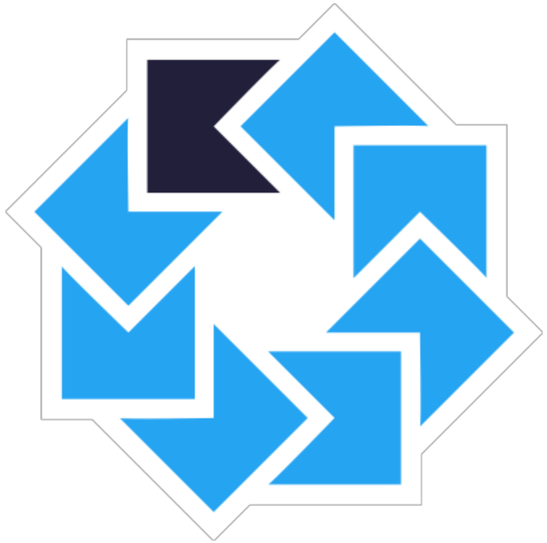
#5: DSL & Multiplatform



SPACE

juno

School.kt Program

0. Intro
1. Object-oriented programming
2. Standard library
3. Functional programming
4. Generics
5. **Kotlin DSL & Multiplatform projects**
6. Coroutines
7. Interoperability with Java
8. Kotlin ecosystem

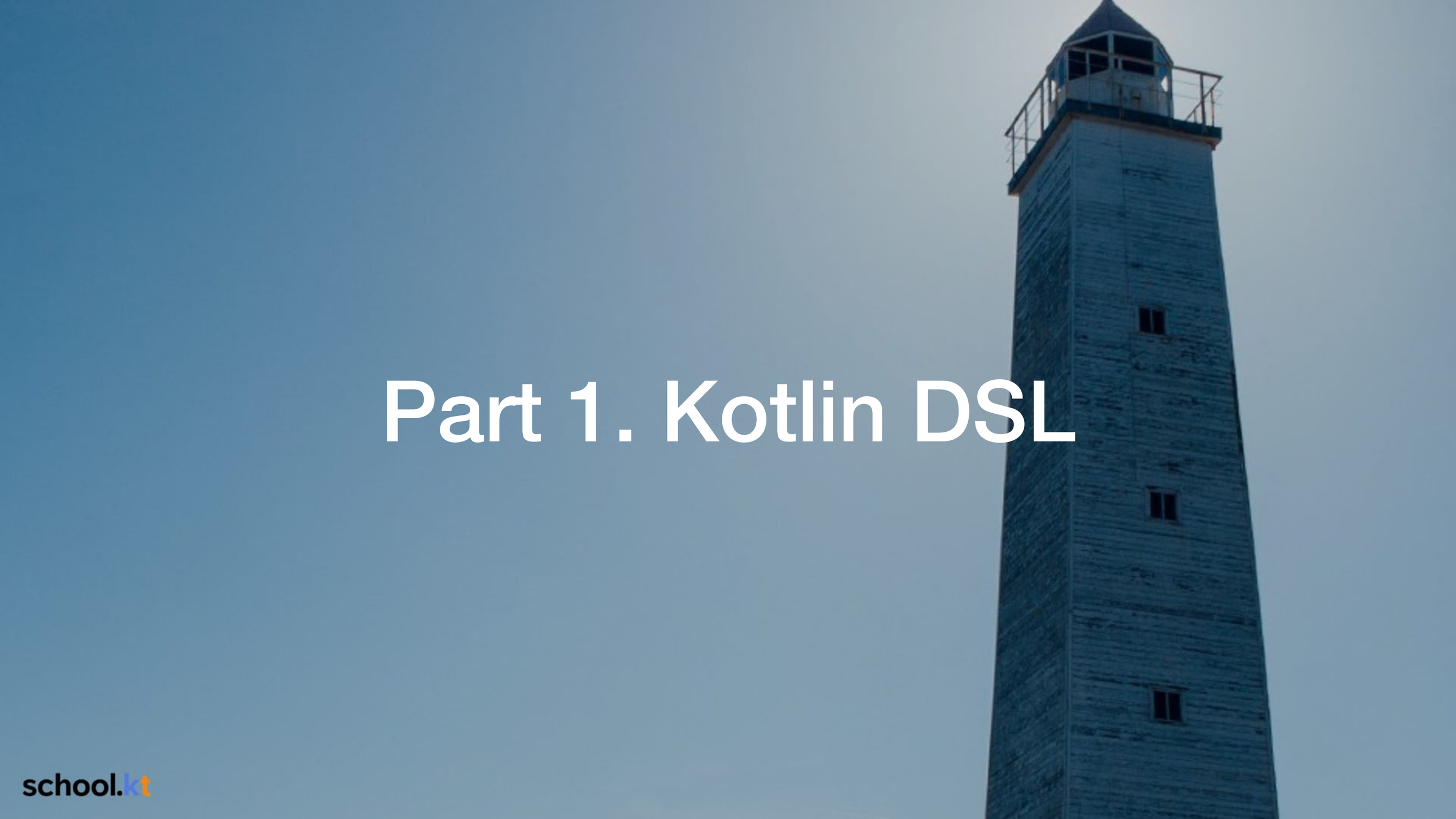

Mentors

5p1. Kotlin DSL

- What is DSL?
- Kotlin DSL
- How to build own DSL

5p2. Multiplatform Projects

- Modern way of reusing code between platforms
- Kotlin Multiplatform Projects idea
- Main principles
- Sample



Part 1. Kotlin DSL

DSL

A **domain-specific language (DSL)** is a computer language specialised to a particular application domain. DSL is opposite for general-purpose language (GPL), which is broadly applicable across domains

wikipedia.org/wiki/Domain-specific_language

Examples of DSL Languages

- SQL
- HTML
- CSS
- Regular expressions

DSL positives

+ Stricter API for specific domain
+ Better solve special tasks

DSL negatives

- Non-trivial to validate the correct interaction of the DSL with the host language at compile time
  Harder to debug the DSL program and to provide IDE code completion
- Can be difficult to combine with a host application in a GPL
  Need to either store program written in DSL in a separate file or embed it in a string literal
- The separate syntax requires separate learning

As opposed to *external DSL*s, which have their own independent syntax, **internal DSL**s are part of programs written in a *general-purpose language*, using exactly the same syntax

DSL Sample .html

```
<!DOCTYPE html>
<html>
    <head>
        <title>This is a title</title>
    </head>
    <body>
        <p>Hello</p>
        <p>world!</p>
    </body>
</html>
```

