A BYTE OF KOTLIN IN A LAND FULL OF JAVA

IGOR KHVOSTENKOV

AUTHOR WHO ARE YOU?

name Igor Khvostenkov

twitter @IgorKhvostenkov

telegram @ikhvostenkov

company Lindenbaum GmbH

DISCLAIMER

- I am not affiliated with JetBrains
- I am not responsible for that you would like to try
- As well as for that you would not like
- I do not know everything, therefore I could be wrong or understand something in the wrong way, where other people understand things correct or do not do mistakes
- But I do my best

NOT ANDROID

HOW WAS MY FIRST TIME?

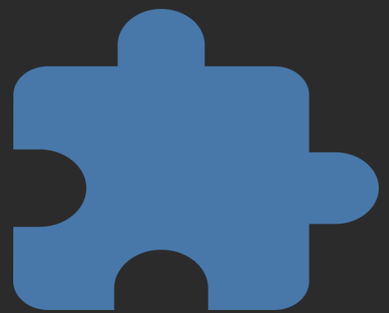
MAIN KOTLIN FEATURES

CONCISE

Drastically reduce the amount of boilerplate code

SAFE

Avoid entire classes of errors such as null pointer exceptions



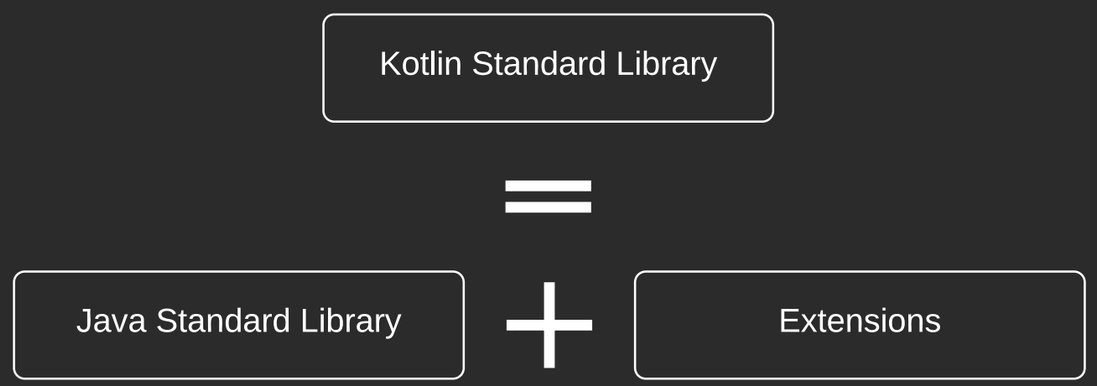
INTEROPERABLE

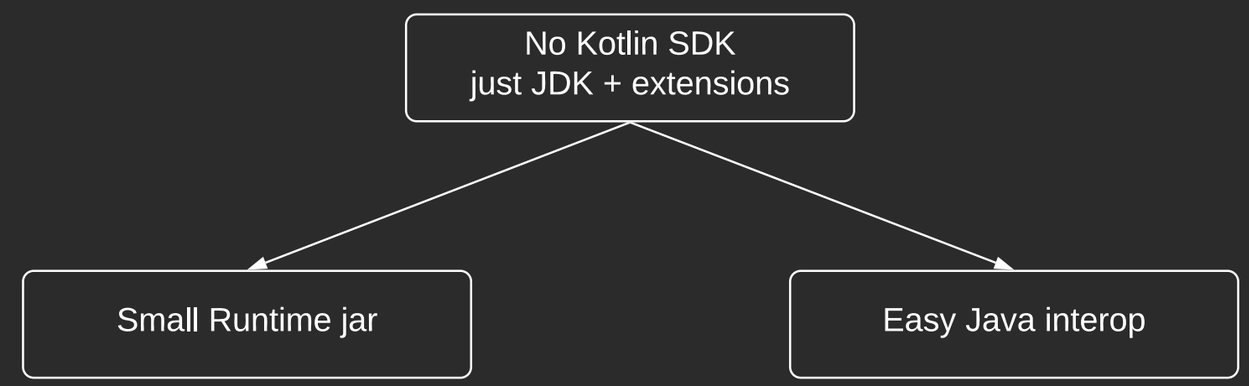
Leverage existing libraries for the JVM, Android and the Browser

