Kotlin CDMX User Group

Kotlin DSL: Moduralización en proyectos

Dinorah Tovar
Mobile Engineer
@ddinorahtovar

@ddinorahtovar
@dinorahto
@dinorahto
Doing code @ Konfio

What the heck is DSL?

A ***domain-specific language*** *(DSL) is a computer* ***language*** *specialized to a* ***particular*** *application* ***domain.*** *This is in contrast to a general-purpose* ***language*** *(GPL), which is broadly applicable across* ***domains.***

Wikipedia® Domain Specific Language

Like, for real? 🤔

- Provides you a flexible tool
- Particular applications
- Kotlin used it already

Has anyone used this before? 😮

YES

For real? 😮

```
Extension.function()
        +
    Lambda {
      //Code
    }
```

Extension Functions

```
//Extension function
fun Int.someCoolStuff {
    this.stuff()
}
```

Extension Functions

```
//Receiver
fun Int.someCoolStuff {
    this.stuff()
}
```

Extension Functions

```
//Lambda
{ () -> doStuff() }
```

Extension Functions

```
//Lambda with receiver
{ () -> this.doStuff() }
```

Lets create an DSL function

```
class IsleOfDogs {
    var type: String? = ""
}
```

Extension Functions

```kotlin
class IsleOfDogs {
    var type: String? = ""
}

fun isleOfDogs (lambda: IsleOfDogs.() -> Unit) : IsleOfDogs {
    return IsleOfDogs().apply(lambda)
}
```

A common example with Kotlin

```kotlin
fun buildString(action: (StringBuilder).() -> Unit): String {
    val stringBuilder = StringBuilder()
    action(stringBuilder)
    return stringBuilder.toString()
}
```

A common example with Kotlin

```
buildString {
    append("<")
    append("We love Kotlin at Konfio!")
    append(">")
}
```

A common example

```
textView.text = "We love Kotlin at Konfio"
textView.setOnClickListener {
        //This is a listener
}
textView.setTextColor(Color.BLACK)
```

Examples:

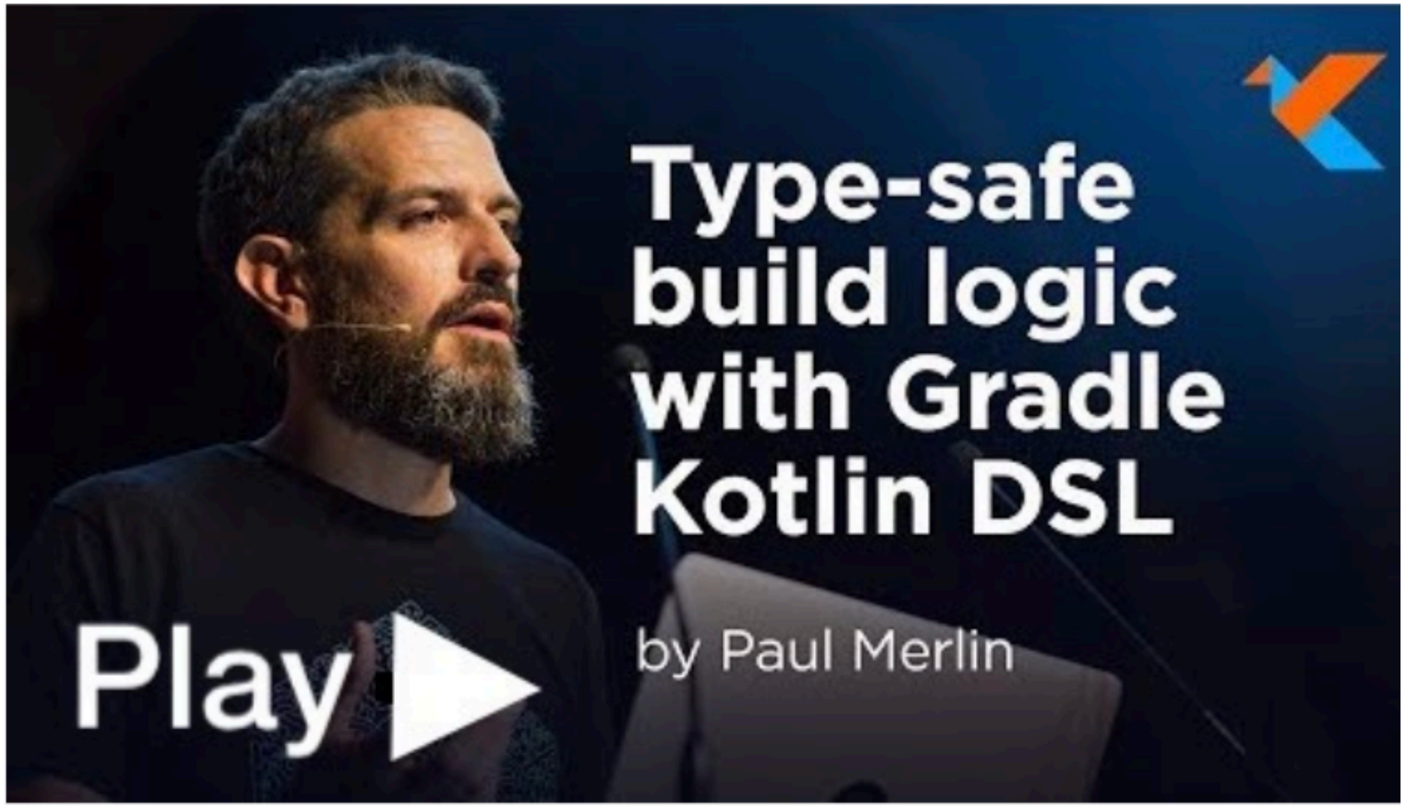
```
textView.apply {
    text = "Hola Konfio!"
    setOnClickListener {
        //This is a listener
    }
    textColor(Color.BLACK)
}
```