DSL Sample .java

DSL Sample .java

```
HtmlBuilder htmlBuilder = createHtml();
htmlBuilder.head()
        .title("This is a title");
htmlBuilder.body()
        .add(p("Hello”))
        .add(p("world!"));
String html = htmlBuilder.build();
```

DSL Sample .kt

```
val html = html {
    head {
        title("This is a title")
    }
    body {
        p("Hello")
        p("world!")
    }
}
```

Kotlin DSL

Kotlin DSL building blocks

- Extension function
  String.capitalize()
- Operator overloading
  mutableList += “item”
- Infix function call
  1 to “one”
- Lambda outside of parentheses
  reader.use { it.readLine() }
- Lambda with receiver
  StringBuilder.() -> Unit

Lambda with receiver

```
fun buildString(body: (StringBuilder) -> Unit): String {
    val builder = StringBuilder()
    body(builder)
    return builder.toString()
}
```

Lambda with receiver

```
buildString {
    it.append("Hello")
    it.append(' ')
    it.append("world")
}

fun buildString(body: (StringBuilder) -> Unit): String {
    val builder = StringBuilder()
    body(builder)
    return builder.toString()
}
```

Lambda with receiver

```
buildString { it: StringBuilder ->
    it.append("Hello")
    it.append(' ')
    it.append("world")
}

fun buildString(body: (StringBuilder) -> Unit): String {
    val builder = StringBuilder()
    body(builder)
    return builder.toString()
}
```

Lambda with receiver

```
buildString {
    it.append("Hello")
    it.append(' ')
    it.append("world")
}
```

Lambda with receiver

```
fun buildString(body: (StringBuilder) -> Unit): String {
    val builder = StringBuilder()
    body(builder)
    return builder.toString()
}
```

Lambda with receiver

```
fun buildString(body: StringBuilder.() -> Unit): String {
    val builder = StringBuilder()
    builder.body()
    return builder.toString()
}
```

Lambda with receiver

```
fun buildString(body: StringBuilder.() -> Unit): String {
    val builder = StringBuilder()
    builder.body()
    return builder.toString()
}
```

Lambda with receiver

```
buildString {
    append("Hello")
    append(' ')
    append("world")
}
```

Lambda with receiver

```
buildString {this: StringBuilder
    append("Hello")
    append(' ')
    append("world")
}
```

with from std library

```
inline fun <T, R> with(receiver: T, block: T.() -> R): R {
    return receiver.block()
}

with(StringBuilder()) {
    append("Hello")
    append(' ')
    append("world")
}
```

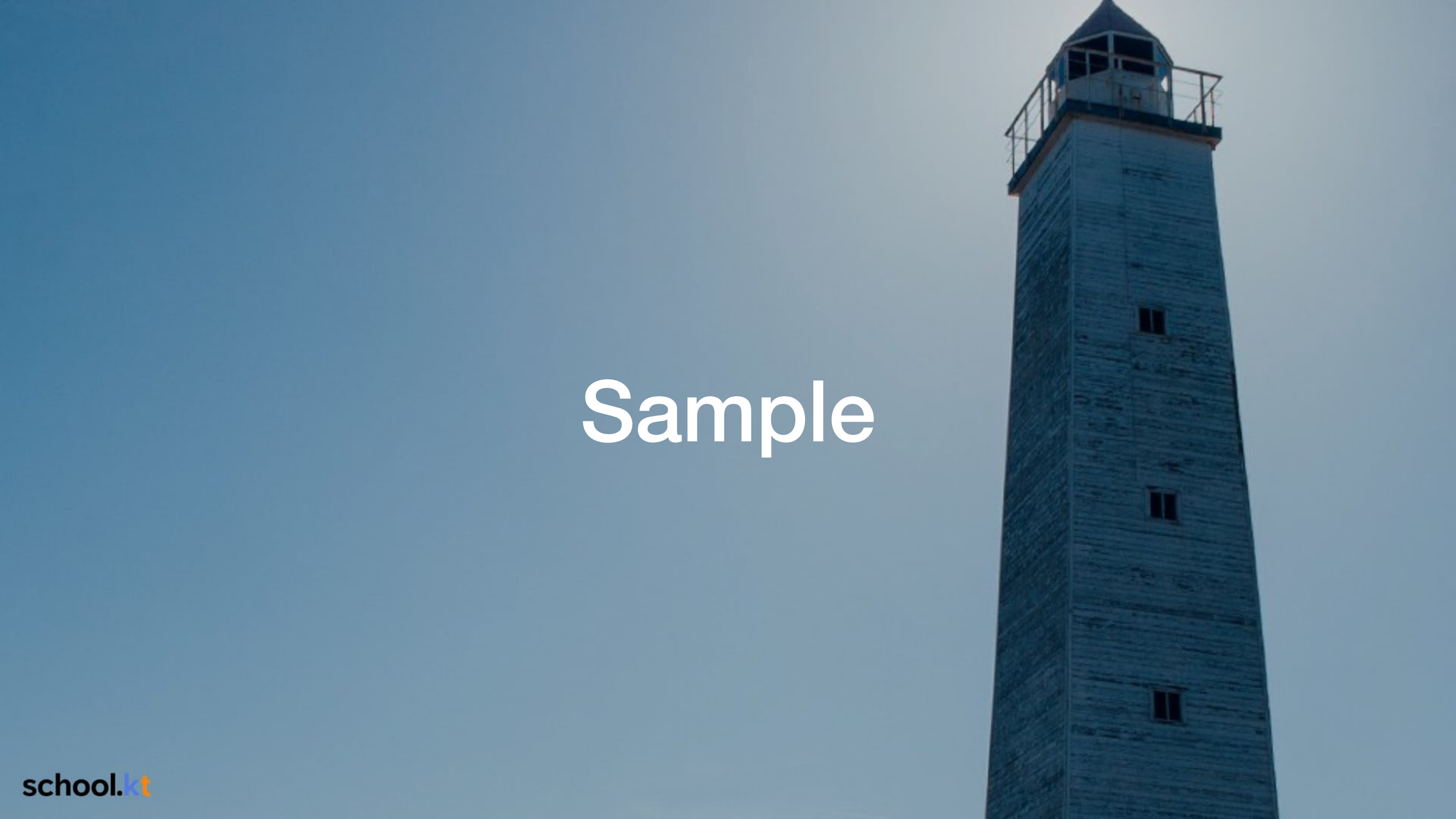
with from std library

```
with(StringBuilder()) {
    append("Hello")
    append(' ')
    append("world")
}
```

with from std library

```
with(StringBuilder()) {
    append("Hello")
    append(' ')
    append("world")
}
```

```
new StringBuilder()
    .append("Hello")
    .append(' ')
    .append("world");
```



