# Ceylon on Android

Gavin King
Red Hat

## Ceylon

A relatively new programming language which features:

- a powerful and extremely elegant static type system
- built-in modularity
- support for multiple virtual machine platforms: JVM, Android, JavaScript, Dart
- powerful multi-language interoperation: Java, JavaScript, Dart
- excellent tooling: CLI, Eclipse, IntelliJ, Android Studio

## Modularity in Ceylon

The module system offers:

- language level constructs for defining modules, expressing their dependencies, and controlling visibility between modules
- versioning
- module archives and module repositories and automatic fetching of dependencies at compilation time and runtime
- module isolation at runtime
- interoperation with Maven and npm
- assembler tools for: Ceylon assembly archives, fat JARs, WARs, WildFly Swarm, Jigsaw `mlib`, Maven repos, Dart assemblies

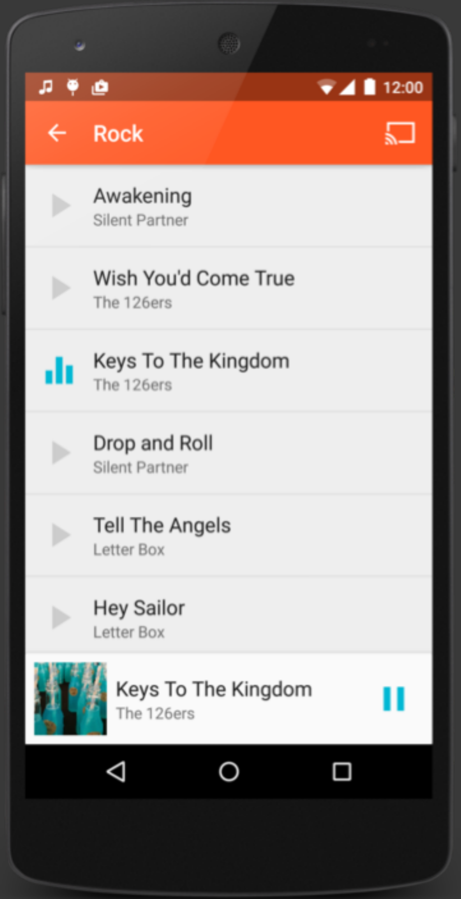
# Ceylon on Android

You get:

- True null safety, and in general, many more errors detected at compile time
- Anonymous functions (lambdas)
- powerful stream processing and comprehensions
- Union and intersection types
- Tuples
- Type inference and flow-sensitive typing
- Much better support for use of immutability
- Streamlined definition of “model” or “data” classes

## The challenge

- As a total novice to the Android platform, I wanted to port the Android sample app UniversalMusicPlayer from Java to Ceylon
- This is a sufficiently large program to test real-world usability
- I'm using Android Studio with the Ceylon IDE plugin
- To make this as realistic as possible, it was done in a massive rush under time constraints


12:00

Rock

Awakening
Silent Partner

Wish You'd Come True
The 126ers

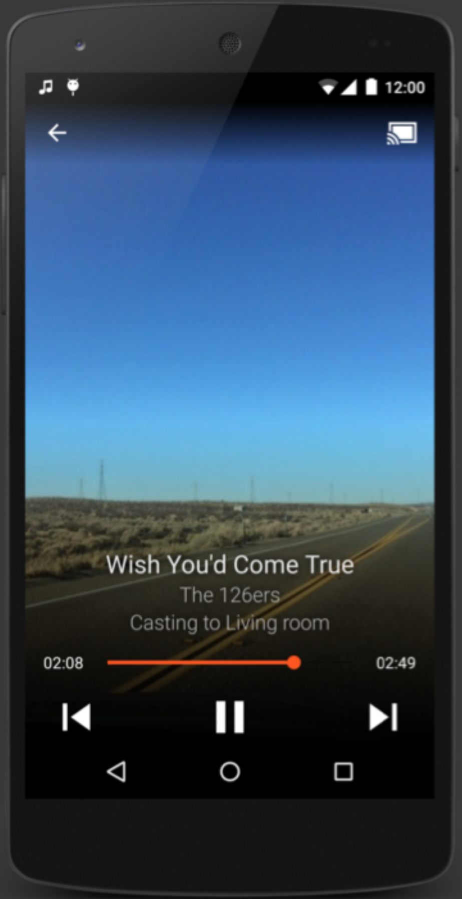
Keys To The Kingdom
The 126ers

Drop and Roll
Silent Partner

Tell The Angels
Letter Box

Hey Sailor
Letter Box

Keys To The Kingdom
The 126ers


12:00

Wish You'd Come True
The 126ers
Casting to Living room

02:08 02:49

## Integration with Gradle

First hurdle:

- The Ceylon compiler integrates dependency resolution and management, including Maven interop, as part of its module system
    - with dependencies defined in `module.ceylon`
- Android is extremely dependent on the use of Gradle for dependency management: to expand AAR assemblies, to generate `R.java` files, to fetch bits and pieces of the SDK, to run dex, etc
    - with dependencies defined in `build.gradle`

## Integration with Gradle

Solution:

- The Ceylon Gradle plugin for Android integrates the Ceylon compiler into Android's build process
- In particular, it aggregates the JAR archives produced by the Cradle build into a standard Ceylon module repository layout, making Android modules visible to Ceylon's module system
- And, of course, it runs the Ceylon compiler
- It supports the mixing of Ceylon and Java code in a single module

