A byte of Kotlin

In a land full of Java

Rusi Filipov and **Igor Khvostenkov**

Lindenbaum GmbH

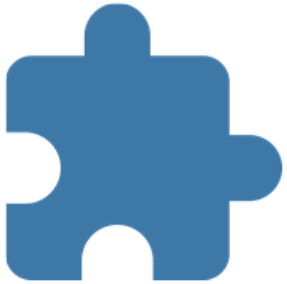
Main Advantages of Kotlin

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

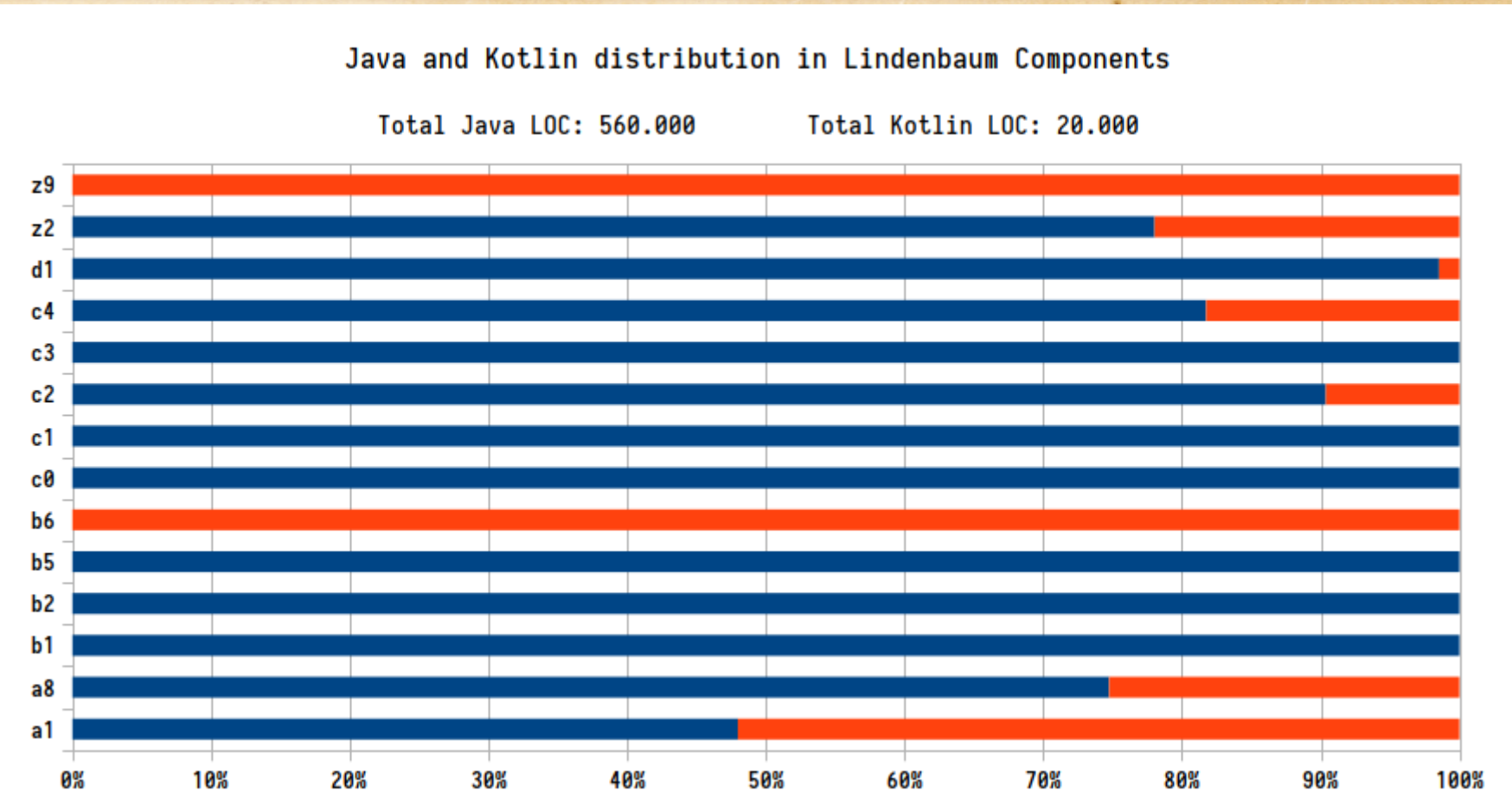
Tool-friendly

Choose any Java IDE or build from the command line

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

Our Java+Kotlin Mix



Why Kotlin? There comes Java-13!