Type-safe Logic in Gradle

Type-safe model accessors

- Dependency and artifact configurations (such as `implementation` and `runtimeOnly` contributed by the Java Plugin)
- Project extensions and conventions (such as `sourceSets`)
- Elements in the `tasks` and `configurations` containers
- Elements in project-extension containers (for example the source sets contributed by the Java Plugin that are added to the `sourceSets` container)

This talk will not cover this, but here is something cool:



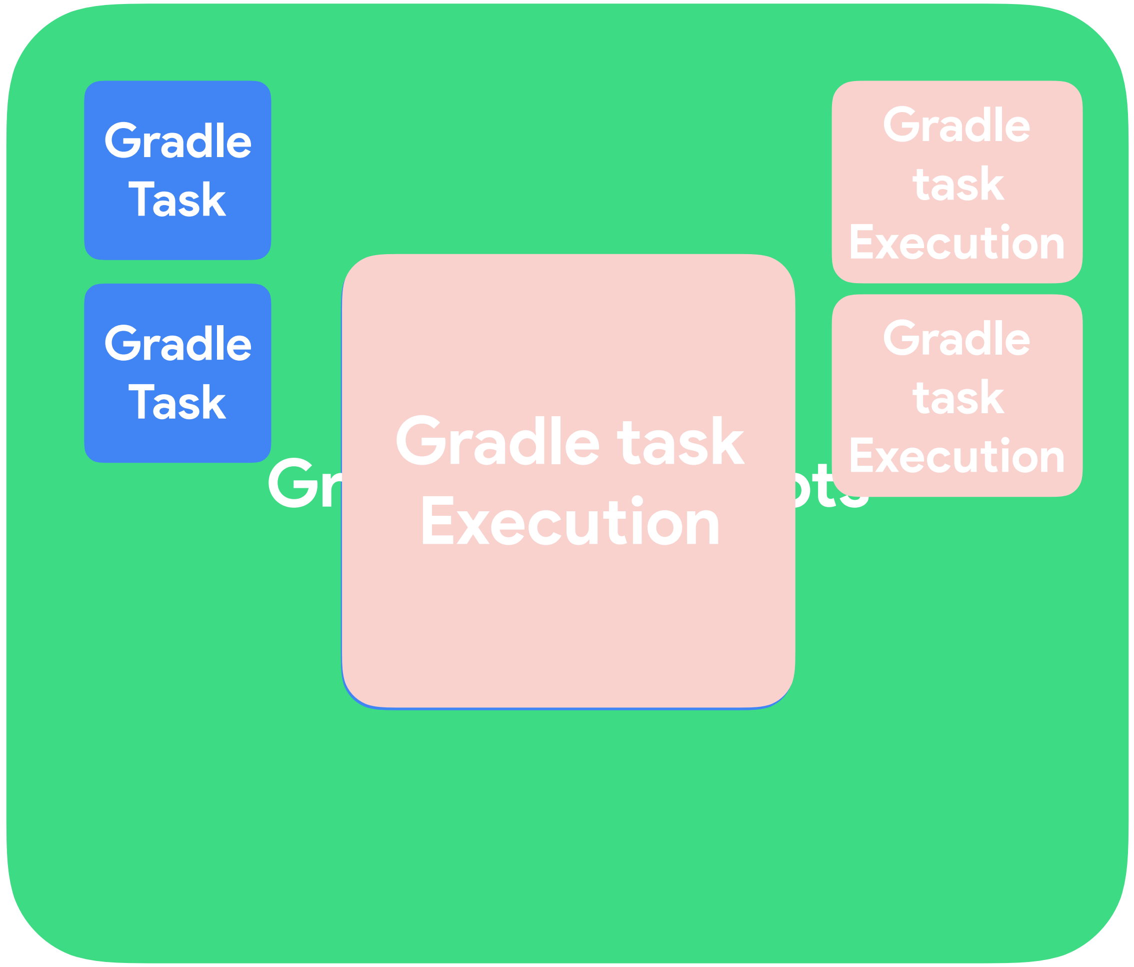
https://www.youtube.com/watch?v=mAtrEPeAJSc&feature=youtu.be

Gradle

Gradle

- Declarative elements describe the “what”
- The underlying logic creates the “how”
- Groovy provides an extensible DSL language

How Gradle Works?