Android StrictMode Sample

```
StrictMode.ThreadPolicy threadPolicy =
    new StrictModeCompat.ThreadPolicy.Builder()
        .detectResourceMismatches()
        .detectUnbufferedIo()
        .penaltyLog()
        .build();

StrictMode.VmPolicy vmPolicy = new StrictModeCompat.VmPolicy.Builder()
        .detectFileUriExposure()
        .detectLeakedRegistrationObjects()
        .detectContentUriWithoutPermission()
        .penaltyLog()
        .penaltyFlashScreen()
        .build();

StrictModeCompat.setPolicies(threadPolicy, vmPolicy);
```

Android StrictMode Sample

```
initStrictMode {
    threadPolicy {
        resourceMismatches = true
        unbufferedIo = true
        penalty {
            log = true
        }
    }

    vmPolicy {
        fileUriExposure = true
        leakedRegistrationObjects = true
        contentUriWithoutPermission = true
        penalty {
            log = true
            flashScreen = true
        }
    }
}
```

Android StrictMode Sample

```
fun initStrictMode(
        config: StrictModeConfig.() -> Unit
) {
    …
}
```

Android StrictMode Sample

```
fun initStrictMode(
        config: StrictModeConfig.() -> Unit
) {
    …
}
```

Android StrictMode Sample

```
fun initStrictMode(
        config: StrictModeConfig.() -> Unit
) {
    …
}
```

Android StrictMode Sample

```
class StrictModeConfig {

    var threadPolicyConfig = ThreadPolicyConfig()
        private set

    var vmPolicyConfig = VmPolicyConfig()
        private set

    fun threadPolicy(config: ThreadPolicyConfig.() -> Unit) {
        threadPolicyConfig.apply(config)
    }

    fun vmPolicy(config: VmPolicyConfig.() -> Unit) {
        vmPolicyConfig.apply(config)
    }
}
```

Android StrictMode Sample

```
class ThreadPolicyConfig {
    var customSlowCalls = false
    var diskReads = false
    var diskWrites = false
    var network = false
    var resourceMismatches = false
    var unbufferedIo = false

    var penaltyConfig = PenaltyConfig()
        private set

    fun penalty(config: PenaltyConfig.() -> Unit) {
        penaltyConfig.apply(config)
    }
}
```

Android StrictMode Sample

```
class PenaltyConfig {
    var death = false
    var deathOnNetwork = false
    var dialog = false
    var dropBox = false
    var flashScreen = false
    var log = false
}
```

Android StrictMode Sample

```
fun initStrictMode(
    config: StrictModeConfig.() -> Unit
) {
    StrictModeConfig().apply {
        config()

        val threadPolicy =
            buildThreadPolicy(threadPolicyConfig)
        StrictMode.setThreadPolicy(threadPolicy)

        val vmPolicy = buildVmPolicy(vmPolicyConfig)
        StrictMode.setVmPolicy(vmPolicy)
    }
}
```

Android StrictMode Sample

```
fun buildThreadPolicy(
    config: ThreadPolicyConfig
): ThreadPolicy {
    val builder = StrictModeCompat.ThreadPolicy.Builder()
    if (config.customSlowCalls) builder.detectCustomSlowCalls()
    if (config.diskReads) builder.detectDiskReads()
    …

    config.penaltyConfig.let {
        if (it.death) builder.penaltyDeath()
        if (it.deathOnNetwork) builder.penaltyDeathOnNetwork()
        …
    }
    return threadPolicyBuilder.build()
}
```

Android StrictMode Sample

```
initStrictMode {
    threadPolicy {
        resourceMismatches = true
        unbufferedIo = true
        penalty {
            log = true
        }
    }
}
```

Android StrictMode Sample

```
initStrictMode { this: StrictModeConfig
    threadPolicy {
        resourceMismatches = true
        unbufferedIo = true
        penalty {
            log = true
        }
    }
}
```

Android StrictMode Sample

```
initStrictMode { this: StrictModeConfig
    threadPolicy { this: ThreadPolicyConfig
        resourceMismatches = true
        unbufferedIo = true
        penalty {
            log = true
        }
    }
}
```

Android StrictMode Sample

```
initStrictMode { this: StrictModeConfig
    threadPolicy { this: ThreadPolicyConfig
        resourceMismatches = true
        unbufferedIo = true
        penalty { this: PenaltyConfig
            log = true
        }
    }
}
```

Android StrictMode Sample

```
initStrictMode { this: StrictModeConfig
    threadPolicy { this: ThreadPolicyConfig
        resourceMismatches = true
        unbufferedIo = true
        penalty { this: PenaltyConfig
            log = true
        }
    }
}
```

Android StrictMode Sample

```
initStrictMode { this: StrictModeConfig
    threadPolicy { this: ThreadPolicyConfig
        resourceMismatches = true
        unbufferedIo = true
        penalty { this: PenaltyConfig
            log = true
            threadPolicy {
                …
            }
        }
    }
}
```

Android StrictMode Sample

```
initStrictMode { this: StrictModeConfig
    threadPolicy { this: ThreadPolicyConfig
        resourceMismatches = true
        unbufferedIo = true
        penalty { this: PenaltyConfig
            log = true
            threadPolicy {
                …
            }
        }
    }
}
```

Android StrictMode Sample

```
@DslMarker
annotation class StrictModeDsl
```

Android StrictMode Sample

```
class StrictModeConfig() {

    var threadPolicyConfig = ThreadPolicyConfig()
        private set

    var vmPolicyConfig = VmPolicyConfig()
        private set

    fun threadPolicy(
        config: ThreadPolicyConfig.() -> Unit
    ) {
        threadPolicyConfig.apply(config)
    }

    fun vmPolicy(config: VmPolicyConfig.() -> Unit) {
        vmPolicyConfig.apply(config)
    }
}
```

Android StrictMode Sample

```
@StrictModeDsl
class StrictModeConfig() {

    var threadPolicyConfig = ThreadPolicyConfig()
        private set

    var vmPolicyConfig = VmPolicyConfig()
        private set

    fun threadPolicy(
        config: @StrictModeDsl ThreadPolicyConfig.() -> Unit
    ) {
        threadPolicyConfig.apply(config)
    }

    fun vmPolicy(config: @StrictModeDsl VmPolicyConfig.() -> Unit) {
        vmPolicyConfig.apply(config)
    }
}
```

Android StrictMode Sample

```
initStrictMode {
    threadPolicy {
        resourceMismatches = true
        unbufferedIo = true

        penalty {
            log = true
            threadPolicy {
                …
            }
        }
    }
}
```

