A Tale of Two Languages

John Pampuch jpampuch@google.com

Key Messages

- Kotlin and Java complement each other
- Kotlin may have accelerated Java development
- Kotlin often leads in features, sometimes by quite a bit

Competition is good: it leads to better ~~approaches~~ everything!

Advantages

Kotlin

- Version independence from underlying platform(s)
  - Independent, more rapid, innovation
  - Slightly easier to support multiple deployment targets
  - E.g, JDK versions, Android versions, etc.
- Somewhat independent library structure
- Leverages the VM and OpenJDK Libraries

Java

- Aligned with JDK and VM development
- Mostly self-consistent
  - E.g, when lambdas were introduced, the entire library surface was refreshed to leverage!
- Highly predictable releases
- Mostly self contained platform

Design Goals

Kotlin

- Kotlin is pragmatic
- Kotlin is interoperable
- Kotlin is object oriented

Java

- Simple, object-oriented, and familiar
- Robust and secure
- Architecture-neutral and portable
- High performance
- Interpreted, threaded, and dynamic

Premise: Competition Leads to Better Results

- New ideas
- Faster progress
- Better outcomes
- Excitement!

Some Caveats

- This talk is mostly about Java and Kotlin
  - Focus on Language features
  - Sometimes this presentation touches on some aspects of the platform
- There is far more to the Java ecosystem than just the programming language
  - Also true of the broader Kotlin ecosystem
- Kotlin often benefits from platform enhancements
  - E.g., JVM performance, enhanced OpenJDK libraries
- Comparing Java and Kotlin is hard: both are part of extensive ecosystems that target slightly different realms.

More Caveats

- I’m highlighting major features
  - Not every feature is captured here
  - Nor on patch releases, security releases, i18n updates, time-zone updates
- Every release has many improvements: bug fixes, security fixes, performance improvements, etc.
  - A hidden “feature” of Java and Kotlin → continuous improvement!
- Both Java and Kotlin tend to release capabilities in at least two phases
  - experimental/preview/incubator
    - I refer to all of these as “Preview”
  - GA/stable/production-ready

A Brief History

Java and Kotlin Evolution since Kotlin’s announcement

2011

Kotlin

- Kotlin announced - July 22, 2011

  Starting from scratch (e.g., new from the ground up) would be easier, but would never get done!

- Instead: Pragmatism
- Bi-directional interop with Java
- And some hints of multi-platform

Java

- Java 7 - July 7, 2011
- Project Coin (small change!)
  - Strings in `switch`
  - Automatic resource management in try-statement aka try-with-resources statement
  - Improved type inference for generic instance creation, aka the diamond operator `<>`
  - Simplified varargs method declaration
  - Binary integer literals
  - Improved numeric literals
  - Catching multiple exception types and rethrowing exceptions with improved type checking
- Also: InvokeDynamic

A lot of time passes! (2012-2014)

- Java moves slowly
  - Sun was moving slowly!
  - For a while, JDK releases were a bit like Star Trek movies
  - Star Trek even numbered movies were good
    - JDK 6: Good
    - JDK 7: yawn
    - JDK 8: Good
    - JDK 9: yawn
- The new OpenJDK release schedule has shaken things up a lot!

- Kotlin
- Focused on getting to its first public milestone
- Few people saw the progress
- Opensourced in Feb 2012
- Priorities:
  - Useful
  - Avoid gratuitous differences - minimize relearning

2014-2016

Kotlin 1.0 - February 15, 2016

- Kotlin works where Java works
- User Experience in mind
  - IDE support, build performance, toolability
- Backwards compatibility commitment
- Smart casts
- `String` Templates
- Named arguments

Leverages many ecosystem elements

- Gradle, Maven, etc.

