Kotlin is just an island too

Kotlin, Multiplatform & Compose

13.03.2025, Java User Group Stuttgart, Germany
by Dr. Michael Paus

Background

About me

- Dr. Michael Paus (mp@jugs.org)
- Aerospace engineer from Stuttgart
- Developer, Consultant, Compose- and JavaFX-enthusiast
- OpenJFX author
- Chairman of the Java User Group Stuttgart e.V. (Organizers of the annual Java Forum Stuttgart)

Agenda

- The language
- Development tools
- Kotlin Multiplatform (KMP)
- Compose Multiplatform (CMP)

The language

Entry points

- https://kotlinlang.org/community/
- https://kotlinlang.slack.com/

Which features are particularly noteworthy?

Null-Safety

- Comparison with Java
- Interaction with other language features
  - Elvis-operator ?:
  - ?.let{}
  - if (x != null) { x.doSomething() }

Data classes

- equals()/hashCode() pair.
- toString() of the form "User(name=John, age=42)".
- componentN() functions corresponding to the properties in their order of declaration.
- copy()

Immutable data

- List vs. MutableList
- https://github.com/Kotlin/kotlinx.collections.immutable

Serialization

- JSON: kotlinx-serialization-json
- Protocol buffers: kotlinx-serialization-protobuf
- CBOR: kotlinx-serialization-cbor
- Properties: kotlinx-serialization-properties
- HOCON: kotlinx-serialization-hocon (only on JVM)

- Custom serializers ?
- Performance ?

Coroutines

- Asynchronous or non-blocking programming
- One language keyword suspend. Mostly handled via libraries. (kotlinx-coroutines-*)
- A topic which can be confusing and difficult to grasp.
  - Distinction between normal and suspend functions
    (e.g., no call to suspend functions inside constructor.)
  - Mapping from coroutines to system threads (e.g., no Dispatchers.IO in Wasm)
- Use well established patterns to avoid erroneous programs.

Development tools

- Kotlin compiler (Jetbrains)
- Build
  - Gradle (Maven not well supported)
  - Amper (https://www.jetbrains.com/help/kotlin-multiplatform-dev/amper.html)
- IDE IntelliJ/Android Studio, (Fleet), ???
- Eclipse plugin not very well supported and cannot be recommended.

Kotlin Multiplatform (KMP)

- Kotlin-(cross-)compilers for many platforms
- Reuse Kotlin code across Android, iOS, web, desktop, and server-side while keeping native code if needed.
- Google officially endorses KMP to share business logic across mobile, web, server, and desktop.
- https://blog.doubleslash.de/software-technologien/coding-and-frameworks/cross-platform-entwicklung-kotlin-oder-compose-multiplatform-ein-direkter-vergleich

Platforms

- JVM
- JS
- Wasm (Browser and WASI)
- Native

- Tier 1:
- macosX64
- macosArm64
- iosSimulatorArm64
- iosX64
- iosArm64

Platforms

- Tier 2:
- linuxX64
- linuxArm64
- watchosSimulatorArm64
- watchosX64
- watchosArm32
- watchosArm64
- tvosSimulatorArm64
- tvosX64
- tvosArm64

- Tier 3:
- androidNativeArm32
- androidNativeArm64
- androidNativeX86
- androidNativeX64
- mingwX64
- watchosDeviceArm64

Usage

- Devide source code into common and platform specific directories.
- Access platform specific code via expect/actual.

```
// common Code (commonMain)
expect fun getPlatformName(): String

// android specific implementation (androidMain)
actual fun getPlatformName(): String = "Android"

// iOS specific implementation (iosMain)
actual fun getPlatformName(): String = "iOS"
```

- Cross-compile for all configured platforms.
- Creates separate artefact for each configured platform.
- Only possible for all platforms from macOS.