TOOL-FRIENDLY

Choose any Java IDE or build from the command line

KOTLIN ECOSYSTEM





OUR JOURNEY WITH KOTLIN

2011 – JetBrains announces JVM-based Language

2013 – We start first experiments with Kotlin

2014 - We start using Kotlin for our test automation API

2016 – JetBrains releases Kotlin 1.0

2016 – We port our Kotlin code back to Java

2017 – Google announces first-class support for Android

2017 – We start deploying Kotlin on production systems

2019 – Kotlin has first-class status for our JVM components

WHAT Δ DO WE EXPECT?

null-safety

code is shorter

automatic type inference

great interoperability with Java

extension methods

ideal delegation

have fun doing the migration

DISADVANTAGES?

nullability noise !!

lost implicit widening conversions

no ternary operator

lost a number of IDE features

third party idiomaticity

```
companion object {
    val n = "noise"
}
```

CODE STYLE

Preferences

- Appearance & Behavior
- Keymap
- Editor
  - General
  - Font
  - Color Scheme
  - Code Style
  - Inspections
  - File and Code Templates
  - File Encodings
  - Live Templates
  - File Types
  - Copyright
  - Inlay Hints
  - Duplicates
  - Emmet
  - Images
  - Intentions
  - Language Injections
  - Spelling
  - TextMate Bundles
  - TODO
- Plugins 3
- Version Control
  - Background
  - Changelists
  - Commit Dialog
  - Confirmation
  - File Status Colors
  - Issue Navigation
  - Shelf
  - Git

Editor › Inspections For current project

Profile: Project Default Project

- Kotlin
  - Java interop issues
  - Migration
  - Naming conventions
  - Other problems
  - Probable bugs
  - Redundant constructs
  - Style issues
    - Accessor call that can be replaced with property access syntax
    - 'arrayOf' call can be replaced with array literal [...]
    - 'assert' call can be replaced with '!!' or '?:'
    - Assignment that can be replaced with operator assignment
    - Boolean expression that can be simplified
    - Boolean literal argument without parameter name
    - Call chain on collection could be converted into 'Sequence' to improve performance
    - Call chain on collection type can be simplified
    - Can be replaced with binary operator
    - Can be replaced with function reference
    - Can be replaced with lambda
    - Cascade if can be replaced with when
    - Class member can have 'private' visibility
    - Collection count can be converted to size
    - Convert manual ranges to indices or iteration over collection
    - Convert Pair constructor to 'to' function
    - Convert to primary constructor
    - Convert try / finally to use() call
    - Convert two comparisons to 'in'
    - 'copy' method of data class is called without named arguments
    - Equality check can be used instead of elvis for nullable boolean check
    - Explicit 'get' or 'set' call
    - Expression body syntax is preferable here
    - File is not formatted according to project settings

Disable new inspections by default

Description:

This inspection reports places that are not formatted according to project settings.

Severity: Weak Warning In All Scopes

Options

Apply only to modified files (for projects under a version control)

Cancel Apply OK

CODING CONVENTIONS

pom.xml

```
<properties>
  <kotlin.code.style>official</kotlin.code.style>
</properties>
```