Gradle and Kotlin

Gradle +
Kotlin DSL

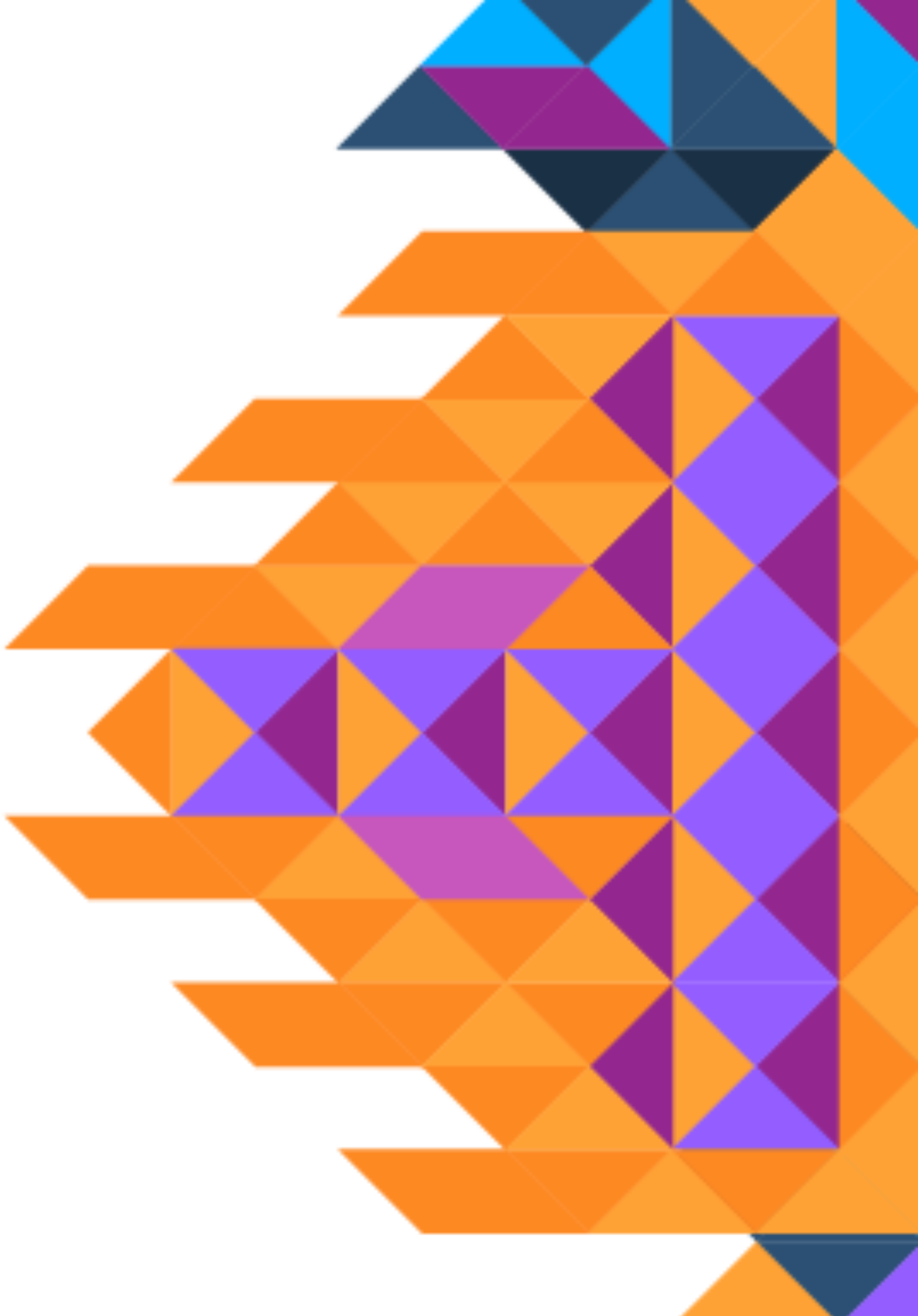
Gradle and Android

```
dependencies {
  implementation("com.squareup.okio:okio:2.0.0")
}
```

The real problem:

- Editing magic strings is error-prone.
- How do I centralize dependencies in a multi-modules project?
- Are there newer versions for my libs?

.kt vs
.kts vs
.gradle.kts.