Java 8 - March 8, 2014

- First LTS release!
- Lambda expressions, and method references (Yay)
- Default methods in interfaces
- Type annotations, repeating annotations
- Improved type inference
- Method parameter reflection

Various platform updates

2017 - rapid updates

Kotlin 1.1 - March 1

- `typealias`
- Bound callable references
- Enhancements to `sealed` and `data` classes!
- Destructuring in lambdas
- Improved numeric literals
- Experimental: Coroutines

Kotlin 1.2 - November 28

- Improvements to type inference and smart casts
- Preview: Multiplat

Java 9 - September 21

- First “new” release model release
- Removed _ as a valid variable name

Kind of a yawner, except

- **Modules!**

2018

Kotlin 1.3 - October 28

- Preview: Contracts
- Improvements:
  - **`when`**
  - nested classes in annotations
  - parameterless **`main`**
- **Coroutines GA**
  - With the Flow API
- Kotlin/native
- Preview: API Opt-in
- Preview: **`inline class`**
- Preview: Unsigned Integers

Java 10 - March 20

- Local Var type inference

Java 11 - September 25

- Second LTS release!
- Improvements to Lambdas, type inference

2019

Kotlin

No major releases!

MANY Minor releases:

1.3.20, 1.3.21, 1.3.30, 1.3.40, 1.3.41, 1.3.50, 1.3.60, 1.3.61

Java 12 - March 19

- Preview: **`switch`** expressions
- Preview: Pattern matching

Java 13 - September 17

- Second Preview: **`switch`** expressions
- Preview: Text Blocks

2020

Kotlin 1.4 - August 17

- SAM conversions for Kotlin interfaces
- Named Arguments improvements
- Trailing Commas!

Java 14 - March 17

- `switch` expressions
- Preview: `record`

Java 15 - September 15

- Preview: Sealed Classes and interfaces

2021

Kotlin 1.5 - May 5

- JVM **`record`** support
- Sealed interfaces
- Sealed class improvements
- **`inline`** (aka value) classes

Kotlin 1.5.30 - August 24

- Preview: Exhaustive **`when`** statement
- Preview: Suspending functions as supertypes

Kotlin 1.6 - November 16

- Exhaustive **`when`** statement
- Suspending functions as supertypes
- Stable suspend conversions
- More improvements in type inference

Java 16 - March 16

- Preview 2: Sealed classes and interfaces
- Records
- Pattern matching for instanceof

Java 17 - September 14 - Third LTS!

- Sealed classes and interfaces
- Preview: Pattern matching for switch
-

2022

Kotlin 1.6.20 - April 4

- Preview: Context receivers for Kotlin/JVM
- Preview: Definitely non-nullable types

Kotlin 1.7 - June 9

- Preview: K2
- Implementation by delegation to inlined value of inline class
- Underscore operator for type arguments
- Builder inference
- Opt-in requirements
- Stable definitely non-nullable types

Kotlin 1.7.20

- Preview: `..<` for open-ended ranges
- Preview: `data objects`

Java 18 - March 22

- Preview 2: Pattern matching for `switch`

Java 19 - September 20

- Preview: Virtual Threads
- Preview: `record` patterns
- Preview 3: Pattern matching for `switch`

2023

Kotlin 1.8 - December 28

- Internal compiler work but no major features

Kotlin 1.8.20 - April 25

- Preview: K2
- Preview: Kotlin/Wasm
- Preview 2: **`data objects`**

Kotlin 1.9 - July 6

- `..<` operator for open-ended ranges

Java 20 - March 21

- Preview 2: **`record`** Patterns
- Preview 4: Pattern Matching for **`switch`**
- Preview: Scoped Values
- Preview 2: Virtual Threads
- Preview 2: Structured Concurrency

Java 21 - September 19 - Fourth LTS!

