

Gradle in Android

- ال gradle هو build automation tool يستخدم في عمل manage build process في ال android projects
- ال gradle يعتبر هو المايسترو اللى بيحدد كل task هيبدا امتى وهيعمل أليه بالظبط زي ما هنشوف قدام

Structure of gradle files

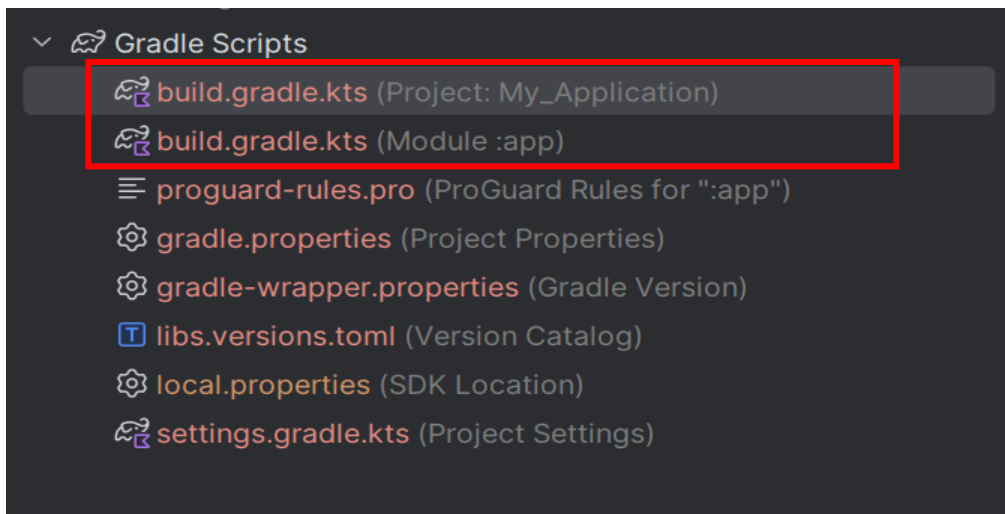
- في files كتيره خاصة بال gradle هنتكلم عنهم كلهم ولكن أهم files هما ال files الخاصة بال build.gradle ودول نوعين

Module Level

build.gradle.kts (Module :app)

Project Level

build.gradle.kts (Project: [project name])



- دلوقتى هنتكلم عن كل واحد منهم والفرق بينهم

■ ايه الفرق بين ال build.gradle الخاص بال app وال build.gradle

الخاص بال project ؟

- إحنا كنا إتكلما عن الجزئية دى بالتفصيل لما كنا بنتكلم عن ال modularization ولكن باختصار قلنا إن اى module بعملة بيكون ليه build.gradle خاص بيه وال app هو عبارة عن module وده ال build.gradle الخاص بيه
- اما ال project ممكن يكون فيه اكر من module علشان كده بيكون فيه build.gradle خاص بال app module و build.gradle خاص بال project كلة
- فأى كود هكتبة فى ال build.gradle الخاص بال app هيطبق على ال app فقط الى هو عبارة عن module
- أما أى كود هكتبة فى ال build.gradle الخاص بال project هيطبق على كل ال project بما فيهم كل ال modules الى منهم ال app module

■ أولاً : build.gradle(Module :app)

- ال location بتاع ال file ده بيكون فى ال module نفسة app/build.gradle
- الغرض منه هو إني بحدد فيه ال configuration الخاص بال module زى ما هنشوف
- لو فتحت ال build.gradle file الخاص بال app module هلاقى كذا block هنتكلم عنهم دلوقتى :

1-plugin Block

- ✓ ال plugins دى هي خاصة بال app module فقط يعنى مش هقدر أستخدمها خارج ال app module فى باقى ال project

```
plugins {  
    id ("com.android.application")  
    id ("kotlin-android")  
}
```

2-android Block

- ✓ ال android block فيه كذا attribute وكذا block هنتكلم عنهم كلهم بالتفصيل

❖ Namespace

```
namespace = "com.example.myapplication"
```

com.example.myApplication



✓ ال app module يعتبر هو ال main module لل project كلة علشان كدة ال namespace هو ال application Id

■ إيه فائدة ال namespace ؟

- ال namespace بيخلي ال resource وال classes الى في ال module تكون متعرفة بشكل unique (uniquely identifiable) علشان ميحصلش conflict زي مثلاً لو عندى two module وكل واحد منهم فيه class خاص بـ feature معينة وكان ليهم نفس الإسم من غير ال namespace هيجعل conflict ولكن باستخدام ال namespace هحدد كل class تبع أنه module

❖ compileSdk

- هنا هحدد ال android Sdk version الى ال app هيجصله compile عليه

```
compileSdk = 35
```

- معنى إني أحدد ال compileSdk يكون 35 يعنى ال project ده هيجصله compile على ال android sdk هو 35 يعنى هستخدم ال feature الخاصة بـ 35

❖ defaultconfig Block

✓ فائدة ال block ده إني بحدد فيه ال default configuration لل application

```
defaultConfig {  
    applicationId = "com.example.islam"  
    minSdk = 24  
    targetSdk = 35  
    versionCode = 1  
    versionName = "1.0"  
    testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
}
```

+ applicationId

✓ ال applicationId هو عبارة عن Unique identifier لل app في ال project او على google play ممكن يكون فيه اكثر من app ليه نفس الإسم على ال store ولكن مينفعش يكون في اكثر من app ليه نفس ال applicationId لانه unique

+ minSdk

✓ معنى ان ال minSdk هو 24 يعني ال app مش هيشغل على أى device ال Api Level بتاعة أقل من 24

+ targetSdk

✓ معنى إن ال targetSdk هو 35 إن ال app مصمم إنه ي run على device ال api level بتاعة هو 35 ويفضل إن ال compileSdk يكون نفس ال targetSdk

+ versionCode

✓ ده عبارة عن رقم integer بيمثل رقم ال version مهم في عمل ال update لل app على ال app store ومينفعش يتكرر لازم يحصل increment وانا برفع ال app على ال store

+ versionName

✓ ده عبارة عن string ممكن يكون بأي format زي مثلاً "4.1" او "5" او "1.9.4" وفائدة انه بيظهر لل user على ال app store ال version بتاع ال app

+ testInstrumentationRunner

✓ ده خاص بال testing بيحدد ال test runner علشان يستخدم ل running instrumented tests

❖ buildFeatures Block

✓ باستخدام ال Block بتاع ال buildFeatures علشان أعمل enable ل features معينة زي مثلاً ال viewBinding او databinding او BuildConfig

```
buildFeatures {  
    buildConfig = true  
    viewBinding = true  
}
```