StrictModeCompat

github.com/kirich1409/StrictModeCompat

Libraries with Kotlin DSL

Libraries with Kotlin DSL

- Gradle Kotlin DSL
  github.com/gradle/kotlin-dsl

Libraries with Kotlin DSL

- Gradle Kotlin DSL
  github.com/gradle/kotlin-dsl
- Kotlin Anko DSL
  github.com/Kotlin/anko

Android Layout XML

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/say_hello"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Anko UI

```
verticalLayout {
    val name = editText()
    button("Say Hello") {
        onClick { toast("Hello, ${name.text}!") }
    }
}
```

Libraries with Kotlin DSL

- Gradle Kotlin DSL
  github.com/gradle/kotlin-dsl
- Kotlin Anko DSL
  github.com/Kotlin/anko

Libraries with Kotlin DSL

- Gradle Kotlin DSL
  github.com/gradle/kotlin-dsl
- Kotlin Anko DSL
  github.com/Kotlin/anko
- kotlinx.html
  github.com/Kotlin/kotlinx.html

Libraries with Kotlin DSL

- Gradle Kotlin DSL
  github.com/gradle/kotlin-dsl
- Kotlin Anko DSL
  github.com/Kotlin/anko
- kotlinx.html
  github.com/Kotlin/kotlinx.html
- Spek
  spekframework.org

Libraries with Kotlin DSL

- Gradle Kotlin DSL
  github.com/gradle/kotlin-dsl
- Kotlin Anko DSL
  github.com/Kotlin/anko
- kotlinx.html
  github.com/Kotlin/kotlinx.html
- Spek
  spekframework.org
- Kakao
  github.com/agoda-com/Kakao

Kotlin DSL homework

Develop library for construct SQL queries with Kotlin DSL

```
SELECT id, name, surname FROM Students
    ORDER BY surname LIMIT 100


query {
    columns = ("id", "name", "surname")
    from = "Students"
    orderBy = "surname"
    limit = 100
}
```

Kotlin DSL homework

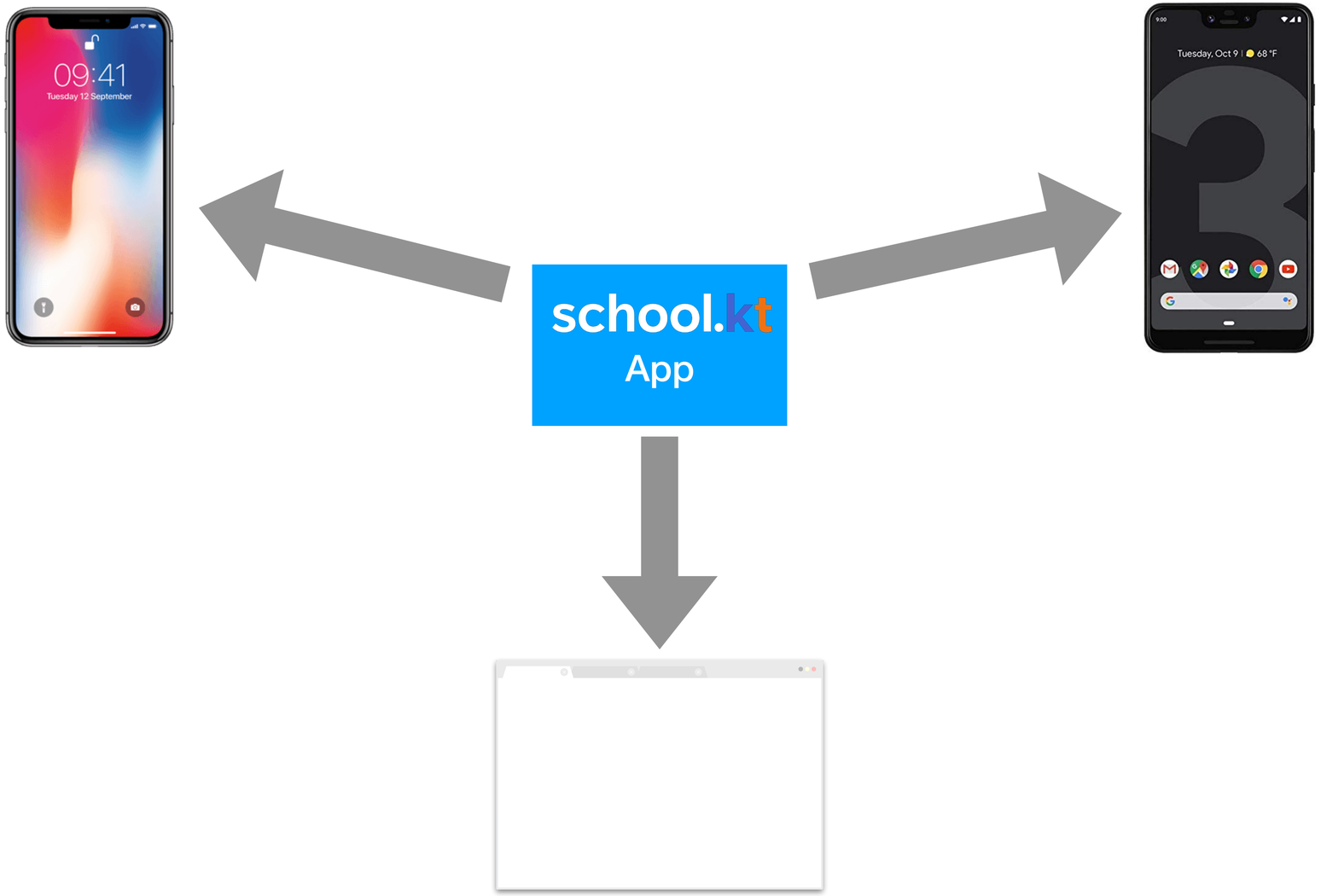
Develop library for construct Notifications base on AndroidX with DSL

```
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle(textTitle)
    .setContentText(textContent)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .build()


var builder = notification(CHANNEL_ID) {
    smallIcon = R.drawable.notification_icon_background
    content {
        title = textTitle
        text = textContent
    }
    priority = DEFAULT
}
```