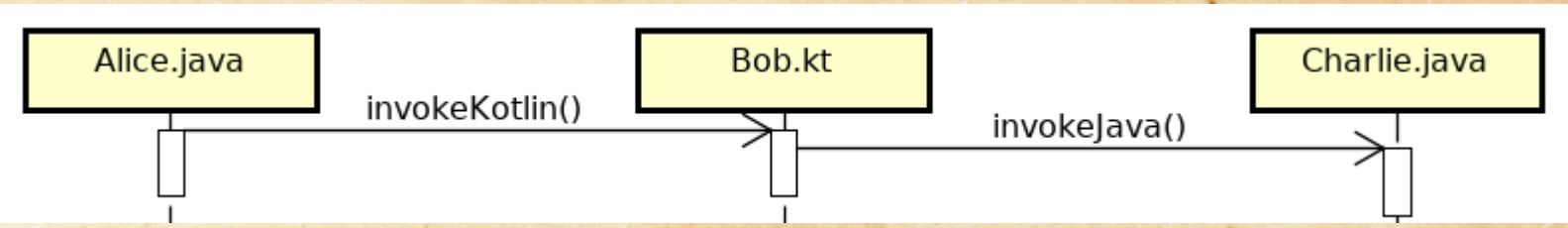
- Superior semantics built-into the language
- Data Classes, Sealed Classes, Immutable Collections
- Null-Safety and Checked Nullables
- Internal Visibility, Sane Generics
- Functional Programming Friendly
- Excellent Standard Library
- Also suitable for people stuck with JVM-1.6

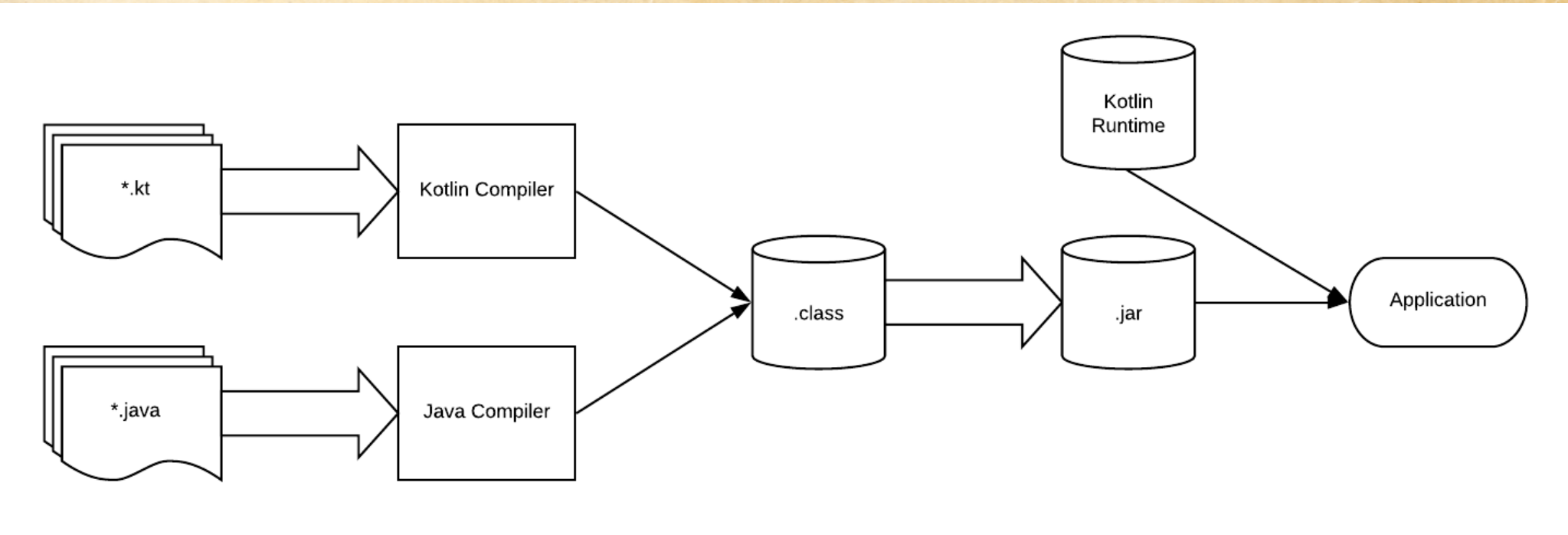
Built-In Immutability Concepts

```
data class MailRecipient(val userId: Int, val address: String) {
  init {
    require(address.matches(Regex( pattern: ".+@.+")))
  }
}

fun normalize(input: MailRecipient): MailRecipient =
    input.copy(address = input.address.toLowerCase())

fun main() {
  val recipient = MailRecipient( userId: 1,  address: "Alice@example.com")
  println(normalize(recipient))
}
```

```
output: MailRecipient(userId=1, address=alice@example.com)
```