❖ buildType Block

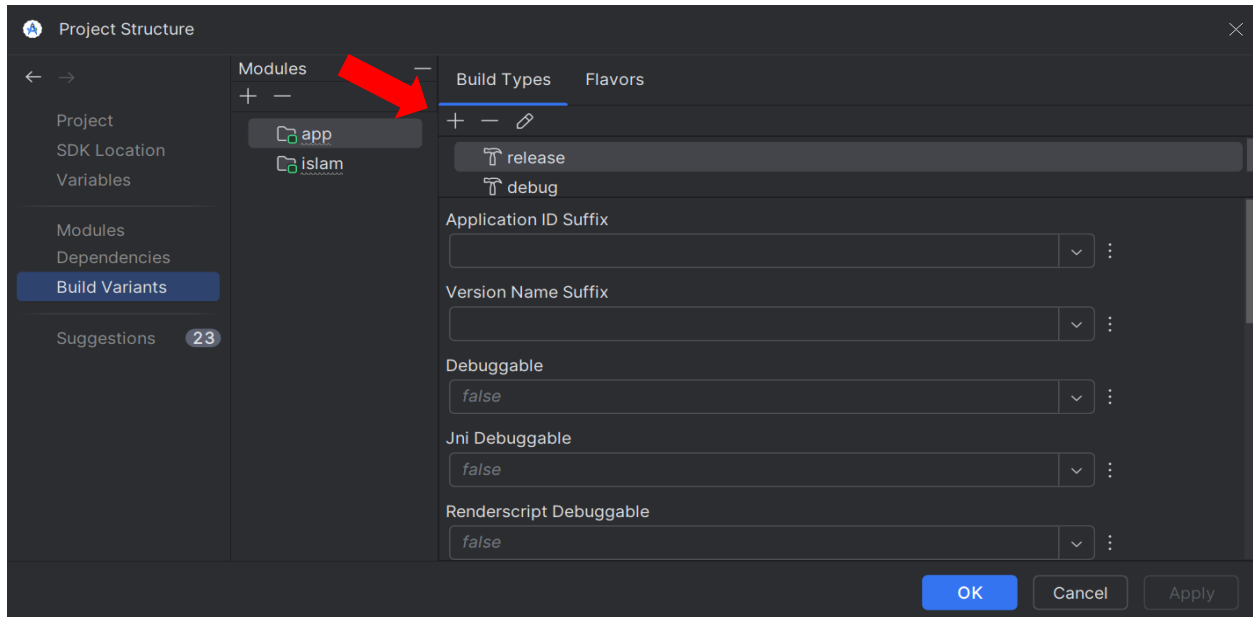
✓ أى app بيكون ليه two default buildTypes وهما ال release وال debug بيتعملهم create مع ال project

```
buildTypes {  
    release {  
        isMinifyEnabled = false  
        proguardFiles(  
            getDefaultProguardFile(name: "proguard-android-optimize.txt"),  
            "proguard-rules.pro"  
        )  
    }  
}
```

- ال release هى النسخة ال production الى هى النسخة بتطلع لل client الى بتترفع على ال play store ويستخدمها فى الآخر
- اما ال debug دى النسخة الى بيستخدمها ال developer او ال tester بمعنى انه يقدر يعمل track لل issue أو الكود الى عندى

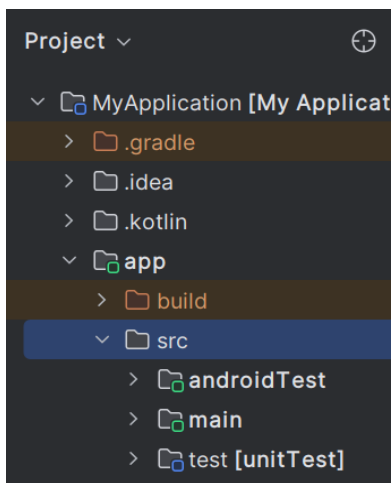
✓ ممكن أضيف أى نوع ثانى بسهولة

File → project structure → Build Variants



✓ هلاحظ ان فيه attributes لل build type زى مثلاً ال application ID Suffix فى ال debug لو حددته debug. وده هيضيف كلمة debug لل package name الأساسية

▪ لو عايز النسخة ال debug تكون اسم ال app مختلف عن الإسم فى النسخة ال release



- 1- فى البداية هتأكد إننا فى ال project View
- 2- بعد كدة هفتح ال app module وهفتح ال src
- 3- هعمل directory وهختار Debug\res
- 4- هعمل resource file إسمه strings
- 5- هلاقية عمل override لملف ال strings بحيث أقدر أغير فيه

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">new name</string>
</resources>
```

- لو عملت run هلاقى ال app اتعمله install على ال device بالإسم الجديد وهو فى ال debug mode
- وبنفس الطريقة ممكن أغير فى اى resource يعنى يكون فى ال release حاجة معينة وفى ال debug حاجة تانية

▪ لو عندى كود معين وعايضة يتنفذ فى حالة ال debug وميتنفذش فى حالة ال release :

1- فى البداية هفعل ال BuildConfig بالشكل ده :

```
buildFeatures {
    buildConfig = true
}
```

2- هكتب الكود بالشكل ده :