Differences and similarities

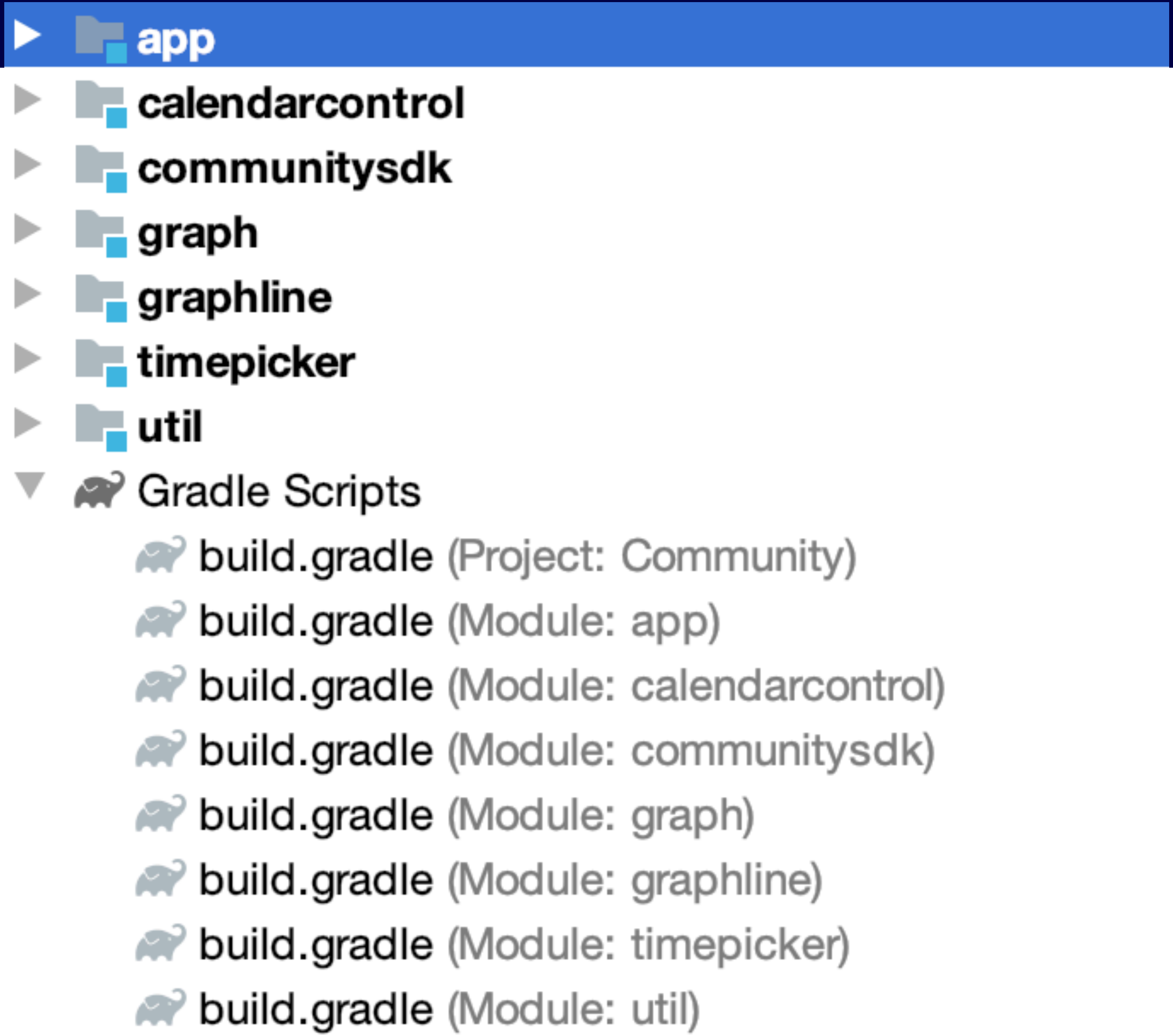
- They all contain Kotlin Code
- **.kt** files are compiled by the **Kotlin compiler**
- **.kts** files are executed by the **Kotlin scripting support**
- **.gradle.kts** are hosted by **Gradle**

.gradle.kts ❤️

- Kotlin friendly extension of the Gradle API
- **Delegated** properties for Gradle properties and collections
- **Dynamically** generates **Kotlin extensions**
- For models elements contributed by plugins, like task or configuration

The solution

The real problem:



app
calendarcontrol
communitysdk
graph
graphline
timepicker
util
Gradle Scripts
build.gradle (Project: Community)
build.gradle (Module: app)
build.gradle (Module: calendarcontrol)
build.gradle (Module: communitysdk)
build.gradle (Module: graph)
build.gradle (Module: graphline)
build.gradle (Module: timepicker)
build.gradle (Module: util)