- **`record`** Patterns
- Pattern Matching for **`switch`**
- Virtual Threads
- Preview: **`String`** Templates
- Preview: Unnamed Patterns and Variables
- Preview: Implicitly Declared classes and Instance Main methods

2024

Kotlin 2.0 - May 21

- K2
- Smart cast improvements
- Multiplat improvements

Java 22 - March 19

- Unnamed Variables and Patterns
- Preview: Statements before `super(...)`
- Preview 2: `String` Templates
- Preview 2: Implicitly Declared classes and Instance Main methods
- Preview 2: Structured Concurrency
- Preview 2: Scoped Values

Java 23 - TBD!

- TBD

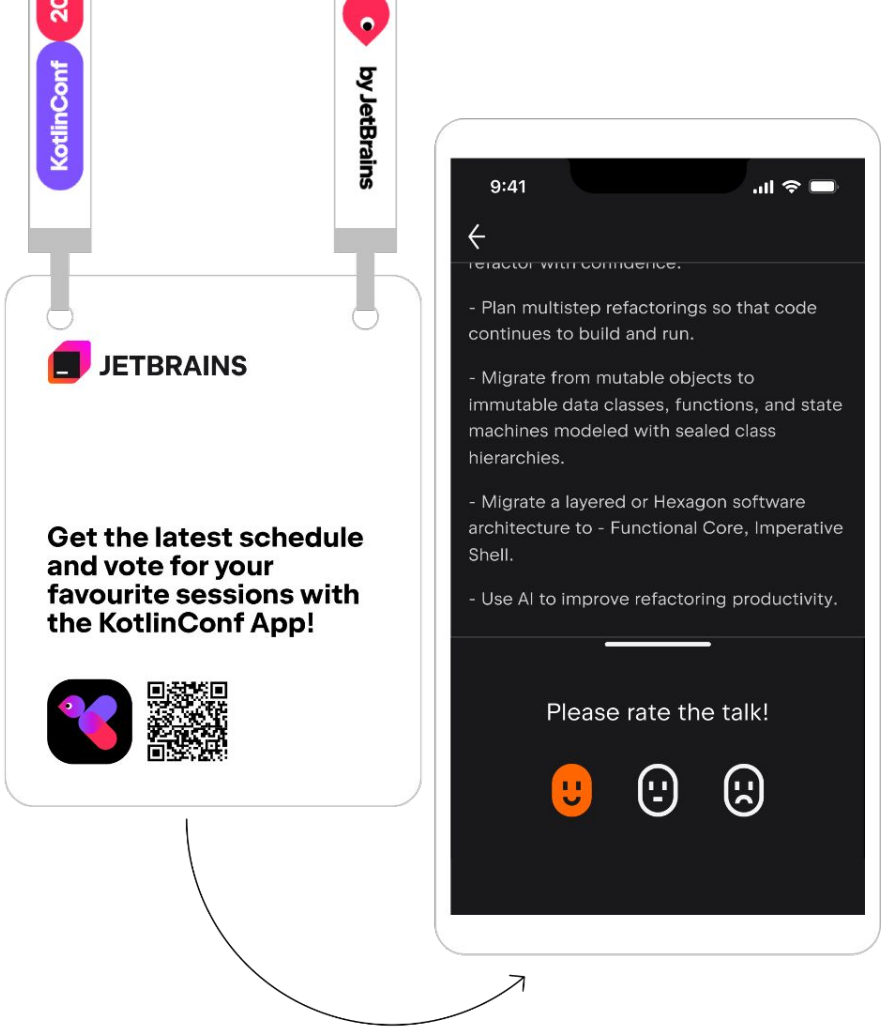
What does this all mean?

Wins and Advantage

- Kotlin: Features faster
- Kotlin: Multiplat support - JVM, Android, iOS, Web, Wasm
- Both: High degree of backward compatibility
- Java: Faster compilation
- Kotlin: coroutines, async flow, channels,
- Kotlin: DSLs
- Kotlin: named arguments

Kotlin’s features are pragmatic, developer friendly abstractions that make code more compact, easier to read and easier to write!

Thank you, and don’t forget to vote



A Tale of Two Languages

John Pampuch jpampuch@google.com