```
if (BuildConfig.BUILD_TYPE == "debug") {
    Toast.makeText(context: this, text: "hello debug", Toast.LENGTH_LONG).show()
}
```

- لو عملت run هلاقيه فى حالة ال debug عمل ال toast وفى حالة ال release مش هيعمل ال toast

productFlavors

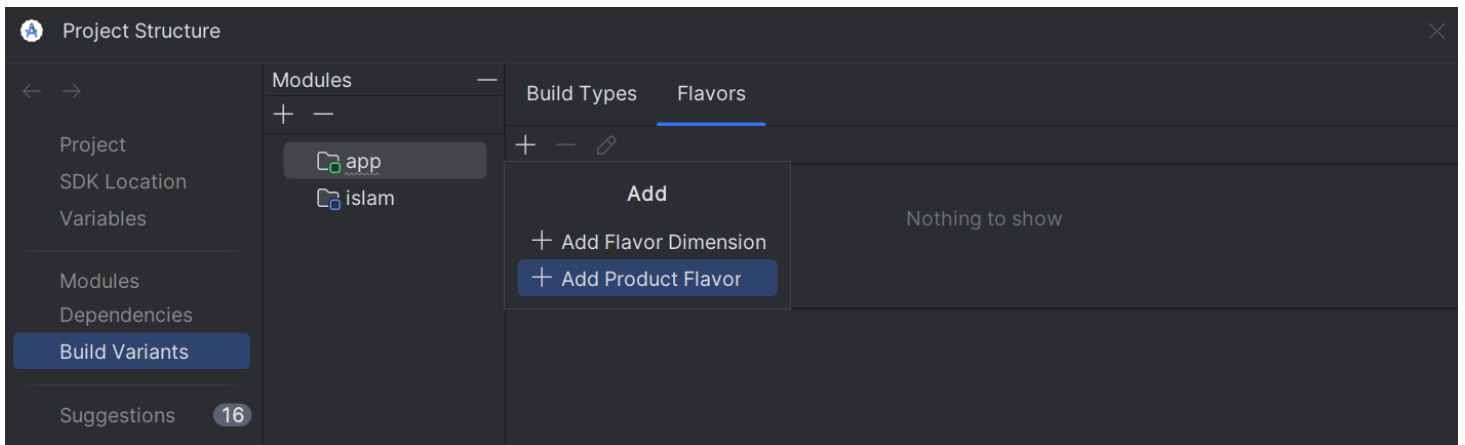
- By default مش بيكون فيه اى flavor في ال app في البداية ولكن بيكون فيه two build type الى هما ال debug وال release الى اتكلمنا عنهم

■ أيه الفرق بين ال buildType وال productFlavors :

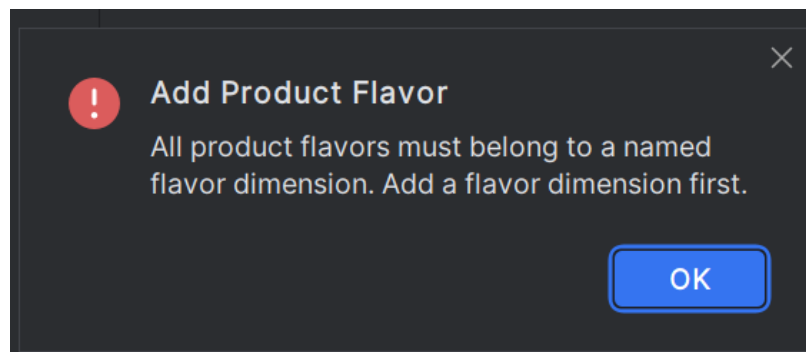
- ال buildType بيطلعك type من ال build مختلف سواء كان debug او release اما ال product flavors بيطلعك apk مع خصائص مختلفة يعني ايه :
- يعني مثلاً لو عندى app ومحتاج أطلع منه نسخة free ونسخة pro إذا كل واحد منهم هيطلع نسختين الى هما ال debug وال release بحيث إني أقدر أعمل test للنسخة من كل الجوانب وتطلع النسخة وأقدر أرفعها

■ إزاي أضيف Flavors ؟

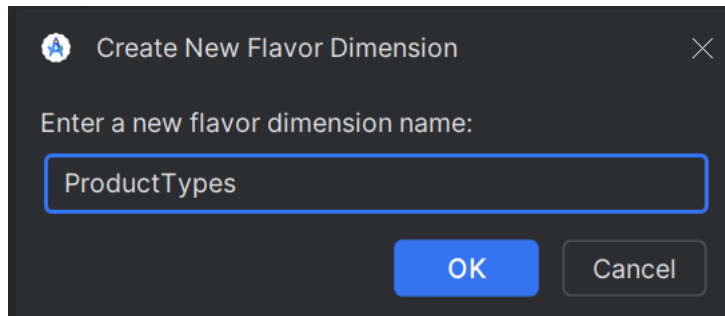
- لو فرضنا إن عندى app وعايز أعمله نسخة مدفوعة ونسخة مجانية
file --> project structure --> flavors



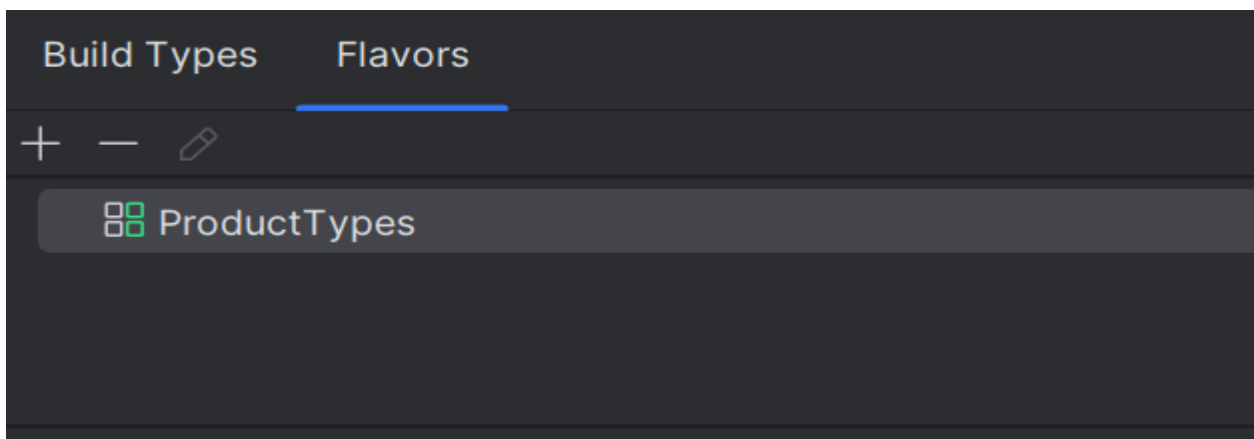
- ✓ لو ضغط Add product Flavor هلاقية بيقولى إنك لازم يكون عندك flavor dimension



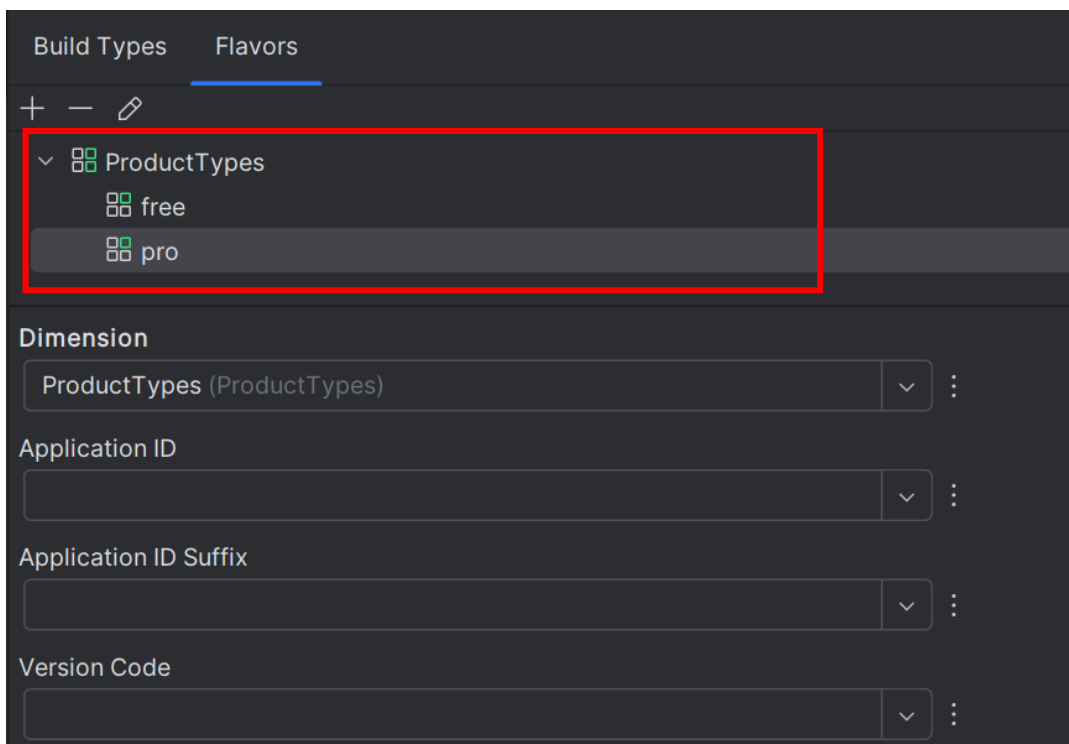
- علشان كدة هحدد الإختيار الأول Add Flavor Dimension هيفتح ال dialog ده هحدد ال flavor dimention name



- هتظهر بالشكل ده :



- هضيف ال product flavor الأول هو ال free والثاني هو ال pro بالشكل ده :



هلاحظ إن فيه attributes لكل flavor ممكن احدها :

- زي مثلا ال version code وال minSdk وغيرها ممكن أحدها لكل واحد من ال flavor
- دلوقتى هحدد مثلاً ال ApplicationID suffix بالشكل ده :

Free --> .free pro --> .pro

- لو رحت لل build.gradle الخاص بال app هلاقية bydefault ضاف الكود ده :