Part 2. Multiplatform Projects

Problem



Existed solutions

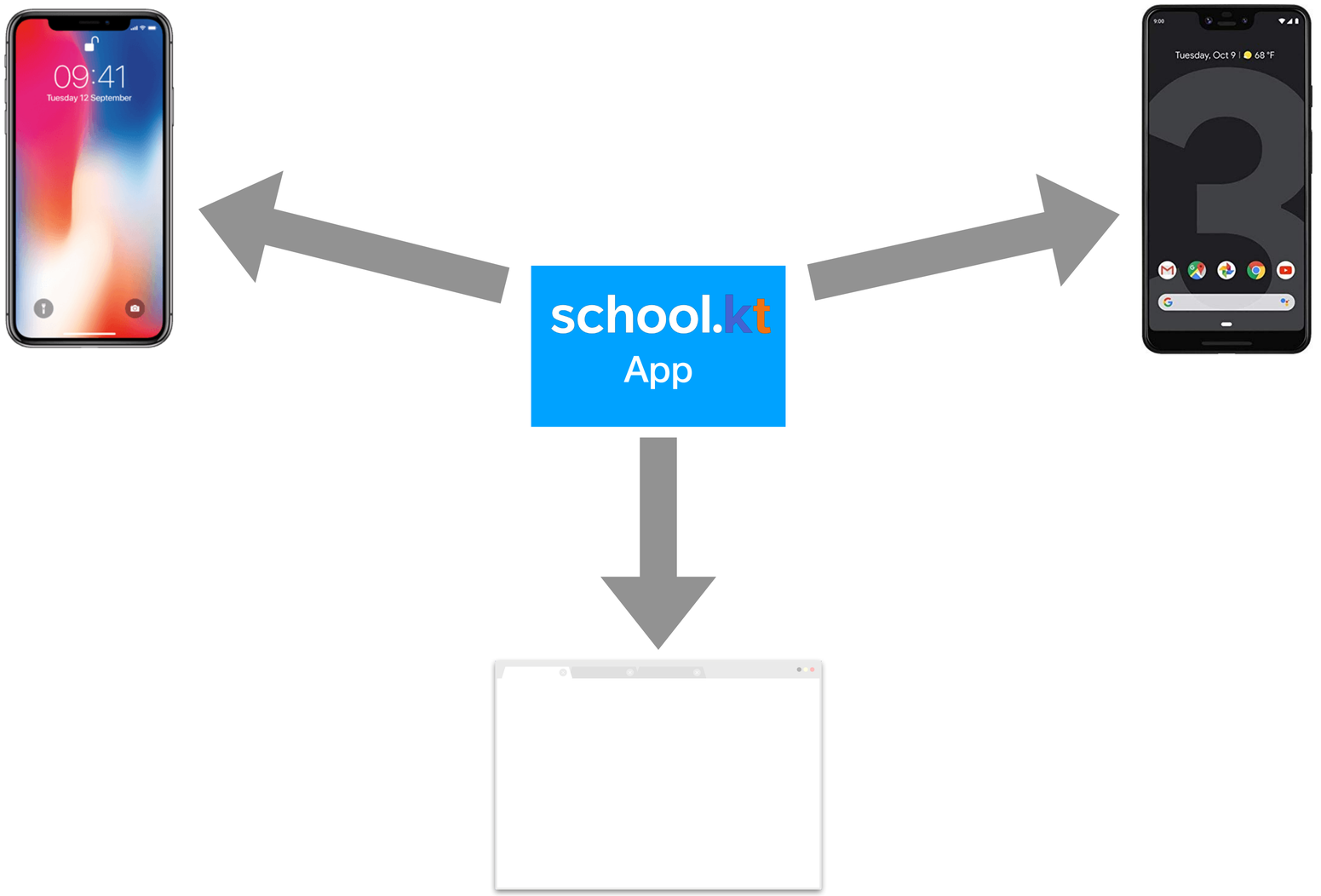
- React Native
- Apache Cordova
- Xamarin
- JS (common with V8 on Android)
- Flutter

Existed solutions summary

Existed solutions summary

- Not native language for the platform
- On some platform need VM to execute intermediate code
- Impacts on performance, memory, app size
- Common code can’t be optimized by target platform
+ Single language for different platforms

Kotlin Multiplatform Project (MPP)



Kotlin Multiplatform Project (MPP)