gradle.properties

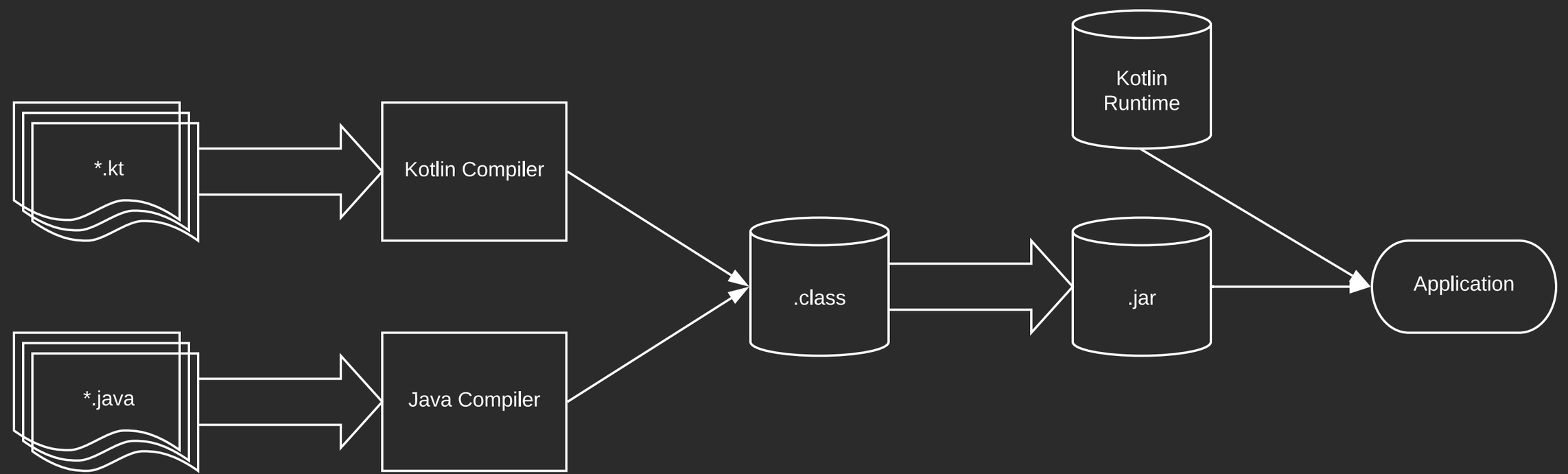
```
kotlin.code.style=official
```

CHECK STYLE

KTLINT

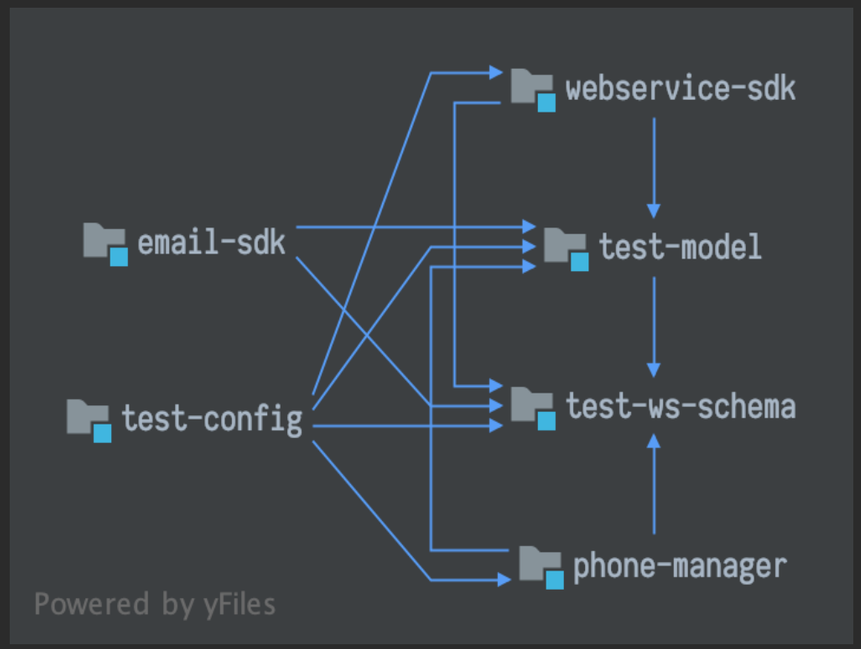
DETEKT

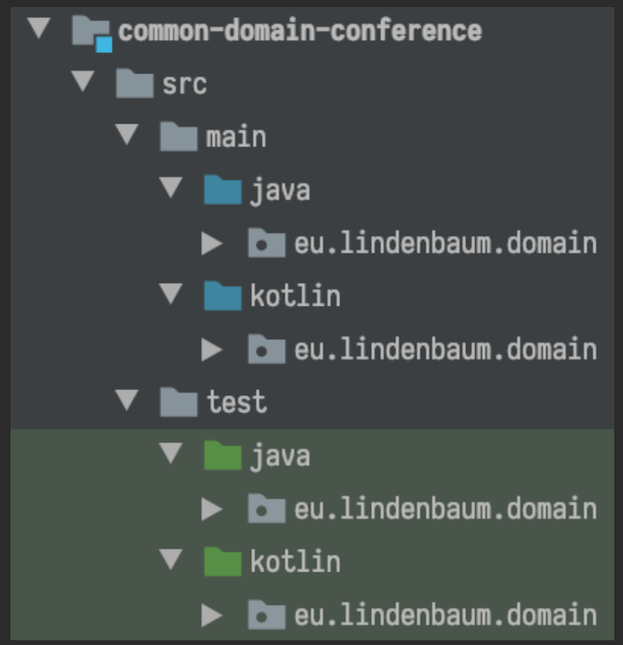
MIXING KOTLIN AND JAVA



STRATEGIES FOR ADOPTION

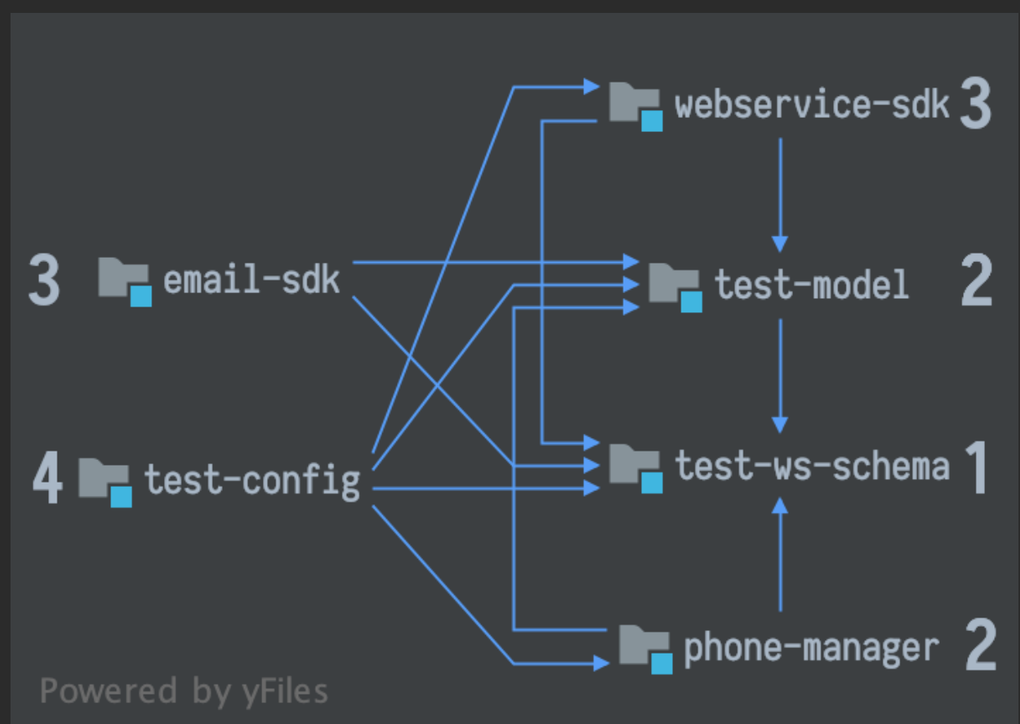
- Use Module Dependency Graph as a map
- Build tools: first invoke kotlinc, then javac
- Kotlin compiler knows when to invoke javac



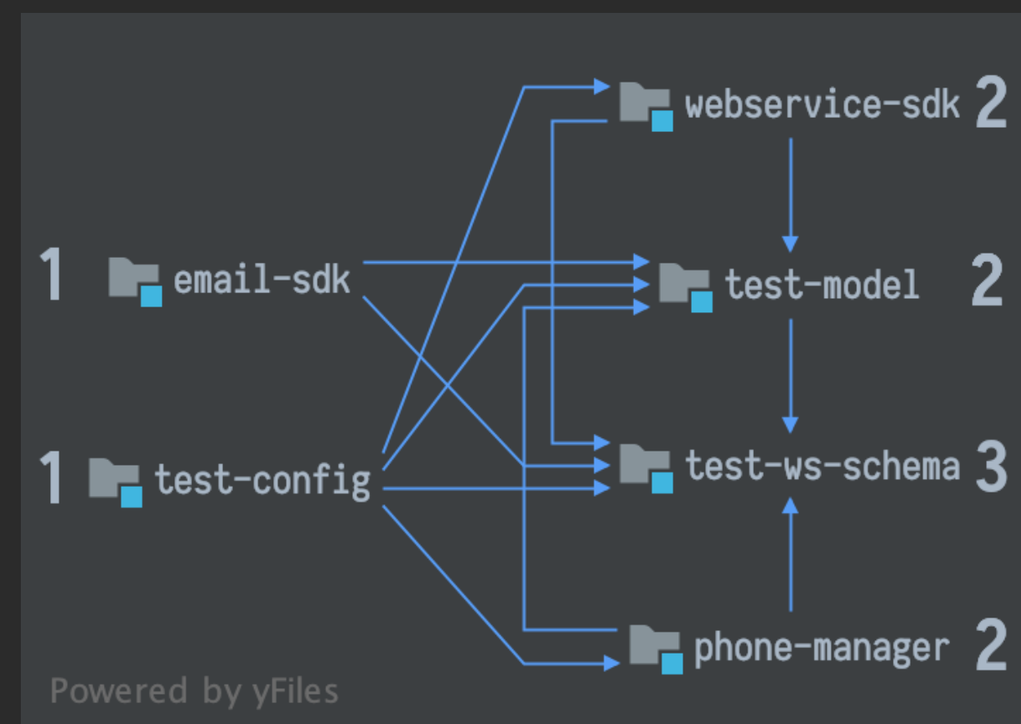


CONVERSION STRATEGIES

Incremental Inside-Out



Incremental Outside-In



JAVA TO KOTLIN CONVERTER

Preferences

Build, Execution, Deployment
Async Stack Traces
Remote Jar Repositories
Python Debugger
Deployment
Arquillian Containers
Application Servers
Clouds
Console
Coverage
Docker
Java Profiler
Jupyter
Required Plugins
Languages & Frameworks
JavaScript
PHP
Play Configuration
Python Template Languages
Schemas and DTDs
BDD
ColdFusion
Flask
Haskell
HOCON
JavaFX
Kotlin
Kotlin Scripting