```
flavorDimensions += listOf("ProductTypes")
productFlavors {
    create( name: "free") {
        dimension = "ProductTypes"
        applicationIdSuffix = ".free"
    }
    create( name: "pro") {
        dimension = "ProductTypes"
        applicationIdSuffix = ".pro"
    }
}
```

- اقدر أضيف اى resource فى أى flavor واقدر كمان اضيفه فى flavor معين وال flavor الثانى مضفيهوش بنفس الطريقة الى فاتت

Product view --> src --> generate directory --> free\res

- هنا هعمل أى resource انا عايزة بنفس الطريقة الى فاتت
- ممكن أتحكم فى تنفيذ اى كود فى أى flavor بنفس الطريقة الى فاتت بالشكل ده :

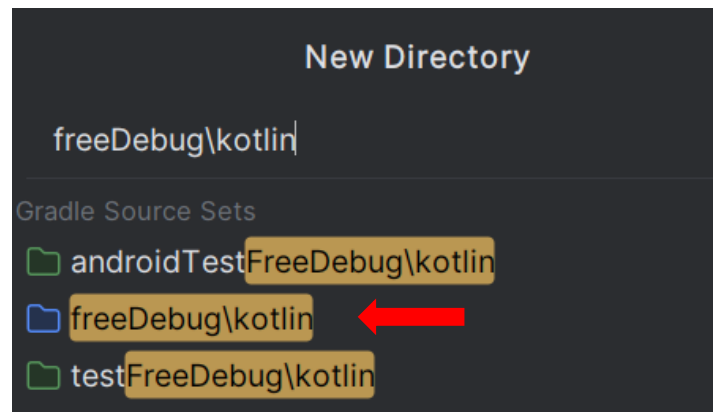
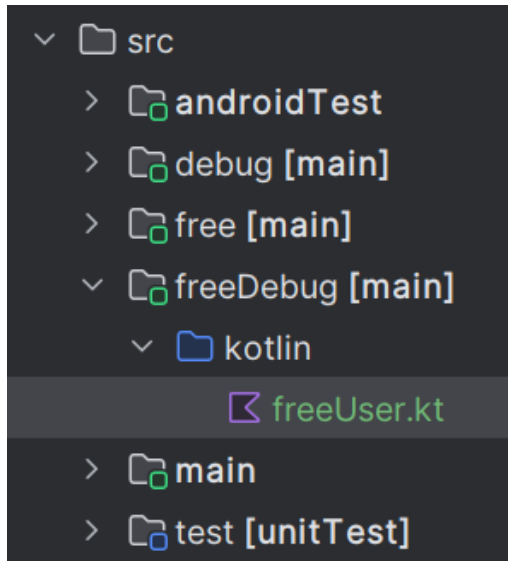
```
if (BuildConfig.FLAVOR == "free") {
    Toast.makeText( context: this, text: "hello free", Toast.LENGTH_LONG).show()
}
```

- كده الكود ده هيتنفذ فى ال free ومش هيتنفذ فى ال pro

➤ في طريقة ثانية ممكن أنفذ بيها كود مختلف في كل Flavor :

■ الخطوة الأولى :

- 1- هعرض ال project في ال project View
- 2- بعد كده لو حددت ال build variant مثلا free debug وبعد كدة هعمل
src <-- directory <-- freeDebug\kotlin



- 3- هلاقية عمل folder إسمه kotlin هعمل class أو file وهعمل فيه ميثود مثلاً بالشكل ده مش هعرف أعمل access إلا لو كنت في ال free Debug

■ الخطوة الثانية:

- 1- هعمل نفس اللى عملته ولكن هحدد build variant مثلاً pro debug وبعد كده هعمل
src <-- directory <-- proDebug\kotlin
- 2- هلاقية عمل folder إسمه kotlin هعمل فيه class أو file بنفس الإسم اللى عملته قبل كده ونفس النظام مش هعرف اعمل access إلا لو كنت في ال pro Debug

■ الخطوة الثالثة:

- لو جيت أعمل run في ال free هيتنفذ الكود اللى عملته في ال free Debug ولو عملت run في ال pro هيتنفذ الكود اللى في ال pro Debug

BuildConfig file

هو عبارة عن class معمول generate والقيم اللى فيه بتكون static

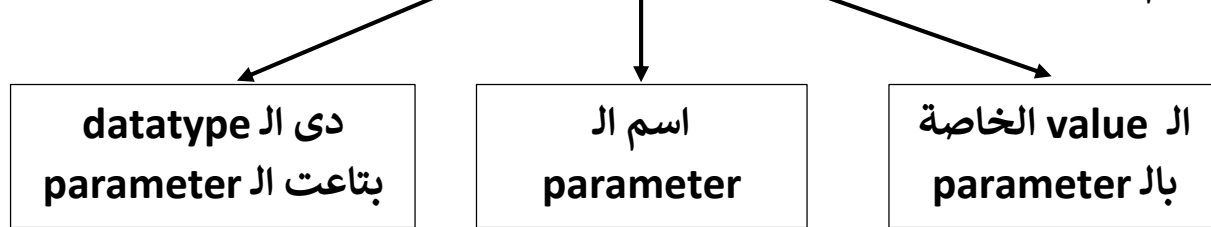
```
public final class BuildConfig {  
    no usages  
    public static final boolean DEBUG = Boolean.parseBoolean(s: "true");  
    no usages  
    public static final String APPLICATION_ID = "com.example.myapplication.free.debug";  
    1 usage  
    public static final String BUILD_TYPE = "debug";  
    1 usage  
    public static final String FLAVOR = "free";  
    no usages  
    public static final int VERSION_CODE = 1;  
    no usages  
    public static final String VERSION_NAME = "1.0";  
}
```

علشان افعله بروج عند ال build.gradle الخاص بال module وأضيف الكود ده

```
buildFeatures {  
    buildConfig = true  
}
```

■ إزاي أضيف parameter في ال BuildConfig :

هستخدام الميثود <-- buildConfigField("Boolean","isHappy","false")



فيه 3 طرق ممكن أضيف بيها parameter في ال BuildConfig بالشكل ده :

الطريقة الأولى : defaultConfig

الميزة في الطريقة دي اني اقدر اعمل ال parameter واقدر اوصلة في اي مكان