Problems

- Complex, lengthy build process
- Only works nicely when all code is in Kotlin. (No JVM dependencies)

Multiplatform libraries

- Find replacement libraries

- https://klibs.io/
- https://github.com/terrakok/kmp-awesome

- Key libraries
  - Ktor - Http-Server/Client
  - Okio - IO
  - Koin - Dependency injection
  - SqlDelight - Database frontend
  - Kotlinx-serialization - Serialization
  - Kotlin-logging - Logging
  - Kotlinx-datetime - Date/time library
  - Multiplatform-settings - Settings store
  - Kotlinx-collections-immutable - Immutable collection library
  - ...

- Important to watch out for missing features for one of your desired platforms.

Filling the gaps

- Create various implementations with different techniques.
  - Reference
    - Reference solution via Java
  - Java to Kotlin conversion
    - Convert manually.
    - Convert via J2K (manually remove Java dependencies).
  - C or native compilation
    - Transpile Java reference solution to C via TeaVM and link to application.
    - Compile Java reference solution to shared library via GraalVM/native image (if that is still supported for iOS via Gluon tooling)and link to application.
    - Compile Java reference solution to shared library via OpenJDK/Mobile (once available) and link to application.
    - Compile Java reference solution to shared library via J2ObjC and link to application.
  - Wasm/Js
    - Transpile Java reference solution to Wasm via TeaVM and run via embedded Wasm runtime.
      - Chasm (pure Kotlin), Chicory (only JVM), GraalWasm (only JVM), …
    - Run (without transpiling) in browser via CheerpJ. +
      - See: (for interfacing) https://kotlinlang.slack.com/archives/CDFP59223/p1735745154663509?thread_ts=1735518020.308579&cid=CDFP59223
    - Transpile via new GraalVM/native-image Wasm backend +
      - https://2025.wasm.io/sessions/the-future-of-write-once-run-anywhere-from-java-to-webassembly/
  - Don't use Java at all. Use Rust which may also wrap some C.

Compose Multiplatform (CMP)

- Compose Multiplatform, a modern UI framework for Kotlin that makes building performant and beautiful user interfaces easy and enjoyable.

- https://github.com/JetBrains/compose-multiplatform

Background

- Developed by Google as the new standard for native Kotlin Android apps.
- Extended by Jetbrains to be used with Kotlin multiplatform.
- Supports Desktop (macOS, Windows, Linux), Mobile (Android and iOS) and Web (JS and Wasm)

What's the difference?

- Supported by two companies who know what they are doing and who also use it for their own products.
- All code is under the truly OSS Apache license.
- Multiplatform use is actively supported by Google.
- Covers platforms (iOS and Web) where Java never succeeded.