Languages & Frameworks › Kotlin

Update channel: Stable Check again

Current Kotlin plugin version: 1.3.61-release-IJ2019.3-1

You have the latest version of the plugin installed.

Experimental Features

☑ Use New Java to Kotlin Converter

Cancel Apply OK

CONVERTING: FROM JAVA

Example.java

```java
public class Example {

    public void test() {
        Conference conference = new Conference();

        String conferenceState = conference.getState().name();

        if (conferenceState.equals(ConferenceState.ENDED.name())) {
            conference.getParticipants().forEach(p -> {
                if (!p.getAudioState().name().equals(AudioState.CONNECTED.nam
                    p.disconnect();
                }
            });
        }
    }
}
```

Participant.java

```java
blic class Participant {

  private AudioState audioState;

  public AudioState getAudioState() {
      return audioState;
  }
```

Conference.java

```java
ublic class Conference {
  private List<Participant> participants;
  private ConferenceState state;

  public List<Participant> getParticipants()
    return participants;
  }

  public ConferenceState getState() {
    return state;
  }
```

Example.kt

```kotlin
class Example {
  fun test() {
    val conference = Conference()
    val conferenceState = conference.state.name
    if (conferenceState == ConferenceState.ENDED.name) {
      conference.participants.forEach(Consumer { p: Participant -
        if (p.audioState.name != AudioState.CONNECTED.name) {
          p.disconnect()
        }
      })
    }
  }
}
```

CONVERTING: TO KOTLIN

Participant.java

```
blic class Participant {

private AudioState audioState;
@Nullable
public AudioState getAudioState() {
      return audioState;
  }
```

Conference.java

```
public class Conference {
  private List<Participant> participants;
  private ConferenceState state;
  @Nullable
  public List<Participant> getParticipants()
    return participants;
  }
  @Nullable
  public ConferenceState getState() {
    return state;
  }
```

Example.java

```
1  public class Example {
2
3      public void test() {
4          Conference conference = new Conference();
5
6          String conferenceState = conference.getState().name();
7
8          if (conferenceState.equals(ConferenceState.ENDED.name())) {
9              conference.getParticipants().forEach(p -> {
10                 if (!p.getAudioState().name().equals(AudioState.CONNECTED.nam
11                     p.disconnect();
12                 }
13             });
14         }
15     }
16 }
```

Example.kt

```
1  class Example {
2    fun test() {
3      val conference = Conference()
4      val conferenceState = conference.state!!.name
5      if (conferenceState == ConferenceState.ENDED.name) {
6        conference.participants!!.forEach(Consumer { p: Participant
7          if (p.audioState!!.name != AudioState.CONNECTED.name) {
8            p.disconnect()
9          }
10       })
11     }
12   }
13 }
```

TESTING IN KOTLIN

NAMING

Test.kt

```kt
@Test
fun `The very detailed and readable test name`() {
  assertThat("String", instanceOf(String::class.java))
}
```

KOTLIN + JUNIT

Test.kt

```kotlin
companion object {
  @JvmStatic @BeforeClass
  fun setUp() {}

  @ClassRule @JvmField
  var resource: ExternalResource = object : ExternalResource() {
    override fun before() {
      conference.connect()
    }

    override fun after() {
      conference.disconnect()
    }
  }
}

@Rule @JvmField
var rule = TemporaryConference()
```

KOTLIN + MOCKITO

Test.kt

```kotlin
import org.mockito.Mockito.`when` as on


inline fun <reified T> kotlinAny(): T = kotlinAny(T::class.java)
inline fun <reified T> kotlinAny(t: Class<T>): T = Mockito.any<T>(t)
```

THE CHAD MOCKITO

MockK.kt

```
val car = mockk<Car>()

every { car.drive(Direction.NORTH) } returns Outcome.OK

car.drive(Direction.NORTH) // returns OK

verify { car.drive(Direction.NORTH) }

confirmVerified(car)
```