```
defaultConfig {  
    applicationId = "com.example.myapplication"  
    minSdk = 24  
    targetSdk = 35  
    versionCode = 1  
    versionName = "1.0"  
    testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
    buildConfigField( type: "Boolean", name: "isHappy", value: "false")  
}
```

بعد ما أعمل sync و build ال parameter هينضاف لـ class وأقدر اعمله access بسهولة بالشكل ده

```
Toast.makeText( context: this, BuildConfig.isHappy.toString(), Toast.LENGTH_LONG).show()
```

لو عملت run هلاقى عمل toast بالقيمة اللى في ال class اللى هي false

الطريقة الثانية : buildType

لو ضففة مثلاً في ال release فقط ده معناة إني مش هقدر أستخدمه غير في ال release نفس النظام بالنسبة لل debug

```
buildTypes {
    release {
        isMinifyEnabled = false
        proguardFiles(
            getDefaultProguardFile( name: "proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }
    debug {
        buildConfigField( type: "String", name: "name", value: "\"islam basher\"" )
    }
}
```

هعملة access بنفس الطريقة الى فاتت بسهولة

الطريقة الثالثة : flavors

لو ضففة في ال flavor الى هي مثلاً free ده معناة إني مش هقدر أستخدمه غير في ال flavor دي فقط سواء ال debug او ال release

```
flavorDimensions += listOf("ProductTypes")
productFlavors {
    create( name: "free" ) {
        dimension = "ProductTypes"
        applicationIdSuffix = ".free"
        buildConfigField( type: "Integer", name: "age", value: "18" )
    }
    create( name: "pro" ) {
        dimension = "ProductTypes"
        applicationIdSuffix = ".pro"
    }
}
```

❖ compileOptions Block

باستخدام ال Block ده علشان أحدد توافق الجافا مع ال source code وال bytecode

```
compileOptions {  
    sourceCompatibility = JavaVersion.VERSION_1_8  
    targetCompatibility = JavaVersion.VERSION_1_8  
}
```

sourceCompatibility = javaVersion.VERSION_1_8

■ السطر ده بيحدد التوافق مع ال source code يعني إيه الكلام ده :

- يعني بحدد إن ال source code هستخدم ال features الموجودة في java 8 معنى كدة إني ممكن أستخدم ال features دي زي مثلاً stream API , lambda expressions وغيرها من ال features الموجودة في java 8 من غير ما أضطر أواجه مشاكل خاصة بال compatibility

targetCompatibility = javaVersion.VERSION_1_8

■ السطر ده بيحدد التوافق مع ال bytecode يعني إيه:

- يعني بحدد إن ال bytecode اللى اتعمله compile لازم يكون compatible مع java 8
- يعني لو فيه حاجة فوق java 8 مينفعش الاقيها في ال bytecode وبالشكل ده هضمن إن ال generated class file ممكن ي run على java 8 runtime environment حتى لو مستخدم version جديد من ال sdk في عمل ال compile
- لما بختار ال JVM لازم يكون متناسب مع ال minSdk لان مثلاً لو إخترت JVM قليل مع minSdk عالي مش هيعرف يشتغل

- قبل ما نتنقل على ال Block الى بعده عايزين نتكلم عن الفرق بين :

Java biteCode Viersion Vs Java Version

أولاً : Java Version

- ال java version كل وظيفة إنة يحدد ال java feature الى هنستخدمها ونفس النظام بالنسبة لل kotlin <-- kotlin version

ثانياً : Java bytecode Version

- ده بيحدد ال java bytecode الى بيكون ناتج عن ال compile الى حصل لل source code وكل version جديد بيكون فيه إضافة زي مثلاً capabilities و optimizations وغيرهم
- ال bytecode version ملوش علاقة بال language feature ودة معناة إني مثلاً ممكن أستخدم 18 --> java bytecode version ولكن مقدرش أستخدم ال features الموجودة في java 18 دي حاجة ودي حاجة تانية

لازم اكون مختار java version و java bytecode version بشكل صحيح ودة لان احيانا مثلاً بكون مستخدم library تكون ب target ل JVM معين ممكن ال library لما يحصلها compile تطلع bytecode مش compatible مع ال JVM الى حدتة

❖ kotlinOptions Block

```
kotlinOptions {  
    jvmTarget = "1.8"  
}
```

jvmTarget = "1.8"

- ده بيحدد إن ال kotlin code لازم ي compile ل java bytecode يكون compatible مع java 8 ده شبة ال targetCompatibility
- وده يضمن إن أى كود kotlin ممكن يستفاد من ال features الموجودة في java 8 وكمان أعملة run على java 8 JVM

3- dependencies Block

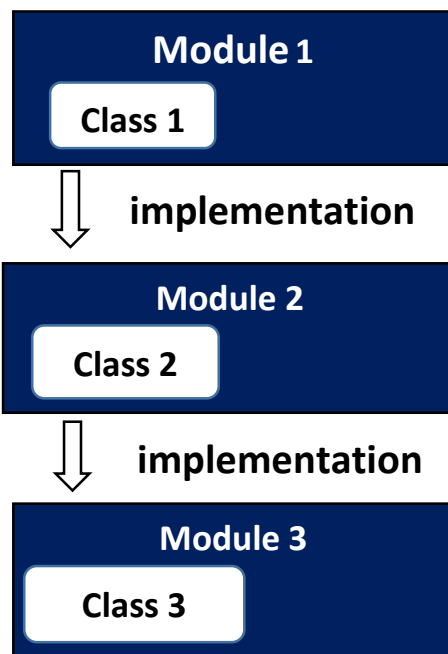
- ال dependencies دي بنضيف فيها ال library اللى هحتاجها في الكود
- في كذا keyword بنستخدمها :

+ implementation

- دي بنستخدمها كتير وعن طريقها بنضيف اى library او حتى module تاني ونقدر نستخدمها في أى مكان في ال module سواء package او test

+ api

- دي نفس نظام ال implementation وكنا إتكلنا عن الفرق بينهم قبل كدة
- هفرض إن عندي ثلاث modules وكل module فيه كلاس بالشكل ده :



- بعد كده هعمل implementation بالطريقة اللى إتكلنا عنها قبل كده اللى هي **Implementation(project(---))**
- السؤال دلوقتي هل module1 يعرف class3 ؟
- الإجابة لا وده معنى كلمة implementation إنه بيعمل implement ولكن لا module نفسه فقط وليس باقي ال modules اللى بيعتمد عليها

هل ينفع أعمل الحالة المعاكسة ؟ يعني أخلى مثلاً module1 يعرف class3 ينفع طبعا وكل اللى هعمله اني هغير كلمة implementation بكلمة api في module2 بالشكل ده : **api(project(":module3"))**

- بالشكل ده أقدر أوصل ل class3 في module1

- + debugImplementation
 - دي معناها إن ال libraries دي مش هتستخدم غير في ال debug فقط
- + releaseImplementation
 - دي معناها إن ال libraries دي مش هتستخدم غير في ال release فقط
- + testImplementation
 - دي خاصة بال implementation اللي ليه علاقة بال Local unit test
- + androidTestImplementation
 - دي خاصة بال implementation اللي ليه علاقة بال test اللي محتاج ال emulator يكون شغال وقت ال test اللي هو ال android test

■ ثانياً : build.gradle(project :project name)

- ال file ده بحدد فيه ال configuration اللي عايزها تطبق على ال project بالكامل
- نفس الكود اللي إنكلنا عنة في ال build.gradle الخاص بال app ممكن أكتبه هنا وأستخدمه في ال build.gradle(project) ولكن هنا زى ما قلنا هيطبق على ال project بالكامل
- ال location الخاص بال build.gradle بيكون موجود في ال root project