Usage

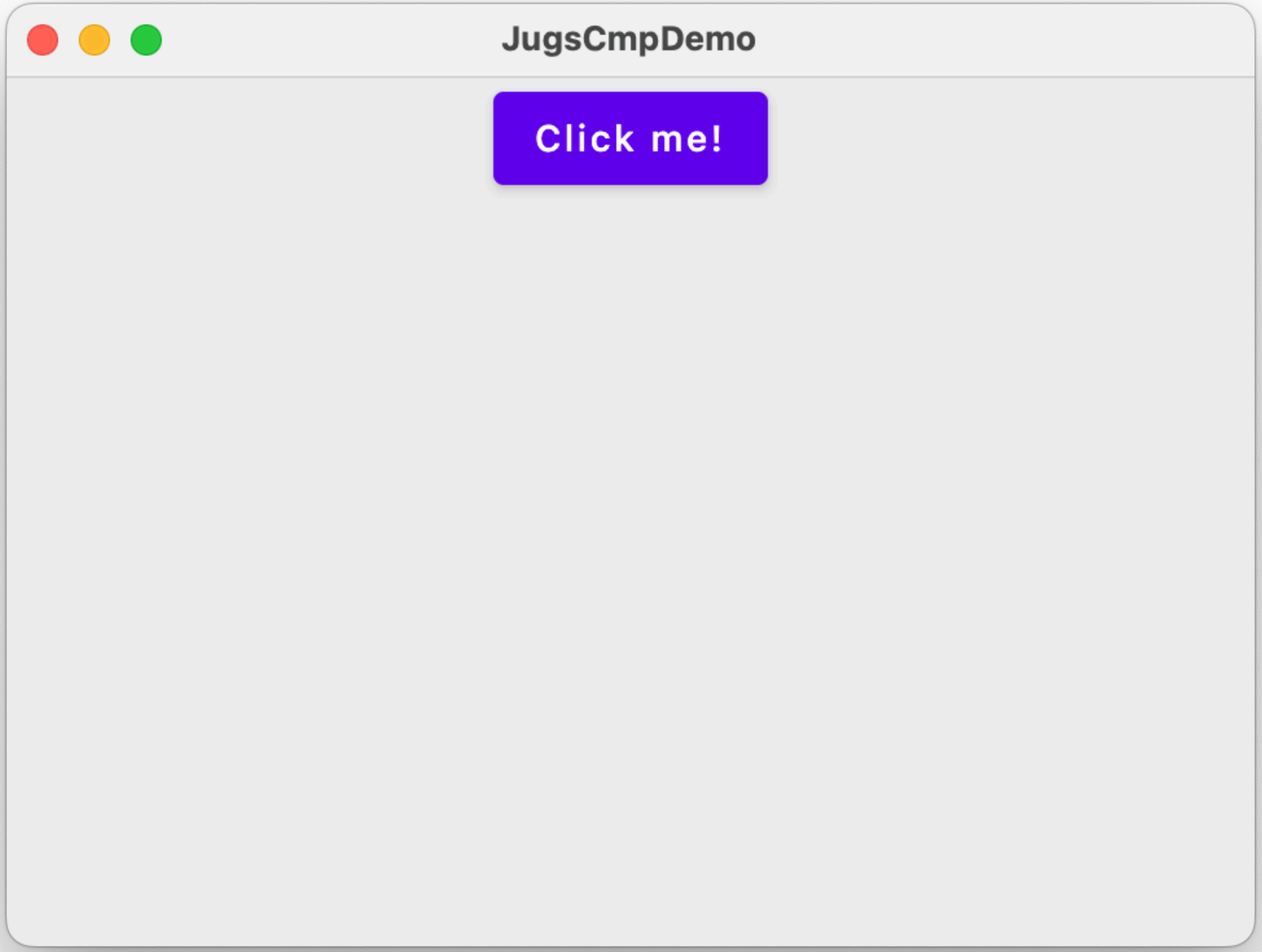
- Create a template project with a wizard.
  - https://kmp.jetbrains.com/
  - https://terrakok.github.io/Compose-Multiplatform-Wizard/

- Examples
  - JugsCmpDemo
  - https://jfspwa.java-forum-stuttgart.de
    - Offline-capable PWA
    - Works only with latest browsers!