Multimodule projects

- Manual Management
- Google’s Recommendation using “ext”
- Kotlin + buildSrc + DSL

Multimodule projects

```
//ModuleA - build.gradle

implementation "com.android.support:support-annotations:27.0.2"
implementation "com.android.support:appcompat-v7:27.0.2"
implementation "com.squareup.retrofit2:retrofit:2.3.0"
implementation "com.squareup.retrofit2:adapter-rxjava2:2.3.0"
```

Multimodule projects

```
//ModuleB - build.gradle

implementation "com.android.support:support-annotations:27.0.2"
implementation "com.android.support:appcompat-v7:27.0.2"
implementation "com.squareup.retrofit2:retrofit:2.3.0"
implementation "com.squareup.retrofit2:adapter-rxjava2:2.3.0"
```

Multimodule projects

```
ext {
  versions = [
    support_lib: "27.0.2",
    retrofit: "2.3.0",
 ]
  libs = [
    support_annotations: "com.android.support:support-annotations:${versions.support_lib}",
    support_appcompat: "com.android.support:appcompat-v7:${versions.support_lib}",
    retrofit :"com.squareup.retrofit2:retrofit:${versions.retrofit}"
  ]
}
```

Multimodule projects

```
//Module-A / build.gradle


implementation libs.support_annotations
implementation libs.support_appcompat_v7
implementation libs.retrofit
implementation libs.retrofit_rxjava_adapter
implementation libs.rxjava
```

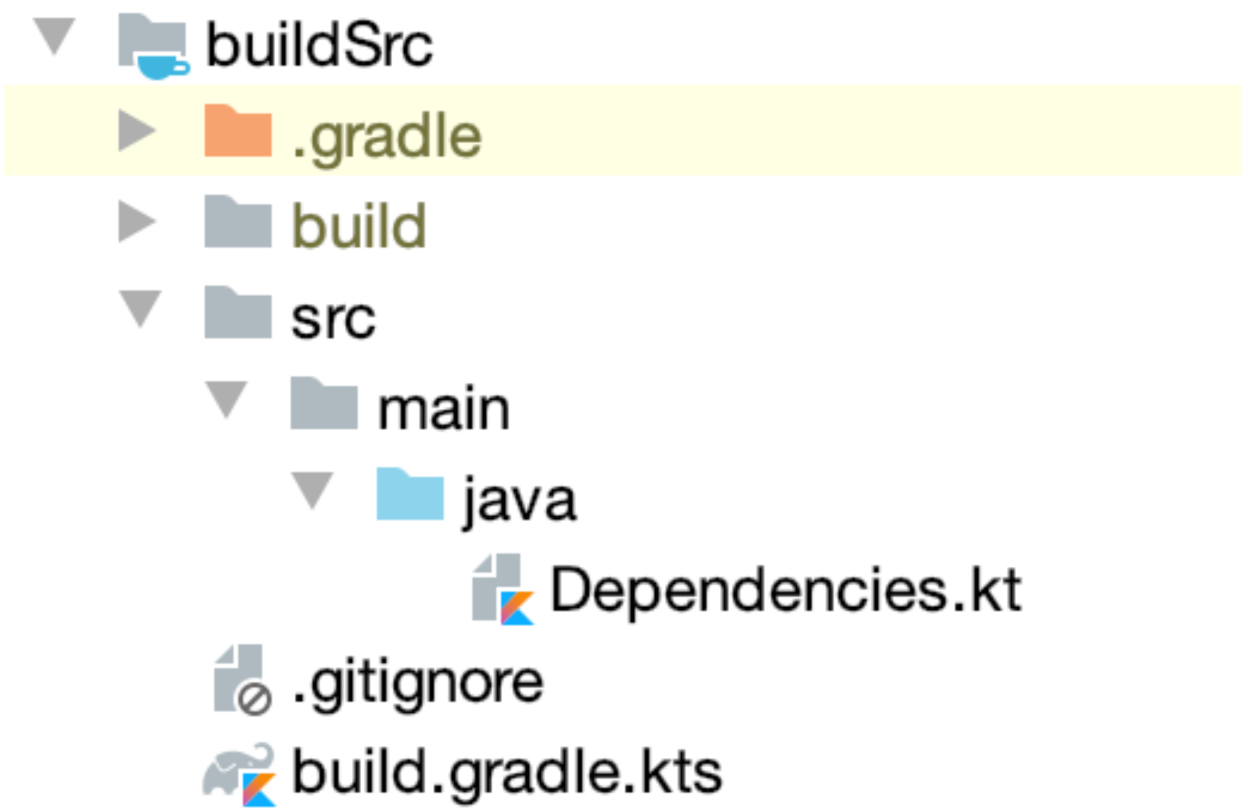
Multimodule projects

//Module-A / build.gradle

```
implementation libs.support_annotations
implementation libs.support_appcompat_v7
implementation libs.retrofit
implementation libs.retrofit_rxjava_adapter
implementation libs.rxjava
```