```
plugins {
    id("com.android.application") version "7.0.4" apply false
    id("org.jetbrains.kotlin.android") version "1.5.31" apply false
    id("com.android.library") version "7.0.4" apply false
}
```

- ال plugins دي المفروض إنها تطبق على ال project بالكامل ولكن طالما حددت إنها ب false ده معناه انها مش هتطبق لا على ال project ولا ال modules

■ ثالثاً settings.gradle.kts:

- ال settings.gradle هو جزء من ال gradle build system بنستخدمه علشان أعمل configure و manage لل multi-module project
- ال settings.gradle سيكون فيه كذا حاجة لعمل ال management لل project هنتكلم عنهم دلوقتى :

1- pluginManagement

```
pluginManagement {
    repositories {
        google {
            content {
                includeGroupByRegex( groupRegex: "com\\.\\.android.*")
                includeGroupByRegex( groupRegex: "com\\.\\.google.*")
                includeGroupByRegex( groupRegex: "androidx.*")
            }
        }
        mavenCentral()
        gradlePluginPortal()
    }
}
```

- الميثود pluginManagement() بتكون فيها ميثود إسمها repositories() فائدة الميثود دى إنى لما بضيف أى plugin يروح يدور عليها فى ال repositories دى

2- dependencyResolutionManagement

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven { url = uri( path: "https://jitpack.io" ) }
    }
}
```

- ال repositories() دى بتكون خاصة بال dependencies يعنى أى dependencies موجودة فى أى module يروح يدور عليها فى ال repositories دى

- لو عندي مثلاً library بالشكل ده :

```
implementation("com.github.davidmigloz:number-keyboard:4.0.6")
```

- لو عملت run هلاقى حصل error ده :

```
Sync x Build Output x Build Analyzer x
> Could not find com.github.davidmigloz:number-keyboard:4.0.6.
Searched in the following locations:
- https://dl.google.com/dl/android/maven2/com/github/davidmigloz/number-keyboard/4.0.6/number-keyboard-4.0.6.pom
- https://repo.maven.apache.org/maven2/com/github/davidmigloz/number-keyboard/4.0.6/number-keyboard-4.0.6.pom
Required by:
  project :app
```

- هو بيقولى إنة مش عارف يوصل لل dependency ولو رحت لل website الى جبت منه ال dependency هلاقيه شارح الطريقة الى هستخدم بيها ال library وبيقولى إنك مفروض تضيف الكود ده :

Step 1

Add the JitPack repository to your `build.gradle` file:

```
allprojects {
    repositories {
        ...
        maven { url "https://jitpack.io" }
    }
}
```

- هضيفه في ال project عندي بالشكل ده :

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven { url = uri( path: "https://jitpack.io" ) }
    }
}
```

- ولو عملت run هلاقى ال library إنضافت وأقدر أستخدمها بسهولة

3- name of the root project

```
rootProject.name = "My Application"
```

- هنا بيحدد ال project name الى انا حددته لل project

4- include function

```
include( ...projectPaths: ":app")
```

- ال function دي مهمة جداً لاني بحدد فيها ال modules الى موجودة عندي في ال project

■ رابعاً : gradle.properties

- هو عبارة عن configuration file بيستخدم في ال gradle build علشان أحدد ال properties وال settings زي ما هنشوف كمان شوية
- ال gradle properties بيخليني أقدر أعمل customize لجوانب كثيرة من ال build process من غير ما أضطر أغير في ال build script

➤ أيه الحاجات الى أقدر أكتبها في ال gradle.properties file ؟

1- Define properties

- ممكن أعرف فية properties وأقدر أوصلة في أى مكان في ال gradle build scripts زي مثلاً ال build.gradle file

```
hilt_version = 2.48
```

بعد كده هقدر أوصلة في أى build scrip file بالشكل ده :

```
val hilt_version = project.findProperty("hilt_version").toString()
```

```
implementation("com.github.davidmigloz:number-keyboard:$hilt_version")
```

2- Configuration settings

- بحدد ال build configuration settings

```
org.gradle.jvmargs=-Xmx8192m -XX:+HeapDumpOnOutOfMemoryError -Dfile.encoding=UTF-8
```

- هنا مثلاً بحدد JVM arguments اللى الـ gradle بيستخدمها وهو بـ run الـ arguments
دى بتعمل allocate memory, performance options, encoding

-Xmx8192

- ال option ده بيحدد ال maximum heap size اللى الـ jvm هستخدمة وهو 8 GB زيادة

-XX:+HeapDumpOnOutOfMemoryError

- ده عبارة عن flag بيعمل create لـ heap dump بحيث لو حصل OutOfMemoryError

-Dfile.encoding=UTF8-

- ده بيحدد ال default file encoding للـ jvm هو UTF-8

```
org.gradle.daemon=true
```

- ده بيعمل enable لـ gradle daemon process بيسرع عملية الـ build

```
org.gradle.parallel=true
```

- ده بيخلي ال gradle ممكن يعمل execute لأكتر من task بشكل parallel لو خلقت قيتة
بـ true ده هيخليه ينفذ أكتر من task فى نفس الوقت طبعاً بيزود ال performance
وكمان بيحسن سرعة ال build عن طريق إستخدام multiple cpu core

```
org.gradle.configureondemand=true
```

- لما ال feature دى بتكون true ده بيخلي ال gradle يعمل configure للـ modules
المطلوبة للـ tasks اللى بتنفذ حالياً بدل ما يعمل configure لكل ال modules

```
android.enableAapt2=true
```

- ده بيعمل enable لـ AAP2 ودة إختصار لـ Android Asset Packaging Tool2 فائدة
إنه بيعمل packaging لكل ال files اللى موجودة فى ال project علشان يطلع ال apk



```
org.gradle.caching=true
```

- ده بيعمل enable ل gradle build cache ودة بيخليه يستخدم ال output من عمليات ال build السابقة ودة ييزود سرعة ال build

```
android.useAndroidX=true
```

- ده بيعمل enable ل AndroidX dependencies libraries ودى تعتبر أفضل من Android libraries القديم