## Integration with Gradle

The solution works, but is imperfect:

- Copying files around makes the build process even a bit slower than it already is
- We have to express module dependencies twice: in `module.ceylon`, and in `build.gradle`
- On the other hand, it seems pretty robust and any “better” solution would probably break between Android releases
- Ultimately this is just some boilerplate stuff that you add to a new project

## Rewrite in Ceylon

‘Paste Java as Ceylon’ does most (90%) of the work:

- The two languages are sufficiently similar that most of the translation can be automated
- The Ceylon compiler allows a module to be written in a mix of Ceylon and Java code, so this can be incremental
- Thus, the process was to copy/paste one file at a time, trying to keep the app working all the way along

## Rewrite in Ceylon

First source of pain:

- The IntelliJ plugin does not (yet) make Ceylon declarations visible to Java source in the same module
    - The code actually compiles, as long as we ignore the annoying red errors highlighted in the Java code
- We must translate incrementally “top down”, starting with the UI
- Or we could write the new Ceylon code in a separate module

## Rewrite in Ceylon

Second source of pain:

- Lots of Java fields are “implicitly” null
- But it’s very hard to mechanically distinguish which
- Problem exacerbated by “late” initialization in, for example, `onCreate()` instead of constructor
- ‘Paste Java as Ceylon’ can’t distinguish these cases

## A word about null safety

The term “null safety” doesn’t just mean having nullable and not-nullable types

- In Java, `null` is the default value for an uninitialized field — and Java *doesn’t* prevent access to uninitialized fields, not even uninitialized `final` fields
- So “null safety” also requires some rather heavy-handed compile time validation of initialization logic, or the type system would simply be unsound!
- Since there are certain cases (circular references) which can’t be checked at runtime, Ceylon has a `late` annotation to mark those

## Rewrite in Ceylon

Therefore, the translation to Java involved lots of manual intervention to decide between three cases:

- The field can be definitely initialized in the initializer or constructor, and is never null
- The field is really initialized just once, but in an `onCreate()`-type method, declare it `late`
- The field is really nullable, declare its type `Whatever?`
- (In any one of these three cases, the fields might be `variable`!)

## Rewrite in Ceylon

- This is sufficiently tricky that I actually screwed up a couple of times, and got `InitializationErrors` from `late` fields at runtime
- But that's a lot better than getting `NullPointerExceptions` far from the source of the initialization bug!
- And these were basically the only errors I ran into at runtime — as usual, the Ceylon compiler found most of my bugs for me as soon as I typed them
- There was one class with such complex initialization dependencies between it and its several inner classes, that I actually had to struggle a bit to come up with something the compiler would accept

## Rewrite in Ceylon

- It’s also extremely common in Java to (unnecessarily) assign a local variable more than once
- ‘Paste Java as Ceylon’ correctly handles this case, of course, by annotating the local `variable`
- But that’s bad style in Ceylon, and I felt compelled to eliminate all these `variable`s by slightly restructuring the code

## Rewrite in Ceylon

- To my surprise, I discovered that Android developers are still forced to `switch` over integer constants (in 2017!)
- Ceylon developers have these oh-so incredibly sophisticated new things called “enumerated types”, so we don’t usually have to do that, and so Ceylon’s `switch` statement didn’t support it
- I got bored of rewriting `case` as `else if`, so I decided to downgrade a Ceylon compiler error to warning
- And now we can do this “fancy” switching on integer constants too!

## Rewrite in Ceylon

- As promised I was able to eliminate several inner classes and replace them with anonymous functions
- However, this had much less of an impact than I expected, since quite a number of Android's callback types are *classes* instead of interfaces
    - Ceylon supports conversion of SAM *interface* types, but not SAM class types
    - I've opened an issue ;-)
- Using an anonymous function loses identity, which caused a big bug

## Rewrite in Ceylon

- The sample app doesn't do much interesting stream or collection processing
  - But I did use some trivial stream processing and a couple of comprehensions
- For now, I didn't even bother to use Ceylon's collection types, and just left the `java.util` collections everywhere
  - There's no reason at all to avoid the use of Java collections in Ceylon!
  - Comprehensions, loops, operators, etc, work with Java collections

## Rewrite in Ceylon

- The Java language doesn't support null safety, but a number of Java libraries offer `@Nullable` / `@NotNull` annotations
- The Ceylon compiler will use them when determining the type of a Java method or field, if they exist
    - Otherwise, it records the fact that the method or field *might* be nullable
- Android has a `@Nullable` annotation but unfortunately it's declared `Retention(SOURCE)`, making it useless to us

## Conclusions

- I'm happy with the end result: the code is significantly cleaner, and it's more typesafe due to the way Ceylon handles null
- Since this was basically a lot of pretty mechanical UI-oriented plumbing code, and because of time constraints, I didn't really run into any problem which could show off any of the really awesome bits of Ceylon
- Ceylon IDE isn't perfect, but it was one huge advantage over other languages on IntelliJ — the 'Problems View'
- Setting breakpoints in Ceylon code did not work on Android for some reason
- I ran into a couple of bugs, and got them fixed

## Conclusions

End result is available at :

https://github.com/gavinking/UniversalMusicPlayer

But note that it depends on (unreleased) Ceylon 1.3.3.