The solution

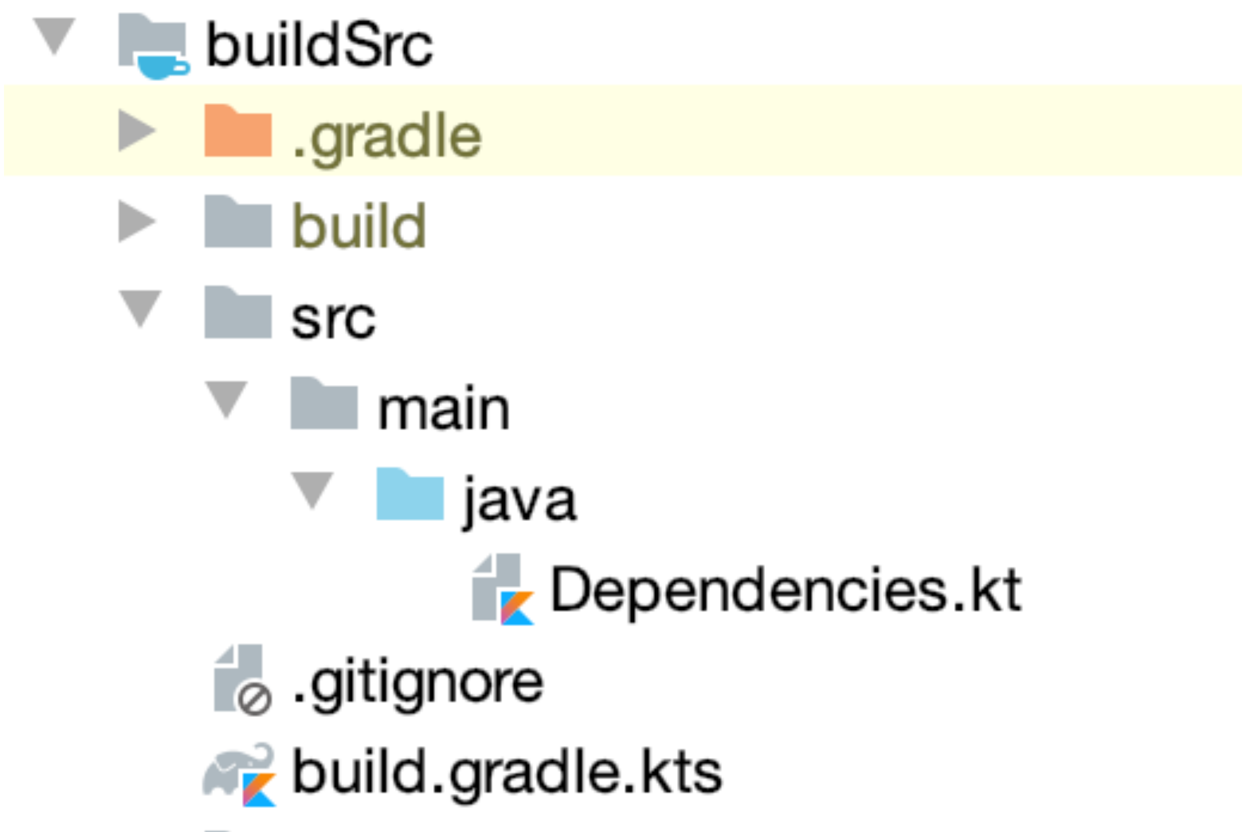
- You can create a **buildSrc** module with **Kotlin** code to manage dependencies and get IDE completion support.



```
▼ buildSrc
    ▶ .gradle
    ▶ build
    ▼ src
        ▼ main
            ▼ java
                Dependencies.kt
    .gitignore
    build.gradle.kts
```