Mixing Kotlin and Java



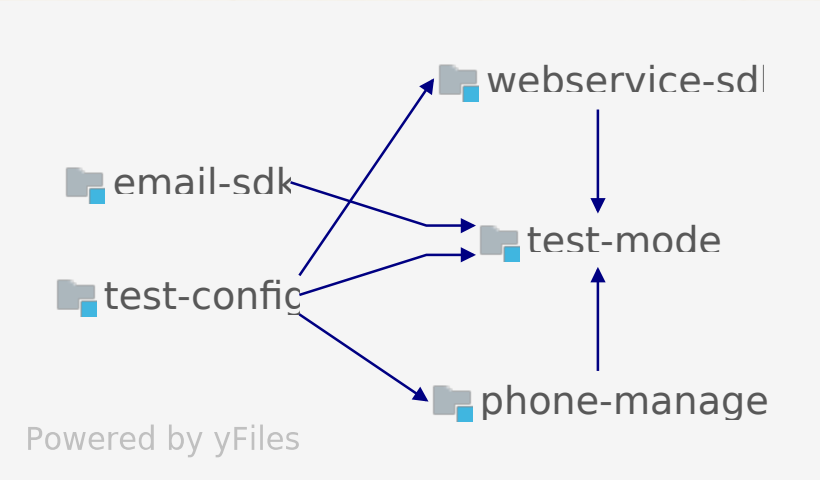


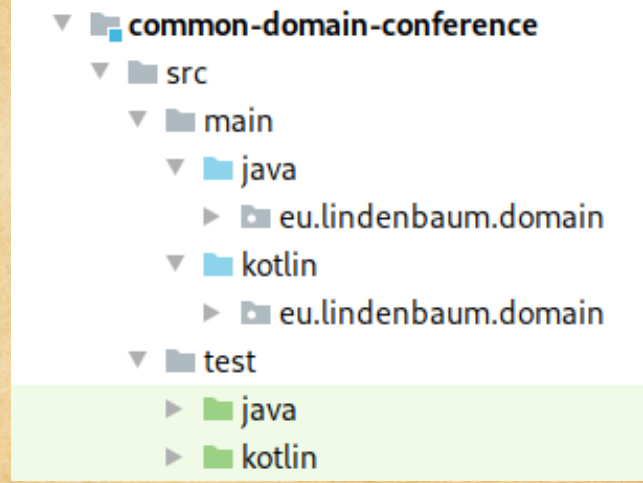
What about Groovy, Scala, Clojure?

- We prefer static type systems - for tooling, safety and speed
- Groovy's typing is inferior to both Kotlin and Scala
- Clojure is solely FP oriented, while Kotlin is a OO/FP hybrid
- But Scala is a OO/FP hybrid too
- Scala devs view Kotlin as "just a better Java"
- Indeed, Scala is more feature-rich (more powerful)
- Has an advanced type system and is more FP affine
- But Scala is more complex - for both programmers and tools

Strategies for Adoption

- Use your Module Dependency Graph as a map
- Build tools first invoke kotlinc, then javac
- But the Kotlin Compiler knows when to invoke javac