PROBLEM OF THE NO-ARGS CONSTRUCTORS IN JAVA

pom.markdown

```
<configuration>
      <compilerplugins>
          <!-- Or "jpa" for JPA support -->
          <plugin>no-arg</plugin>
      </compilerplugins>

      <pluginoptions>
          <option>no-arg:annotation=com.my.Annotation</option>
        <!-- Call instance initializers in the synthetic constructor -->
        <!-- <option>no-arg:invokeInitializers=true</option> -->
      </pluginoptions>
</configuration>
<dependencies>
      <dependency>
          <groupid>org.jetbrains.kotlin</groupid>
          <artifactid>kotlin-maven-noarg</artifactid>
          <version>${kotlin.version}</version>
      </dependency>
</dependencies>
```

gradle.properties

```
buildscript {
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-noarg:$kotlin_version"
    }
}

apply plugin: "kotlin-noarg"

noArg {
    annotation("com.my.Annotation")
}
```

PROBLEM OF THE FINAL CLASSES IN KOTLIN

pom.xml

```xml
<configuration>
      <compilerplugins>
          <!-- Or "spring" for the Spring support -->
          <plugin>all-open</plugin>
      </compilerplugins>

      <pluginoptions>
          <!-- Each annotation is placed on its own line -->
          <option>all-open:annotation=com.my.Annotation</option>
          <option>all-open:annotation=com.their.AnotherAnnotation</option>
      </pluginoptions>
</configuration>

<dependencies>
      <dependency>
          <groupid>org.jetbrains.kotlin</groupid>
          <artifactid>kotlin-maven-allopen</artifactid>
          <version>${kotlin.version}</version>
      </dependency>
</dependencies>
```

gradle.properties

```gradle
buildscript {
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-allopen:$kotlin_version"
    }
}


apply plugin: "kotlin-allopen"

allOpen {
    annotation("com.my.Annotation")
    // annotations("com.another.Annotation", "com.third.Annotation")
}
```

SOME LANGUAGE FEATURES

ParticipantDetails.java

```java
public interface ParticipantDetails {
  String getName();
  String getLastName();
}
```

SOME LANGUAGE FEATURES

ParticipantDetails.java

```java
public interface ParticipantDetails {
  String getName();
  String getLastName();
}
```

Participant.kt

```kotlin
data class Participant(private val name: String,
                       private val lastName: String) : Conference.ParticipantDetails {
  override fun getName(): String {
    return name;
  }

  override fun getLastName(): String {
    return lastName;


  }
}
```

WORKING WITH ANNOTATIONS

Moderator.kt

```
1 data class Moderator(
2     @NotNull
3     val name: String,
4     @Active
5     val conference: Conference
6                     ) {
7 }
```

WORKING WITH ANNOTATIONS

Moderator.kt

```
data class Moderator(
    @NotNull
    val name: String,
    @Active
    val conference: Conference
                    ) {
}
```

Moderator.kt

```
data class Moderator(
    @field:NotNull
    val name: String,
    @field:Active
    val conference: Conference
                    ) {
}
```

JAVA + LOMBOK = KOTLIN

@Getters	Properties
@Setters	
@NonNull	Nullable Types
val/var	val/var

JAVA + LOMBOK. DELOMBOKING

@Equals

@HashCode

@ToString

@Synchronized

CAVEATS FOR JAVA DEVELOPERS

ConferenceCall.kt

```kotlin
class ConferenceCall {
  val id: Int = Random.nextInt(0, 100)
}
```

ConferenceFactoryTest.kt

```kotlin
fun getConference() {
  val conferenceCall = ConferenceCall()
  for (i in 1..10) {
    println("Conference Call ID: ${conferenceCall.id}")
  }
}
```

CAVEATS FOR JAVA DEVELOPERS

ConferenceCall.kt

```kt
class ConferenceCall {
  val id: Int
    get() {
      return Random.nextInt(0, 100)
    }
}
```

ConferenceFactoryTest.kt

```kt
fun getConference() {
  val conferenceCall = ConferenceCall()
  for (i in 1..10) {
    println("Conference Call ID: ${conferenceCall.id}")
  }
}
```

HUMAN FACTORS: REJECTION

Unusual Syntax

Feels Bad

Exotic Language

Java is a Safe Bet

HYPE!

How to find Kotlin devs?