The solution

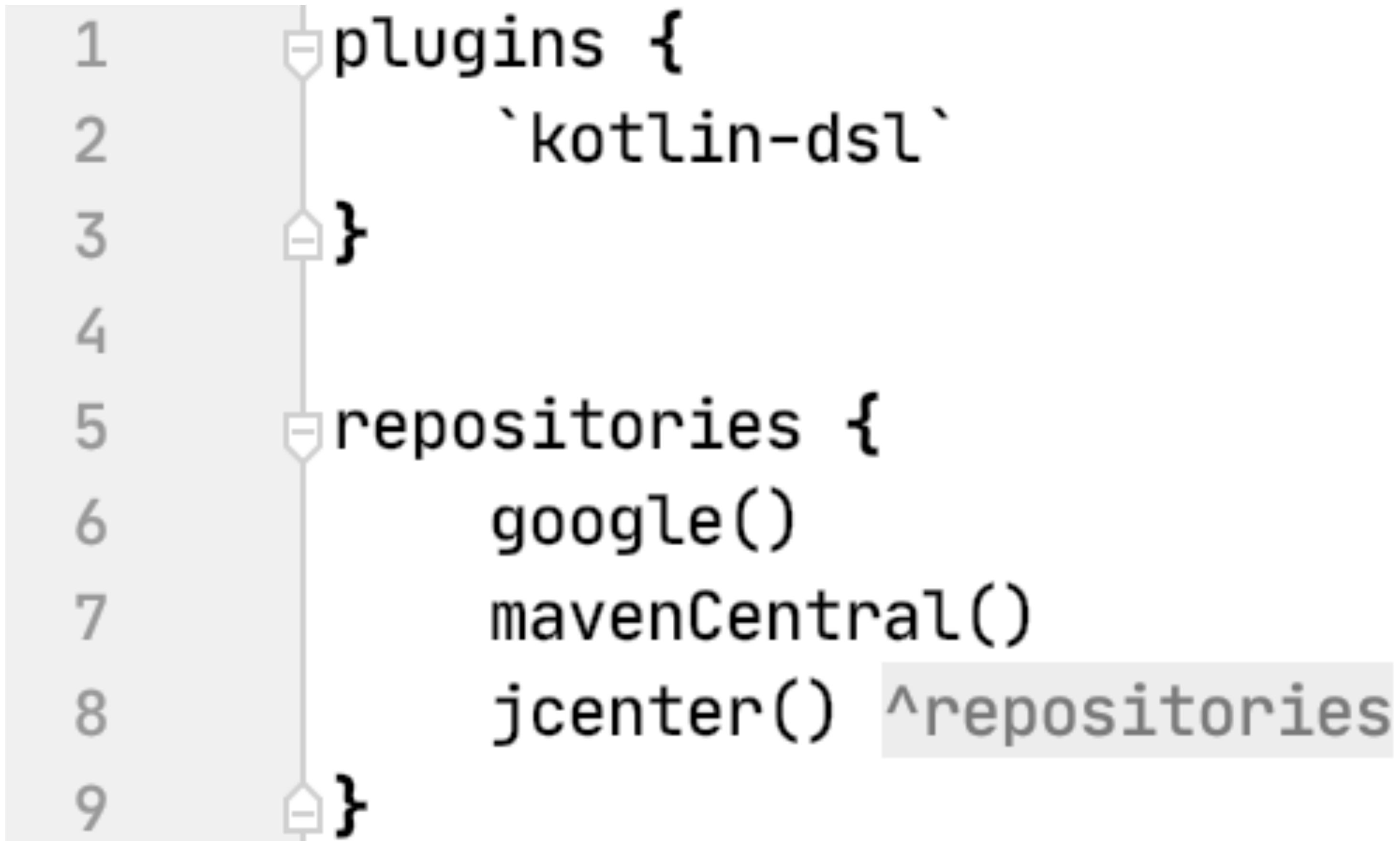
- You can create a **buildSrc** module with **Kotlin** code to manage dependencies and get IDE completion support.



```
▼ buildSrc
  ▶ .gradle
  ▶ build
  ▼ src
    ▼ main
      ▼ java
          Dependencies.kt
  .gitignore
  build.gradle.kts
```

The solution

- Inside **build.gradle.kts**

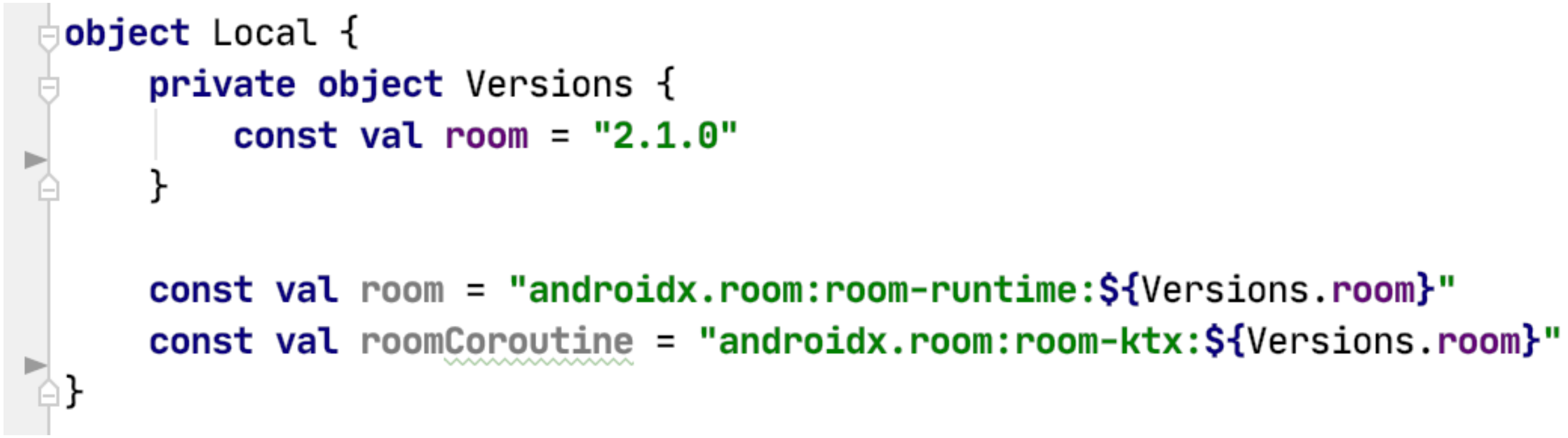


```
plugins {
	`kotlin-dsl`
}

repositories {
	google()
	mavenCentral()
	jcenter() ^repositories
}
```

