Conventional development

In Gradle

https://github.com/erikeelde/shorty

What if we could turn this

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
}

android {
    namespace = "se.eelde.android.shorty"
    compileSdk {
        version = release(36)
    }

    defaultConfig {
        applicationId = "se.eelde.android.shorty"
        minSdk = 24
        targetSdk = 36
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner =
            "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile(
                    "Proguard-android-optimize.txt"
                ),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }
    kotlinOptions {
        jvmTarget = "11"
    }
    buildFeatures {
        compose = true
    }
}

dependencies {
    // Our dependencies
}
```

Into this

```
plugins {
    id("shorty.application")
    id("shorty.application.compose")
}

android {
    namespace = "se.eelde.android.shorty"

    buildTypes {
        release {
            proguardFiles(
                getDefaultProguardFile(
                    "Proguard-android-optimize.txt"
                ),
                "proguard-rules.pro"
            )
        }
    }
}

dependencies {
    // Our dependencies
}
```

And our feature modules looks like this

```
plugins {
    id("shorty.module")
}

android {
    namespace = "se.eelde.android.feature"
}

dependencies {
    // Our dependencies
}
```

Agenda

- What are Gradle conventions
- Prerequisites
- Implementing our convention
- Demo

What is a convention

- A widely accepted practice or rule

Android conventions

- Application conventions vs Library conventions
- Java compiler version
- minSdk version
- compileSdk version
- Test configuration
- Gradle managed devices configuration

Conventions in Gradle

- Allows us to formalize our widely accepted practices and rules into a set of Gradle plugins

Benefits of gradle Convention Plugins

- Unify module configuration
  - We can override convention values
- Great for simplifying module setup
- Great when updating your module configuration

Agenda

- What are Gradle conventions
- Prerequisites
- Implementing our convention
- Demo

Showcase application

- An android application
- “Shorty” - https://github.com/erikeelde/shorty
- Moderately complex
  - 1 App module
  - 1 Feature module

Where do we place our convention

- BuildSrc
- Composite build

buildSrc

- “Magic” directory in project root, /buildSrc
- Project that is built before all other projects
- Available on other projects classpaths
- Gradle < 8, built first using gradle daemon java compiler
- Gradle > 8, similar to composite build
- Recommendation is to use a composite build

Composite Build

- A build that includes other builds
- “includeBuild()” in settings.gradle.kts
- Works for both regular dependencies and plugins
- Allow you to connect independent builds together
- You can replace a maven coordinate with sources (github repo)

Simple composite build

```
includeBuild("../../open_source_repositories/ktor")
```

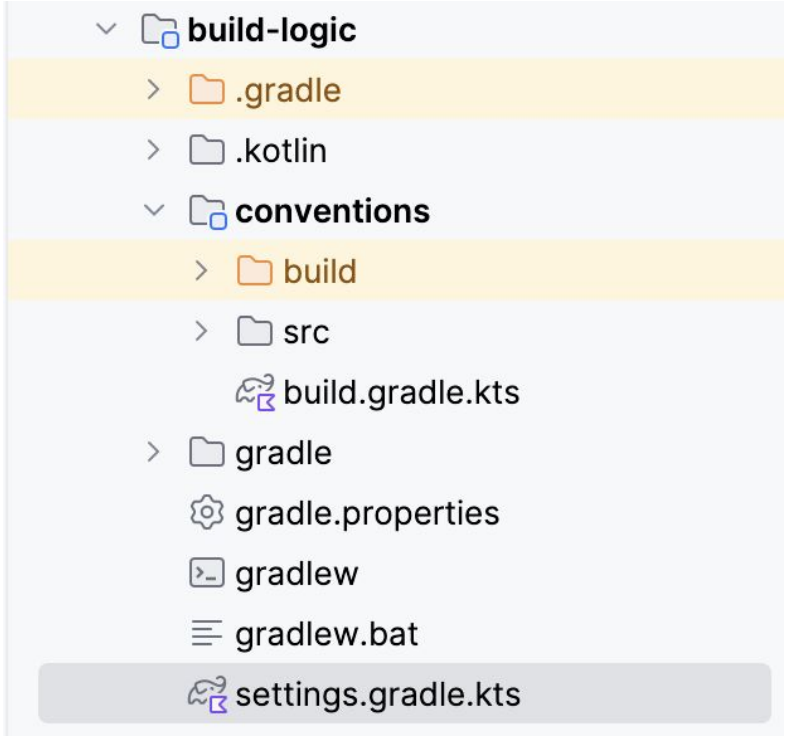
Replacing dependencies with sources

```
includeBuild("../../open_source_repositories/ktor") {
    dependencySubstitution {
        substitute(module("io.ktor:ktor-http"))
            .using(project(":ktor-http"))
    }
}
```

We’re targeting the “:ktor-http” subproject inside of the ktor folder because that’s how ktor organizes their projects in their repo.

Composite build for our convention

```
includeBuild("build-logic")
```



Agenda

- What are Gradle conventions
- Prerequisites
- Implementing our convention
- Demo

Two types of conventions

- Dsl convention
- Programmatic convention

Dsl convention

- The way Gradle intended it?
- Official Gradle documentation
- Sharing build logic between subprojects Sample

Example in shorty:

- https://github.com/erikeelde/shorty/pull/3

Programmatic convention

- Sometimes Dsl is not enough - CommonExtension<*,*,*,*,*,*>
- Example from now in android repository

Example in shorty:

- https://github.com/erikeelde/shorty/pull/4

Tips

- Treat your conventions as separate project. Don’t share anything between projects
- Every change in conventions will require reconfiguration of every module in your project (that uses the convention)
- Be cautious about adding dependencies in conventions. It is hard to generalize what is “required minimal configuration”
- Consider your end user and their expectations of your apis
- When your convention breaks - your build breaks. Work in small increments

Agenda

- What are Gradle conventions
- Prerequisites
- Implementing our convention
- Demo

Demo

- For debug builds of our application we want to target the most recent version of android - “targetSdk = 36”

Demo

- Implement your build logic in a regular build.gradle.kts file
- Control click into the implementation to find the gradle Extension
- In the convention use extensions.configure<Extension> { /*…*/ }
- You can often copy paste your dsl from the build.gradle (not always)

Questions?

Shorty:

https://github.com/erikeelde/shorty

Gradle conventions:

https://docs.gradle.org/current/userguide/implementing_gradle_plugins_convention.html

Composite build:

https://docs.gradle.org/current/userguide/composite_builds.html

Sample DSL convention plugin:

https://docs.gradle.org/current/samples/sample_convention_plugins.html

Now in android:

https://github.com/android/nowinandroid/tree/main/build-logic