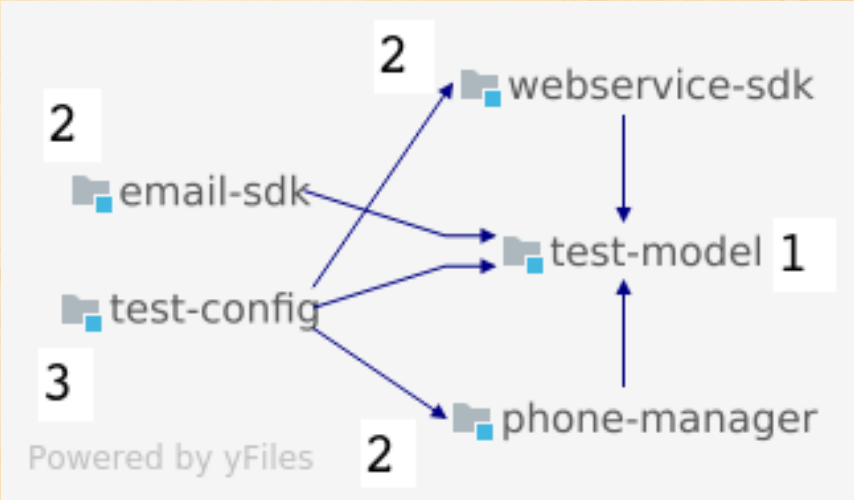


Conversion Strategies

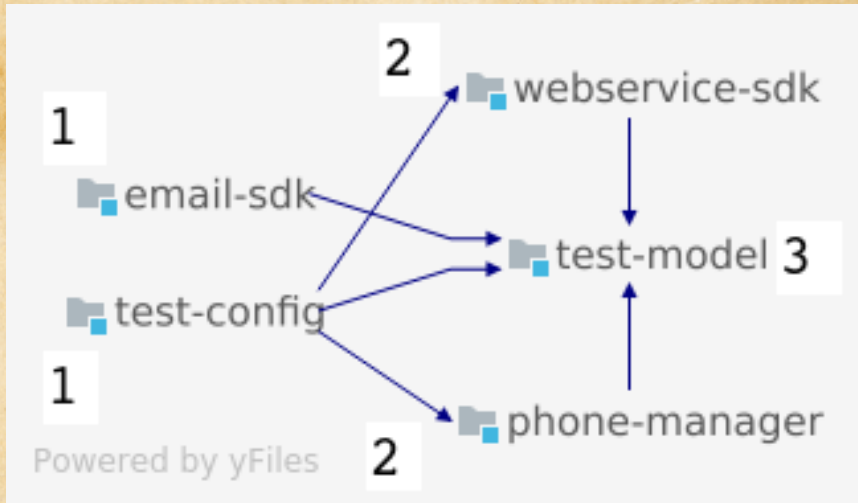
- Classes and modules can be incrementally converted
- The order of conversion has impact on the progress
- Inside-Out: for max strictness (if you are brave)
- Outside-In: for lower risk (if you are deffensive)
- Random and Wild: context driven (non-systematic)
- But be careful with non-null parameters after conversion
- For Java callers might still be passing null (runtime error)
- For Starters: Convert your test code first

Inside-Out vs Outside-In

- Incremental Inside-Out



- Incremental Outside-In



Runtime Dependencies

- The Kotlin Standard Library (Kotlin Runtime)
- Just a few small jars with additional types and functions
- Packaged as regular dependencies of the application
- No SDK, no native binaries, no installation required
- Kotlin stdlib includes a Collections library (FP style)
- Immutable collections by default, map(), fold() etc.
- Also useful extensions to the Java stdlib (similar to Guava)

Converting: From Java

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

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

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

```java
public class Participant {

  private AudioState audioState;

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

```java
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.name())) {
            p.disconnect();
        }
    });
}
```

Converting: To Kotlin

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

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

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

```java
public class Participant {

    private AudioState audioState;

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

```kotlin
val conference = Conference()

val conferenceState = conference.state!!.name

if (conferenceState == ConferenceState.ENDED.name) {
  conference.participants!!.forEach { p : Participant! ->
    if (p.audioState!!.name != AudioState.CONNECTED.name) {
      p.disconnect()
    }
  }
}
```

Our Adoption

- Initially, we wrote only tests and our test APIs in Kotlin.
- Then we introduced a small shared Kotlin module for multiple Java modules.
- This typically meant using Kotlin code from Java.
- Up until then, we were introducing Kotlin strictly inside-out.
- Later, we started converting small data classes, interfaces and enums and to Kotlin, on a feature-by-feature basis.
- We learned, that a 100% conversion of the whole module (.jar) is preferable.
- Hybrid Java+Kotlin modules work, but present refactoring issues.

Human Factors: Rejection

- Eclipse Users
- Scala Fans
- Java Fans
- “Unusual Syntax”
- “Feels Bad”
- “Exotic Language”
- “Java is a Safe Bet”
- “How to find Kotlin devs?”
- “Language is just a tool.”
- “You don’t really need it.”
- “Java is just as good.”
- “This is just a hype.”
- ”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 interopability 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

Caveats for Java Developers

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

```
val conferenceCall = ConferenceCall()
repeat( times: 3) { println("ID: ${conferenceCall.id}") }
```

```
ID: 27
ID: 27
ID: 27
```

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

```
ID: 17
ID: 31
ID: 47
```

Kotlin Tooling in 2019

- 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 codebases