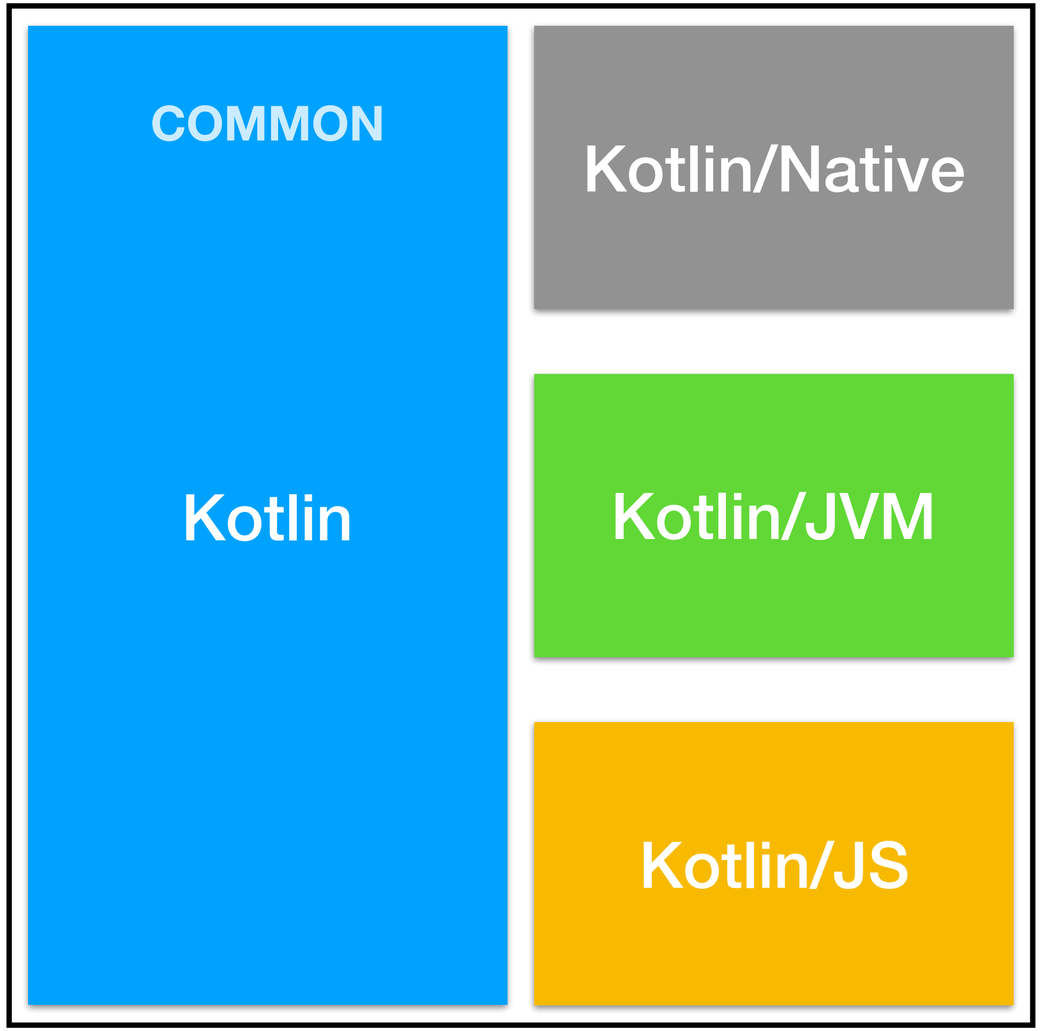
Kotlin/Native

Kotlin/JVM

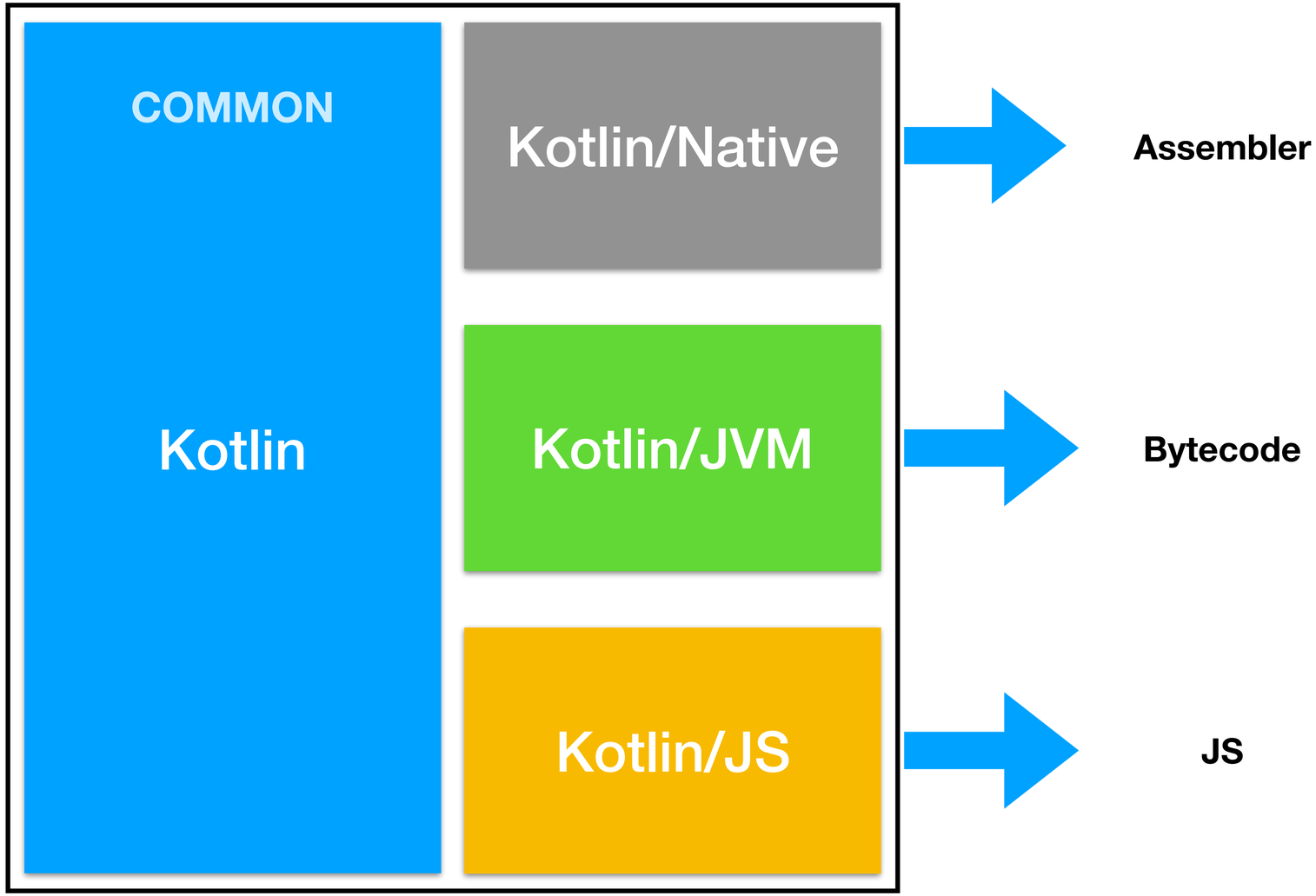
school.kt
App

Kotlin/JS, Kotlin/Native

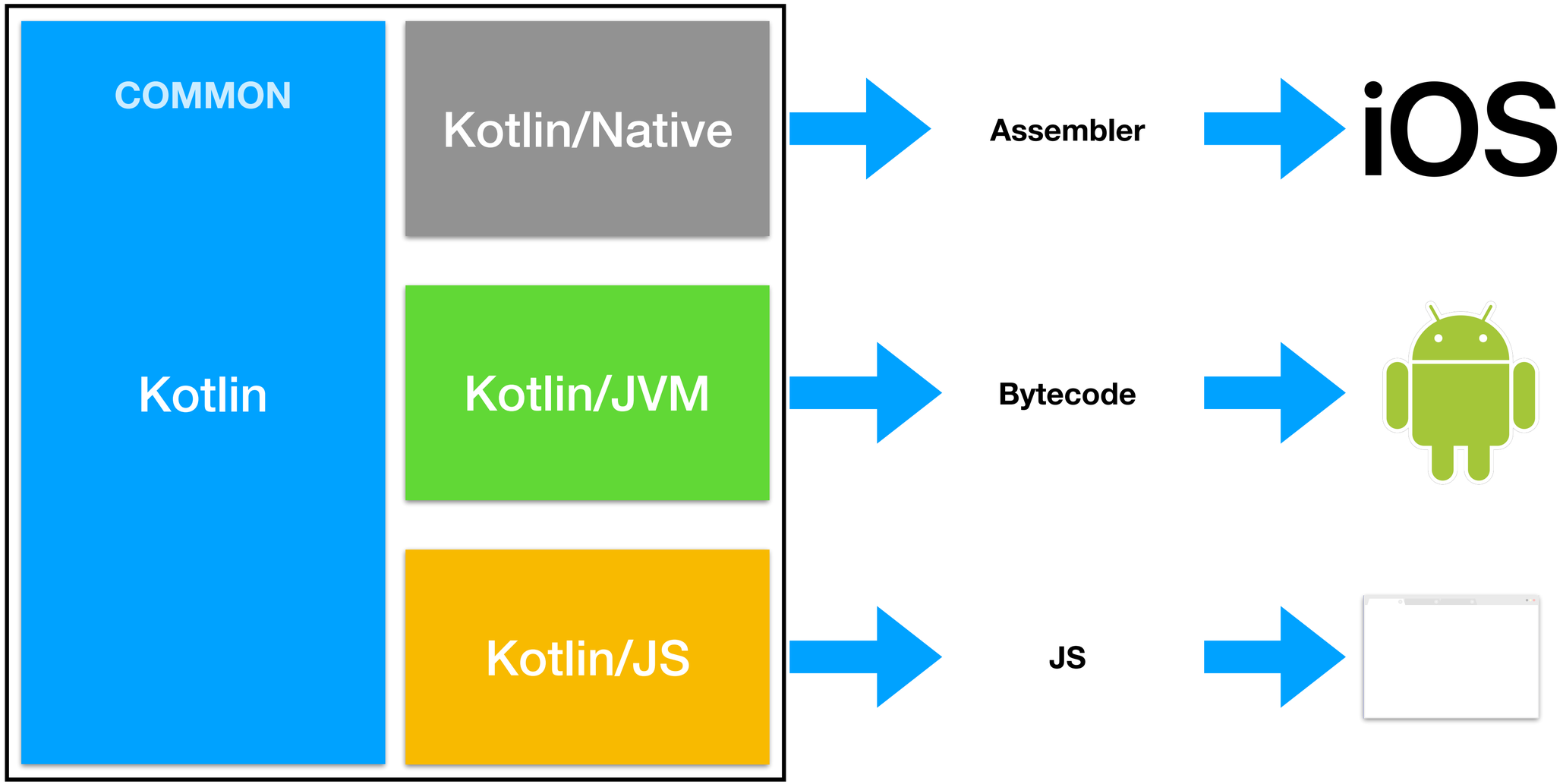