The solution

- Inside **Dependencies.kt**

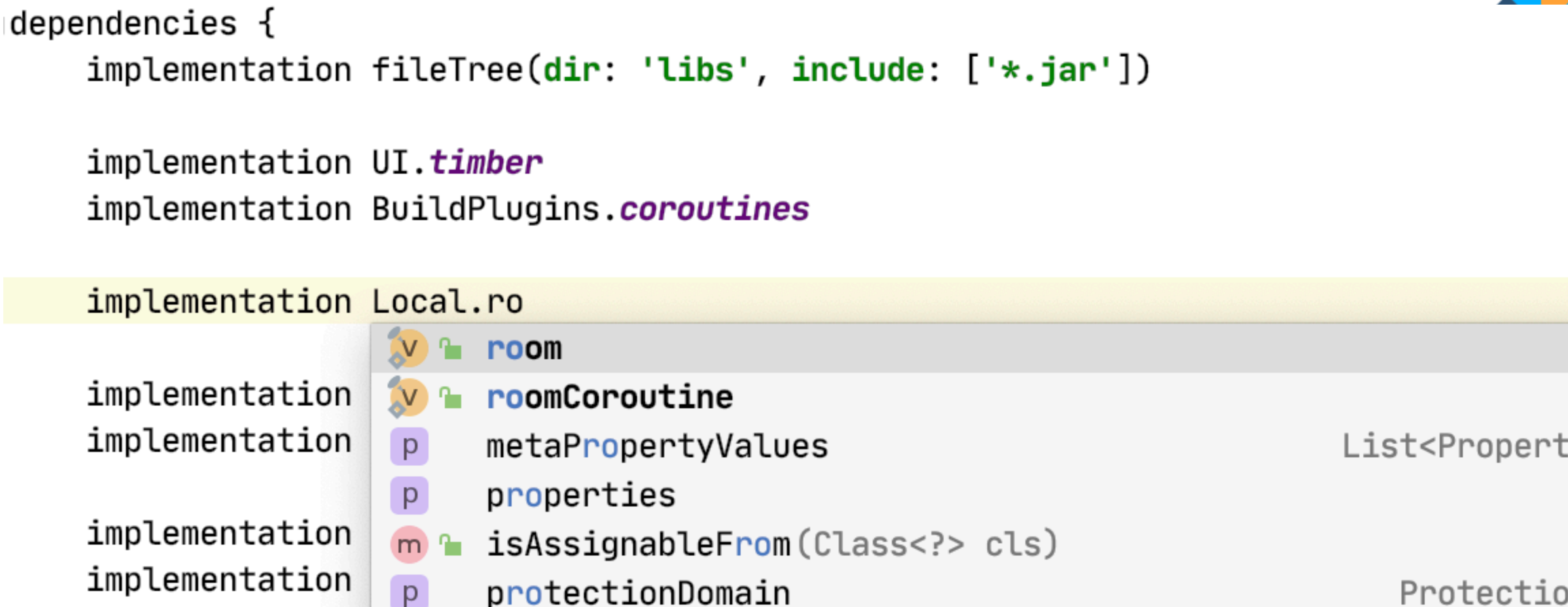


```
object Local {
    private object Versions {
        const val room = "2.1.0"
    }

    const val room = "androidx.room:room-runtime:${Versions.room}"
    const val roomCoroutine = "androidx.room:room-ktx:${Versions.room}"
}
```

The solution

- Inside your App Gradle



```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation UI.timber
    implementation BuildPlugins.coroutines

    implementation Local.ro
                       room
                       roomCoroutine
    implementation     metaPropertyValues                    List<Propert
    implementation     properties
                       isAssignableFrom(Class<?> cls)
    implementation
    implementation     protectionDomain                       Protectio
```

Kotlin CDMX User Group

Kotlin DSL: Moduralización en proyectos

Dinorah Tovar
Mobile Engineer
@ddinorahtovar

@ddinorahtovar
@dinorahto
@dinorahto
Doing code @ Konfio