```
android.enableJetifier=true
```

- ده بيعمل enable ل Jetifier tool وال tool دى بتساعد فى عمل compatibility بين ال Older Android libraries وال AndroidX libraries

■ خامساً : local.properties

- ودة file من ضمن ال files اللى بتكون موجودة عندى فى ال project وهو عبارة عن simple text file بيتعملة generate من الأندرويد استديوا
- فى حاجة مهمة هنا ال file ده bydefault بيكون موجود فى ال git.ignor يعنى هو من ضمن ال files اللى هيتم تجاهلهم لما برفع ال project على أى version control

➤ أيه فائدة ال local proferties file ؟

1-Environment-specific configuration

- ممكن أحدد فيه ال configuration settings زى مثلاً sdk paths

```
sdk.dir=C:\\Users\\EEdbeshir\\AppData\\Local\\Android\\Sdk
```

2-Sensitive information

- لو عندى sensitive information زى مثلاً api_key ممكن أعمله فى ال local properties زى ما هنشوف كمان شوية

3-Convenience

- ممكن أوضح أو أشرح خطوات أى حاجة انا عايزها لـ new developer

➤ دلوقتي عايزين نخزن ال Api_key في ال local properties :

الخطوة الأولى :

- هروح عند ال local properties وهكتب ال api_key هناك بالشكل ده :

```
api_key=12345-abcde-67890-fghij-klmno
```

الخطوة الثانية :

- هروح عند ال build.gradle(app) في ال default config وهضيف الكود ده :

```
val properties= Properties()
properties.load(project.rootProject.file("local.properties").InputStream())
buildConfigField( type: "String", name: "api_key", value: "\"${properties.getProperty("api_key")}\"")
```

الخطوة الثالثة :

- هعمل rebuild لل project

الخطوة الرابعة :

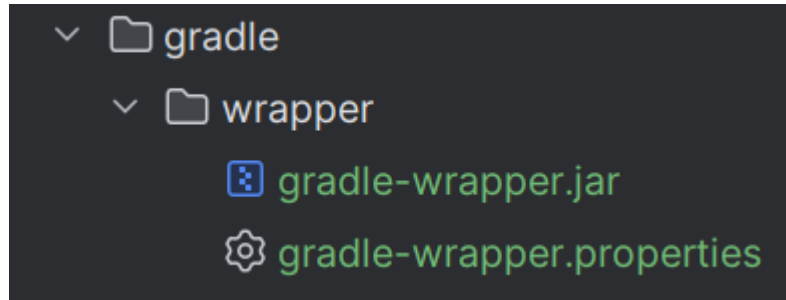
- كدة هقدر أستخدم ال api_key في ال app بسهولة

```
val api_key = BuildConfig.api_key
```


Gradle wrapper file

➤ ايه هو ال gradle wrapper ؟

- ال (gradlew) gradle wrapper هو عبارة عن application صغير بيكون ضمن ال source code وظيفته يعمل download و launch لل gradle وده ببسهل ال build execution



- لو فتحت folder ال gradle هلاقى فيه folder لل wrapper ال folder ده فيه two files :

اولاً: gradle-wrapper.jar

- ده عبارة عن Java Archive (JAR) ال file ده بيحتوى على ال logic المطلوب لعمل download لل gradle distribution

➤ ايه هو ال gradle distribution ؟

- هو عبارة عن مجموعة ال versions of the gradle build tool ودى اللى هتعمل download و execute لل gradle commands
- كل distribution بيحتوى على ال files اللى هحتاجها وقت ال run زى مثلاً libraries, scripts, other resources

ثانياً: gradle-wrapper.properties

- هو عبارة عن file بيحتوى على مجموعة properties هنتكلم عنهم دلوقتى

```
distributionBase=GRADLE_USER_HOME
```

- ال property ده بيحدد ال path اللى فيه ال base directory اللى هيتخزن فيه ال gradle distribution

GRADLE_USER_HOME

- ده بيشير إلى ال path نفسه عادة في ال windows بيكون

`% userProfile % \.gradle --> C:\users\username\.gradle`

```
distributionPath=wrapper/dists
```

- ده عبارة عن relative path لل distributionBase وهو بيحدد مكان اللى هيتعمل فيه ال download لل gradle distribution

`C:\users\username\.gradle\wrap per\dists`

```
distributionUrl=https\://services.gradle.org/distributions/gradle-8.4-bin.zip
```

- ده أهم property هو في البداية بيعمل check هل ال version ده من ال gradle موجود ولا لا لو موجود تمام مش هيعمل download ولكن لو مش موجود هيعمل download من ال url ده اللى متحدد في ال property

```
zipStoreBase=GRADLE_USER_HOME
```

- ده شبة ال distributionBase لانه بيحدد ال base directory اللى بيتخزن فيه ال zip files

```
zipStorePath=wrapper/dists
```

- ده شبة ال distributionPath لانه عبارة عن relative لل zipStoreBase وكمان بيحدد المكان اللى هيتعمل فيه ال download لل gradle zip files

`c:\users\username\.gradle\wrap per\dists`

➤ دلوقتي عايزين نعرف لما بعمل run لـ app إيه بالضبط اللي الـ gradle بيعمله بالترتيب :

1- Gradle starts the build process

- في البداية الـ gradle wrapper(./gradlew) بيتتم إستدعائه
- الـ gradle بيقرأ الـ configuration من الـ build.gradle file وبيحدد إنه task هيتنفذ بناءً على الـ build type إذا كان debug او release

2- Dependency Resolution

- الـ gradle بيعمل check و resolve لـ dependencies اللي حددتها في الـ build.gradle file فبيعمل download لـ library او module مهم من الـ repository اللي موجوده فيه اللي حددتها في الـ settings.gradle

3- Compilation Tasks

- الـ gradle بيعمل initiates لـ compile tasks لـ project بعد كده الـ compiler بيعمل compile لـ source code ويحولها لـ bytecode
- باقى الخطوات إتكلّمنا عنها قبل كده بالتفصيل قبل كدة لحد الـ user ما يقدر يستخدم الـ App

▪ How to Migrate Gradle Groovy to Gradle kotlin DSL

- فيه two versions من ال gradle الأول groovy والثاني kotlin
➤ **دلوقتي عايز أحول ال gradle من groovy الى kotlin :**
- الخطوة الأولى : هغير الأسماء الخاصة بال gradle files

- build.gradle(app) --> build.gradle.kts
- build.gradle(project) --> build.gradle.kts
- settings.gradle --> settings.gradle.kts

- الخطوة الثانية : بعد كدة هبدأ أعمل ال Migration
- أولاً : في ال groovy ممكن أستخدم single quotes وال double quotes
أما ال kotlin بيستخدم ال double quotes فقط
- فهغير مثلاً ال plugin من الشكل ده :

```
plugins {  
    id 'com.android.application'  
    id 'kotlin-android'  
}
```

- الى الشكل ده :

```
plugins {  
    id "com.android.application"  
    id "kotlin-android"  
}
```

- ونفس النظام مع كل ال strings الموجودة في ملفات ال gradle

ثانياً : نحول ال attributes الى مكتوبة بال groovy إلى function عن طريق إضافة الاقواس بالشكل ده :

```
plugins {  
    id("com.android.application")  
    id("kotlin-android")  
}
```