Kotlin Multiplatform Project (MPP)



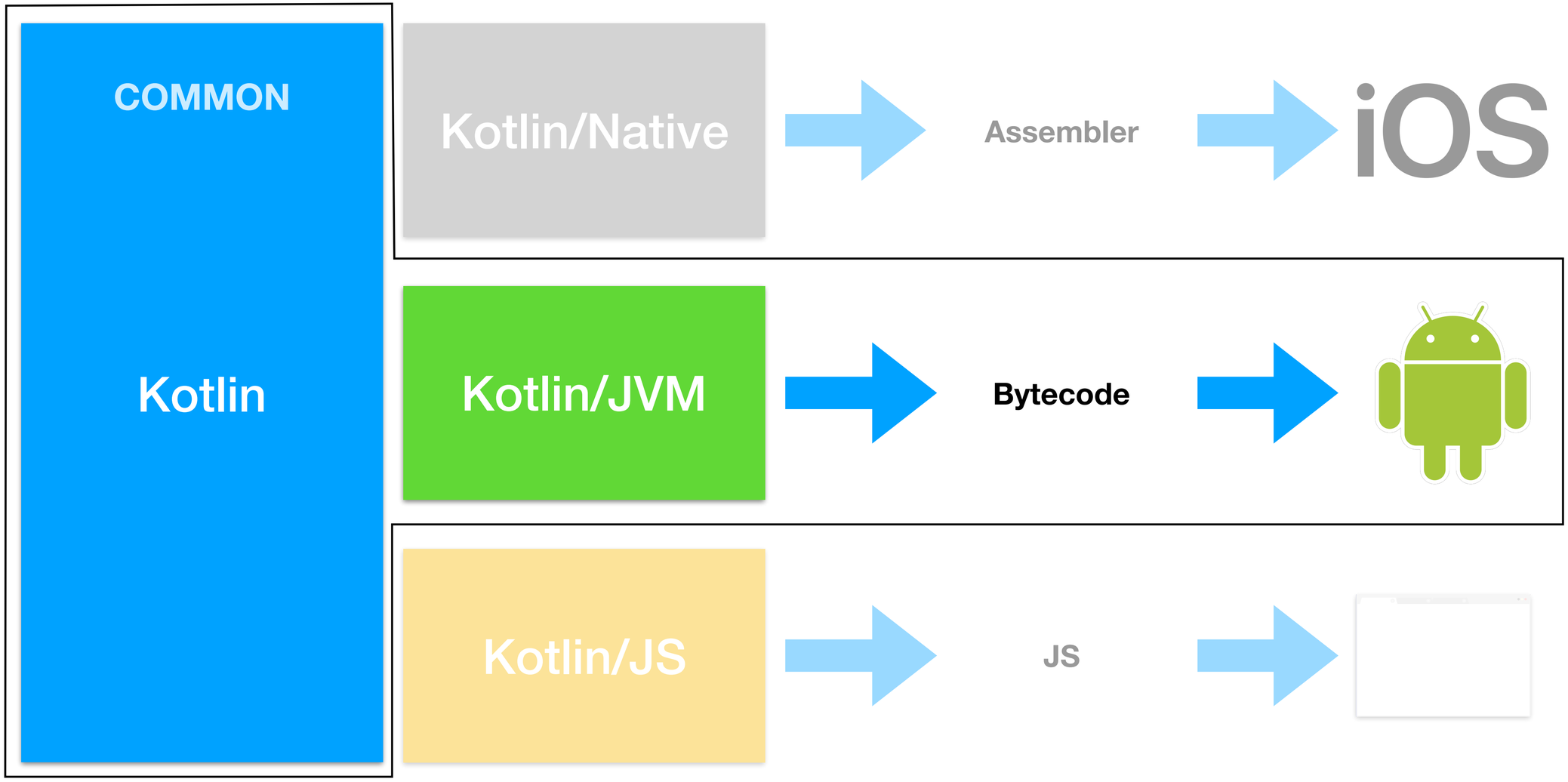
Kotlin Multiplatform Project (MPP)



Kotlin Multiplatform Project (MPP)



Kotlin Multiplatform Project (MPP)

