Topic for discussion:

Groovy DSL Script + Kotlin DSL Script

Workshop hosted by Rohan(Android Engineer)

What is Groovy DSL Script?

As part of our Android Studio project structure, Groovy DSL (Domain Specific Language) Script is used by default. Using the Gradle files, it builds the apk based on dependencies, plugins, project settings, etc.

What is Kotlin DSL Script?

Kotlin DSL (Domain Specific Language) is a Gradle build language that uses Kotlin files for development and provides the benefits of Kotlin. It is fully supported in Android Studio IDE and provides enhanced editing features such as IntelliSense, code assist, and many more.

What are the benefits of switching from Groovy to Kotlin Gradle Script in Android Studio?

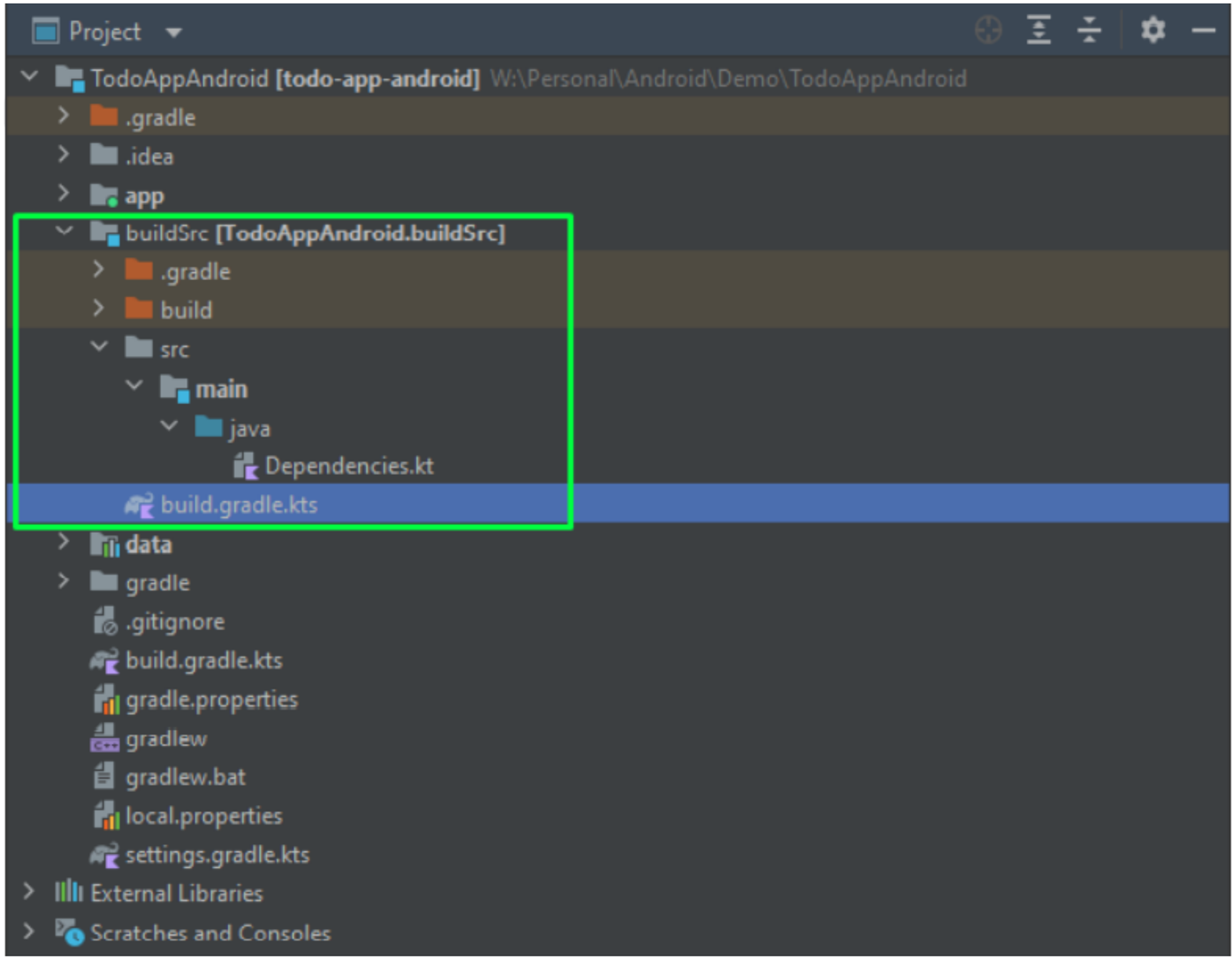
1. Future of Android Studio
2. Central dependency management
3. IDE support & better compile time checking
4. Use of Kotlin variables and functions
5. Compatible with Android Studio Editor

