kotlin
val meh = JavaComparable(...)
val huh = JavaComparable(...)

meh > huh // works thanks to conventions
```

To sum up

Take aways

Differences

```
Learn the differences

- static
- visibility
- conventions
- null
- etc
```

Take aways

Differences
Pragmatic

Set a solid goal!
But be pragmatic about how to reach it.

Take aways

Differences
Pragmatic
Iterate

As in all development, but maybe more important in this case.

Convert
Refactor, Refactor, Refactor
Repeat

Take aways

Differences
Pragmatic
Iterate
Cheat

```
Decompile the kotlin byte code
to see what to expect from java.

There’s no need to guess.
```

Take aways

Differences
Pragmatic
Iterate
Cheat
Help Kotlin

```
Use final, @Nullable and
@NotNull extensively in java.

And set lint warnings to errors
instead to avoid as many
problems with null.
```

Take aways

Differences
Pragmatic
Iterate
Cheat
Help Kotlin
Organic

Let the Kotlin code base grow and take over organically so that your team have time to adopt and learn.

Isolated, small, independent.

Take aways

Differences
Pragmatic
Iterate
Cheat
Help Kotlin
Organic
Conventions

```
Use the strengths in Kotlin that
theres a lot of conventions that
is adopted for Java.
```

Take aways

Differences
Pragmatic
Iterate
Cheat
Help Kotlin
Organic
Conventions
Null

And where kotlin has features to be better than Java.

Be sure to use them smart.

Thank you!

Bob Dahlberg

bob@hedy.tech

@dahlberg.bob - medium.com
bobdahlberg - speakersdeck.com

When writing a lot of Kotlin code in a mixed project you'll start to get a false sense of security around null.