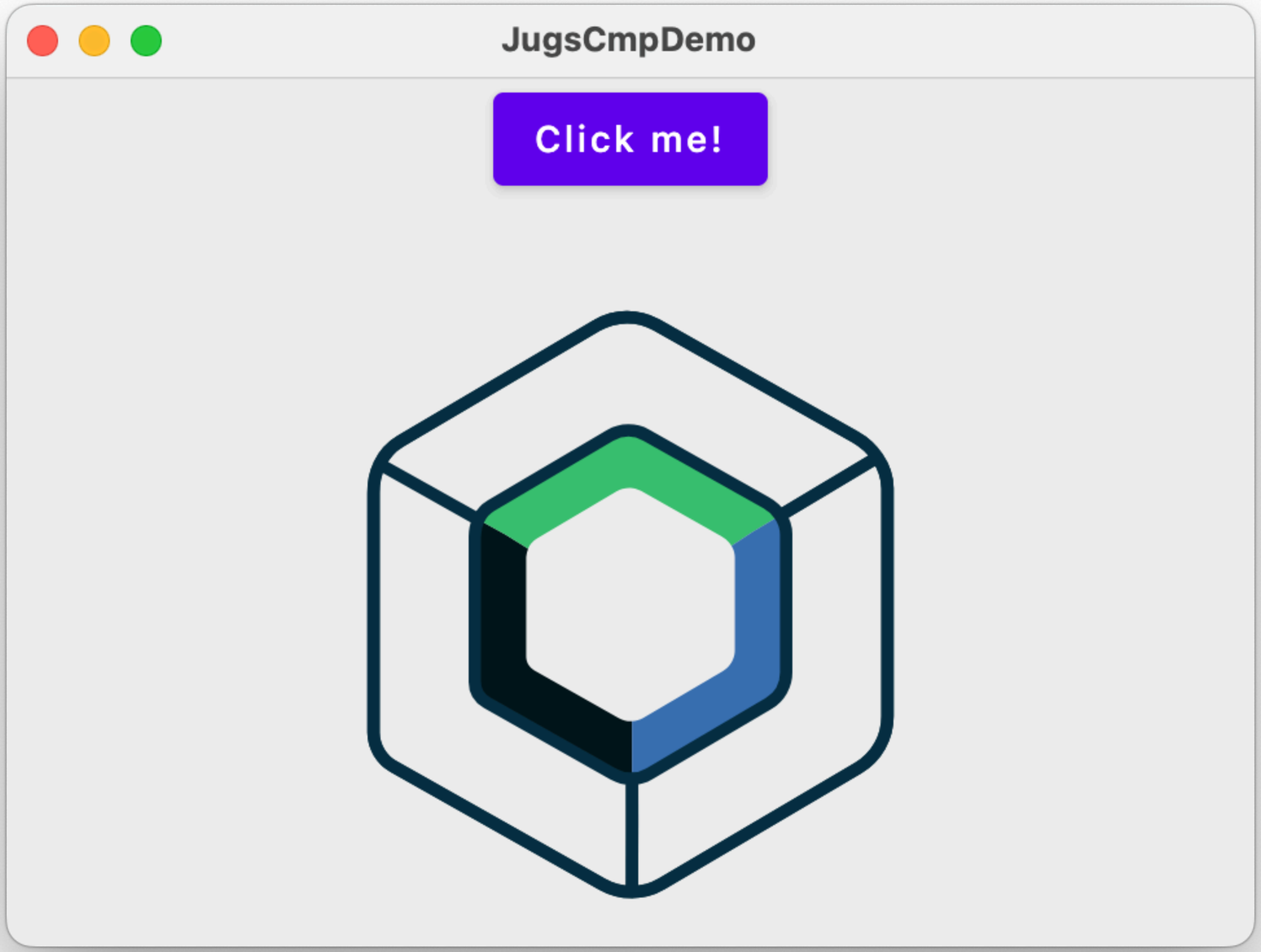
Example

```
1   package org.jugs.cmp
2
3   import ...
16
17  @Composable
18  fun App() {...}
25
26  @Composable
27  fun Page(greeting: String, showContent: Boolean, onShowContentChange: () -> Unit) {
28      Column(Modifier.fillMaxWidth(), horizontalAlignment = Alignment.CenterHorizontally) {
29          Button(onClick = onShowContentChange) {
30              Text("Click me!")
31          }
32          if (showContent) {
33              Image(painterResource(Res.drawable.compose_multiplatform), null)
34              Text("Compose: $greeting")
35          }
36      }
37  }
```

Example



Example



Things to watch out for

- Platform dependencies
  - No Android dependencies
  - No JVM dependencies
  - Unsupported platforms of libraries
  - Unimplemented features of libraries

- Conceptional differences between platforms
  - No arbitrary file system access (sandbox).
  - No file system at all. (Web)
    - LocalStorage, OPFS, IndexedDB, in-memory file system, etc. instead
  - No multithreading (Web)
    - No threads. Have to use workers instead.
    - No Dispatcher.IO.
    - No runBlocking().
  - Cross-Origin Resource Sharing (CORS)
  - Different concepts for Back-Button handling.

Questions