Basic steps to migrate your project to Kotlin Gradle Script:

Step1:
Create the buildSrc directory in your rootDirectory of the project as shown in the image:

- Add *build.gradle.kts* file and apply the kotlin-dsl plugin configuration as shown in next slide.
- Add Dependencies.kt file.

This buildSrc directory is a Gradle feature that allows us to prepare a Central Dependency Management System for our project.



Step2:

Make your build.gradle.kts files as shown in the image in buildSrc directory

```
import org.gradle.kotlin.dsl.`kotlin-dsl`
plugins {
    `kotlin-dsl`
}
repositories {
    gradlePluginPortal()
    mavenCentral()
}
```

This is plugin which enables to use Kotlin DSL features in gradle files.

Make Dependencies.kt file as shown in the image.

It includes the versions of app-related configs and dependencies that will be used in app-level build.gradle.kts file.

```
import org.gradle.api.artifacts.dsl.DependencyHandler

object AppConfig {
    const val compileSdk = 32
    const val minSdk = 21
    const val targetSdk = 32
    const val versionCode = 1
    const val versionName = "1.0"
    const val testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
}

object Versions {
    const val junitVersion = "4.13.2"
    const val extJunitVersion = "1.1.3"
    const val espressoVersion = "3.4.0"

    const val xCoreKtxVersion = "1.7.0"
    const val lifecycleRuntimeKtxVersion = "2.4.1"
    const val composeVersion = "1.1.1"
    const val composeCompilerVersion = "1.2.0-beta01"
    const val activityComposeVersion = "1.4.0"
```

```
    const val retrofitVersion = "2.6.2"
    const val gsonVersion = "2.8.5"
    const val gsonConverterVersion = "2.1.0"
    const val loggingVersion = "3.14.0"
    const val coroutineVersion = "1.6.1"
    const val navComposeVersion = "2.4.2"
}

object Dependencies {
    //Core
    private const val xCoreKtx =
"androidx.core:core-ktx:${Versions.xCoreKtxVersion}"
    private const val lifeCycleRuntime =
"androidx.lifecycle:lifecycle-runtime-ktx:${Versions.lifecycleRuntimeKtxVersion}"

    //Compose
    private const val uiCompose =
"androidx.compose.ui:ui:${Versions.composeVersion}"
    private const val materialCompose =
"androidx.compose.material:material:${Versions.composeVersion}"
    private const val uiComposeTooling =
"androidx.compose.ui:ui-tooling:${Versions.composeVersion}"
    private const val uiComposeToolingPreview =
"androidx.compose.ui:ui-tooling-preview:${Versions.composeVersion}"
    private const val activityCompose =
"androidx.activity:activity-compose:${Versions.activityComposeVersion}"
    private const val navigationCompose =
"androidx.navigation:navigation-compose:${Versions.navComposeVersion}"
    private const val junit = "junit:junit:${Versions.junitVersion}"
    private const val extJunit =
"androidx.test.ext:junit:${Versions.extJunitVersion}"
    private const val espressoCore =
"androidx.test.espresso:espresso-core:${Versions.espressoVersion}"
    private const val junit4Compose =
```

Variables declared in Dependencies{...} section are meant for the libraries used in the Android Project.

All the variables will be used in app-level build.gradle.kts files

```
"androidx.compose.ui:ui-test-junit4:${Versions.composeVersion}"

    //Retrofit
    const val retrofit =
"com.squareup.retrofit2:retrofit:${Versions.retrofitVersion}"
    const val loggingInterceptor =
"com.squareup.okhttp3:logging-interceptor:${Versions.loggingVersion}"

    //Coroutines
    const val coroutinesCore =
"org.jetbrains.kotlinx:kotlinx-coroutines-core:${Versions.coroutineVersion}"
    const val coroutinesAndroid =
"org.jetbrains.kotlinx:kotlinx-coroutines-android:${Versions.coroutineVersion}"

    //Gson
    const val gson = "com.google.code.gson:gson:${Versions.gsonVersion}"
    const val gsonConverter =
"com.squareup.retrofit2:converter-gson:${Versions.gsonConverterVersion}"

    //Only meant for implementation("...")
    val appLibs = arrayListOf<String>().apply {
        add(xCoreKtx)
        add(lifeCycleRuntime)
        add(uiCompose)
        add(materialCompose)
        add(uiComposeToolingPreview)
        add(activityCompose)
        add(navigationCompose)
    }

    //Only meant for debugImplementation("...")
    val debugAppLibs = arrayListOf<String>().apply {
        add(uiComposeTooling)
    }
```