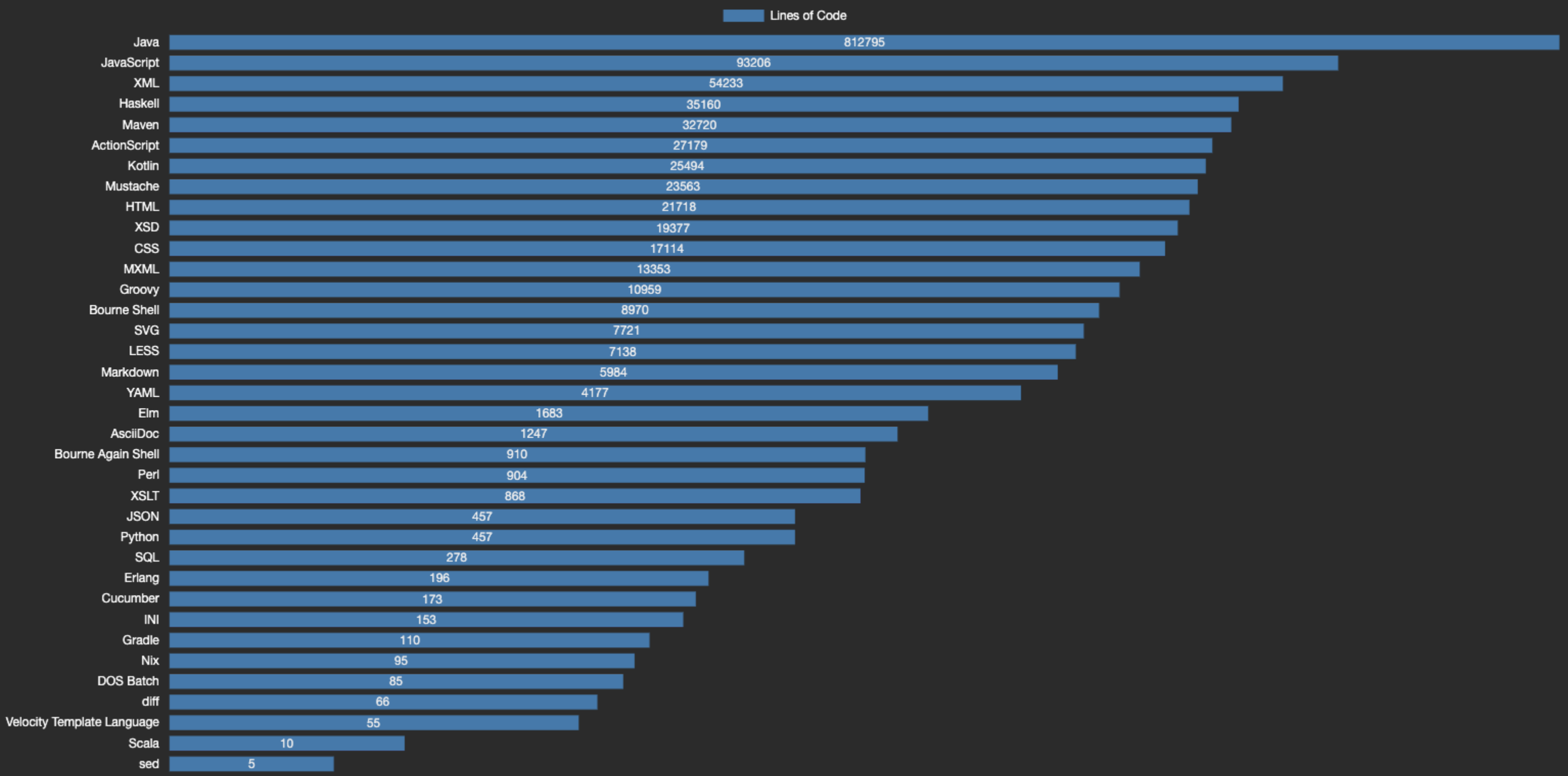
It's not purely functional

Tomorrow it will be gone

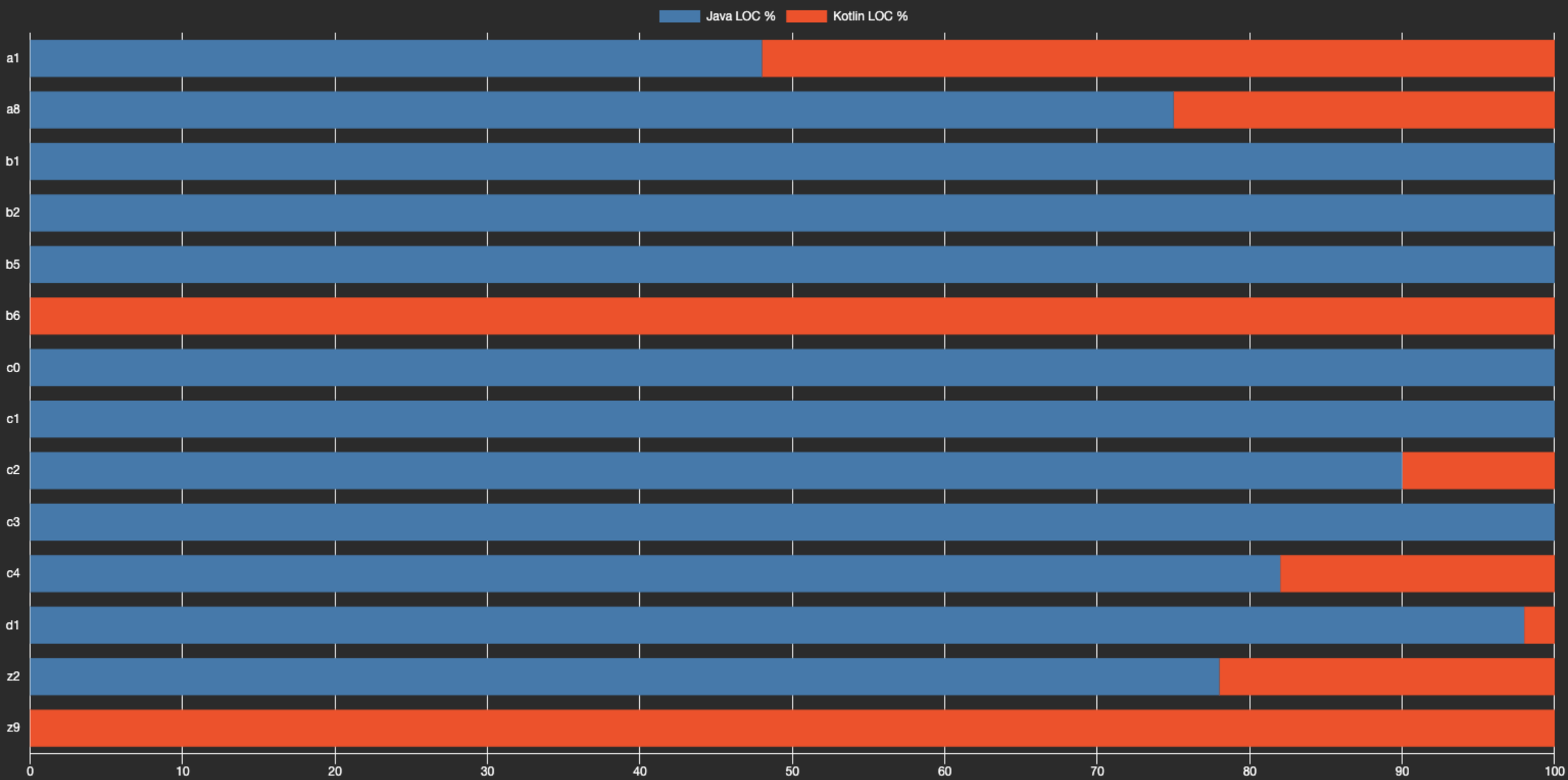
HUMAN FACTORS: APPROACHING

- Avoid the “coolness factor” argument
- Focus on language semantics and consistency
- Amplify Safety, Correctness and Maintainability
- Strive for gradual Evolution, not a Revolution
- Remind of very good interoperability with Java
- Remind of good tooling improving over time
- Start on small modules with few users or test-only
- Take care of up-to date tooling and IDE configuration

OVERALL CODE BASE



JAVA + KOTLIN


Java LOC %
Kotlin LOC %
a1
a8
b1
b2
b5
b6
c0
c1
c2
c3
c4
d1
z2
z9
0
10
20
30
40
50
60
70
80
90
100

KOTLIN TOOLING IN 2020

- Kotlin is built with Tooling in mind from the ground up
- IDEs: Good with IntelliJ and Android Studio
- Yet, refactorings are not as rich as for Java
- But steadily improving in JetBrains IDEs
- Caveat: Not much for other IDEs available
- Certain vendor Lock-In inevitable as of today
- Very good support in Maven and Gradle

CONCLUSION

- There is no free lunch, but Kotlin tastes good
- Especially suitable for users of JetBrains IDEs
- Very good interop with Java and small footprint
- It is a good trade-off with significant advantages
- More safety, more solid design, less boilerplate
- Applicable for new and legacy codebase

BUT THIS NUMBER DID NOT CHANGE ≈ 10 YEARS