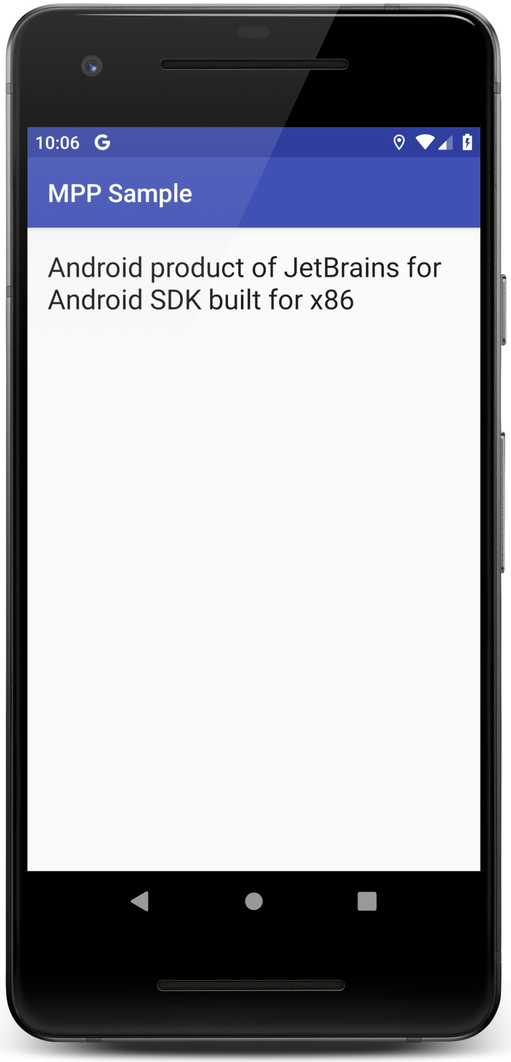


MPP

MPP

+ Has no any machine for execute platform abstract code
+ No huge performance impact, memory consuming or impact on app size
+ Code can be optimized by compiler
+ Single language for different platforms

Sample


10:06
MPP Sample
Android product of JetBrains for
Android SDK built for x86

Possible Targets

- JVM
- JS
- macOS
- iOS
- Android
- Native

Sample

```
greeting/
|-- src/
    |-- commonMain/
        |-- kotlin/ - Kotlin
    |-- androidLibMain/
        |-- kotlin/ - Kotlin/JVM
    |-- iosMain/
        |-- kotlin/ - Kotlin/Native
|-- build.gradle
```

build.gradle

```
kotlin {
    targets {
        fromPreset(presets.android, "androidLib"')
        fromPreset(presets.iosX64, "ios")
    }

    sourceSets {
        commonMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }

        androidMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib"
        }
    }
}
```

build.gradle

```
kotlin {
    targets {
        fromPreset(presets.android, "androidLib"')
        fromPreset(presets.iosX64, "ios")
    }

    sourceSets {
        commonMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }

        androidMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib"
        }
    }
}
```

build.gradle

```
kotlin {
    targets {
        fromPreset(presets.android, "androidLib"')
        fromPreset(presets.iosX64, "ios")
    }

    sourceSets {
        commonMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }

        androidMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib"
        }
    }
}
```

build.gradle

```
kotlin {
    targets {
        fromPreset(presets.android, "androidLib"')
        fromPreset(presets.iosX64, "ios")
    }

    sourceSets {
        commonMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }

        androidMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib"
        }
    }
}
```

build.gradle

```
kotlin {
    targets {
        fromPreset(presets.android, "androidLib"')
        fromPreset(presets.iosX64, "ios")
    }

    sourceSets {
        commonMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }

        androidMain {
            dependencies.implementation "org.jetbrains.kotlin:kotlin-stdlib"
        }
    }
}
```

Sample

```
class Product {
    val user: String
}

object Factory {
    fun create(config: Map<String, String>): Product

    val platform: String
}
```

common.kt

```
expect class Product {
    val user: String
}

expect object Factory {
    fun create(config: Map<String, String>): Product

    val platform: String
}
```

android.kt

```
class Product(val user: String) {

    override fun toString(): String {
        return "Android product of $user for ${Build.MODEL}"
    }
}

object Factory {
    fun create(config: Map<String, String>): Product {
        return Product(config.getValue("user"))
    }

    val platform = "android"
}
```

android.kt

```
actual class Product(actual val user: String) {

    override fun toString(): String {
        return "Android product of $user for ${Build.MODEL}"
    }
}

actual object Factory {
    actual fun create(config: Map<String, String>): Product {
        return Product(config.getValue("user"))
    }

    actual val platform = "android"
}
```

android.kt

```
actual class Product(actual val user: String) {

    override fun toString(): String {
        return "Android product of $user for ${Build.MODEL}"
    }
}

actual object Factory {
    actual fun create(config: Map<String, String>): Product {
        return Product(config.getValue("user"))
    }

    actual val platform = "android"
}
```

android.kt

```
actual class Product(actual val user: String) {

    override fun toString(): String {
        return "Android product of $user for ${Build.MODEL}"
    }
}

actual object Factory {
    actual fun create(config: Map<String, String>): Product {
        return Product(config.getValue("user"))
    }

    actual val platform = "android"
}
```

android.kt

```
actual class Product(actual val user: String) {

    override fun toString(): String {
        return "Android product of $user for ${Build.MODEL}"
    }
}

actual object Factory {
    actual fun create(config: Map<String, String>): Product {
        return Product(config.getValue("user"))
    }

    actual val platform = "android"
}
```

android.kt

```
actual class Product(actual val user: String) {
    fun androidSpecificOperation() {
        println("I am ${Build.MODEL} by ${Build.MANUFACTURER}")
    }

    override fun toString(): String {
        return "Android product of $user for ${Build.MODEL}"
    }
}

actual object Factory {
    actual fun create(config: Map<String, String>): Product {
        return Product(config.getValue("user"))
    }

    actual val platform = "android"
}
```

ios.kt

```
actual class Product(actual val user: String) {
    val model: String = memScoped {
        alloc<utsname>().apply { uname(ptr) }.machine.toKString()
    }

    fun iosSpecificOperation() = println("I am $model")

    override fun toString() = "iOS product of $user for $model"
}

actual object Factory {
    actual fun create(config: Map<String, String>): Product {
        return Product(config.getValue("user"))
    }

    actual val platform = "ios"
}
```

Why not interfaces?

- Can’t make constructor abstract
- Can’t guarantee existence of interface, class or object

Multiplatform libraries

- kotlinx.coroutines
  github.com/Kotlin/kotlinx.coroutines
- kotlinx.io
  github.com/Kotlin/kotlinx-io
- kotlinx.serialization
  github.com/Kotlin/kotlinx.serialization
- kotlinx.html
  github.com/Kotlin/kotlinx.html
- ktor
  ktor.io
- ktor-http-client
  ktor.io/clients/http-client


Thanks!!!