We can also group the libraries into a list and declare it in a single variable.

This prevents us from declaring individual variables in app-level build.gradle.kts files.

This improves the readability of your grade files.

```
    val androidTestLibs = arrayListOf<String>().apply {
        add(extJunit)
        add(espressoCore)
        add(junit4Compose)
    }

    //Only meant for testImplementation("...")
    val testLibs = arrayListOf<String>().apply {
        add(junit)
    }
}

//util functions for adding the different type dependencies from build.gradle file
fun DependencyHandler.kapt(list: List<String>) {
    list.forEach { dependency ->
        add("kapt", dependency)
    }
}

fun DependencyHandler.implementation(list: List<String>) {
    list.forEach { dependency ->
        add("implementation", dependency)
    }
}

fun DependencyHandler.androidTestImplementation(list: List<String>) {
    list.forEach { dependency ->
        add("androidTestImplementation", dependency)
    }
}

fun DependencyHandler.testImplementation(list: List<String>) {
    list.forEach { dependency ->
        add("testImplementation", dependency)
    }
}
```

In order to use the grouping of libraries into a single variable declaration, we need the extension functions as shown in the image.

Such as -
fun DependencyHandler.kapt() {...}
fun DependencyHandler.implementation() {...}

These are the benefits of using Kotlin programming with Gradle Script.

```
fun DependencyHandler.debugImplementation(list: List<String>) {
    list.forEach { dependency ->
        add("debugImplementation", dependency)
    }
}
```

Step3:

All build.gradle files should be converted to build.gradle.kts files as shown in the image.

Change the app/build.gradle.kts file as follows.

```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("name.remal.check-dependency-updates") version "1.5.0"
}

android {
    compileSdk = AppConfig.compileSdk

    defaultConfig {
        applicationId = "com.app.todo_app_android"
        minSdk = AppConfig.minSdk
        targetSdk = AppConfig.targetSdk
        versionCode = AppConfig.versionCode
        versionName = AppConfig.versionName

        testInstrumentationRunner = AppConfig.testInstrumentationRunner
        vectorDrawables {
            useSupportLibrary = true
        }
    }

    buildTypes {
```

```
        getByName("release") {
            isMinifyEnabled  = false
            proguardFiles(getDefaultProguardFile("proguard-android-optimize.txt"),
"proguard-rules.pro")
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
    buildFeatures {
        compose = true
    }
    composeOptions {
        kotlinCompilerExtensionVersion = Versions.composeCompilerVersion
    }
    packagingOptions {
        resources {
            excludes += "/META-INF/{AL2.0,LGPL2.1}"
        }
    }
}

dependencies {
    androidTestImplementation(Dependencies.androidTestLibs)
    testImplementation(Dependencies.testLibs)
    debugImplementation(Dependencies.debugAppLibs)

    implementation(Dependencies.appLibs)

    implementation(Dependencies.coroutinesCore)
    implementation(Dependencies.coroutinesAndroid)
}
```

Here you will see that we have used the functions and variables of Kotlin file (i.e. Dependency.kt) into our build.gradle.kts file.

This is only possible with Kotlin DSL.

Open project/build.gradle.kts file and change it as given below

```
buildscript {

}// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id("com.android.application") version "7.1.3" apply false
    id("com.android.library") version "7.1.3" apply false
    id("org.jetbrains.kotlin.android") version "1.6.21" apply false
}

tasks.register("clean", Delete::class) {
    delete(rootProject.buildDir)
}
```

Open project/settings.gradle.kts file and change it as given below

```
pluginManagement {
    repositories {
        gradlePluginPortal()
        google()
        mavenCentral()
    }
}
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
    }
}
rootProject.name = "todo-app-android"
include(":app")
include(":data")
```