```
implementation("androidx.compose.ui:ui:1.5.0")  
implementation("androidx.compose.material:material:1.5.0")  
implementation("androidx.compose.ui:ui-tooling:1.5.0")  
implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.1")  
implementation("androidx.activity:activity-compose:1.7.0")
```

```
include( ...projectPaths: ":app")
```

ثالثاً : هحول ال fields الموجودة في ال groovy إلى variables
- من الشكل ده (groovy) :

```
defaultConfig {  
    applicationId "com.orange.mobinilandme"  
    minSdkVersion 24  
    targetSdkVersion 34  
    versionCode 166  
    versionName "9.0.0"  
    multiDexEnabled true  
  
    javaCompileOptions {  
        annotationProcessorOptions {  
            arguments['dagger.hilt.disableModulesHaveInstallInCheck'] = 'true'  
        }  
    }  
}
```

- الى الشكل ده (kotlin) :

```
defaultConfig {
    minSdk = 24
    targetSdk = 34
    versionCode = 166
    versionName = "9.0.0"
    multiDexEnabled = true
    javaCompileOptions {
        annotationProcessorOptions {
            arguments["dagger.hilt.disableModulesHaveInstallInCheck"] = "true"
        }
    }
}
```

رابعاً : بالنسبة لـ buildTypes مش هينفع إني أستخدم إسم ال buildType
مباشراً ولكن هنستخدم ميثود getName() من الشكل ده (groovy) :

```
buildTypes { NamedDomainObjectContainer<BuildType> it ->
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
}
```

من الشكل ده (kotlin) :

```
buildTypes {
    getByName(name: "release") {
        isMinifyEnabled = false
        proguardFiles(
            getDefaultProguardFile(name: "proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }
}
```

Gradle Catalogs

➤ ايه فائدة ال gradle Catalogs ؟

- لو عندى أكثر من module فى ال project بتاعى بيكون فيه حاجات كتير متكررة فى ال gradle لو إحتجت أعمل update مثلاً زى إني أعمل update لا versions الخاصة بال libraries فى الحالة دى هضطر إني أغير فى كل ال modules علشان كدة بنستخدم ال gradle catalogs
- أى library بتكون من 3 حاجات بالشكل ده :

androidx.core :core-ktx 1.8.0
group name version

- كمان ال plugin بتتكون من حاجتين :
- id("com.example.mycustomplugin") version "0.1.0"
- id version

➤ دلوقتى بمجرد ما أعمل project جديد ال catalogs بتكون موجودة

ولكن لو عندى project قديم مثلاً وعازب أعمله refactor وأخلى

يستخدم ال catalog ؟

- الخطوة الأولى :
- هخلي ال project يتعرض فى ال project view
- هروح عند ال gradle وهعمل file جديد وهسميه libs.versions.toml
- الخطوة الثانية :
- ال file بنقسمه الى 3 أجزاء بالشكل ده :

```
[versions]
[libraries]
[plugins]
```

- الخطوة الثالثة :

- هفرض ان عندى library بالشكل ده:

```
implementation("androidx.core:core-ktx:1.2.0")
```

- هفتح ال file بتاع ال catalogs هنكتب تحت كل section الجزء الخاص بيه بالشكل ده :

```
[versions]
ktx = "1.2.0"
[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "ktx" }
[plugins]
```

- كدة أقدر أستخدم ال library على طول عن طريق ال toml بالشكل ده

```
implementation(libs.androidx.core.ktx)
```

- الخطوة الرابعة :

- لوعندى plugin بالشكل ده :

```
id("com.android.library") version "8.0.2" apply false
```

- نفس النظام الى عملته مع ال library هعملة مع ال plugin هضيف ال plugin تحت ال section الخاص بيه

```
[versions]
ktx = "1.2.0"
agp="8.0.2"
[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "ktx" }
[plugins]
android-library = { id = "com.android.library", version.ref = "agp" }
```


- كدة أقدر أستخدم ال plugin على طول عن طريق ال toml بالشكل ده :

```
alias(libs.plugins.android.library) apply false
```

- بالشكل ده لو إحتجت أعمل update بدل ما كنت بعمل update في كل module هعملة في مكان واحد فقط

- الخطوة الخامسة :

- لو عايز اعمل نفس الكلام مع ال variables الموجودة في ال gradle هعملها بالشكل ده :

```
[versions]
ktx = "1.2.0"
agp = "8.0.2"
targetSDK = "35"
minSDK = "24"
```

- بعد كدة هستخدمهم بسهولة في ال build.gradle بالشكل ده :

```
minSdk = libs.versions.minSDK.get().toInt()
targetSdk = libs.versions.targetSDK.get().toInt()
```

Gradle script

- معظم ال files الى إتكلما عنهم عبارة عن Gradle script زي مثلاً :

- build.gradle
- settings.gradle
- gradle.properties

➤ إيه هو ال gradle scripts ؟

- هو عبارة عن file يستخدم في ال build Configurations project
هو عادة بيستخدم Domain-Specific-Language(DSL) سواء كان
kotlin او groovy

➤ إيه هو ال Domain-Specific-Language (DSL) ؟

- هو عبارة عن برنامج معين مصمم لمشاكل معينة في ال android
development وده بيسمح للمبرمجين إنهم يعبروا عن ال tasks او
ال concepts او ال functionalities بطريقة مختصرة وسهلة الفهم
بدل من إستخدام general purpose program language زي
الجافا او الكوتلن

➤ إيه الفرق بين ال DSL زي مثلاً kotlin DSL وال normal kotlin ؟

1- Purpose

- ال kotlin DSL بيكون في حالة ال Gradle مصمم لعمل build project وتنظيم ال dependencies وتنفيذ ال tasks
- اما ال Normal Kotlin هي تعتبر general purpose programming language هي بتستخدم في كتابة حاجات كتيرة سواء android app او ال desktop أو حتى ال backend

2- Structure & Syntax

- ال kotlin DSL بيستخدم keywords معينة خاصة بال build زي مثلاً
plugin{} , android{} , dependencies{} ال syntax بتكون مختصرة
أكثر ومعبرة عن ال tasks الخاصة بال build

- اما ال Normal Kotlin ده بيتبع standard kotlin syntax علشان يحدد ال classes او ال function او

3- Context and Execution Environment

- ال kotlin DSL ده بيتنفذ في سياق ال Gradle build lifecycle وده بييسمح إنه reference ل gradle specific API, configurations, tasks ودة معناة إنه بـ interact بشكل مباشر مع ال build system
- اما ال Normal Kotlin ده بـ run على أى kotlin compatible environment زي مثلاً JVM, Android وده مش بس محصور في عمليات ال build ولكن ممكن يعمل عمليات كبيرة في ال App او اي logic مطلوب

➤ إزاي أعمل Custom Gradle Script :

الخطوة الأولى : create new file

- هعرض ال project في Project View
- هعمل file جديد وليكن هسمية flavors.gradle

الخطوة الثانية : Define your custom task

- هضيف فيه الكود الى عايزة ينفذ وليكن الكود الخاص بال flavors بالشكل ده :

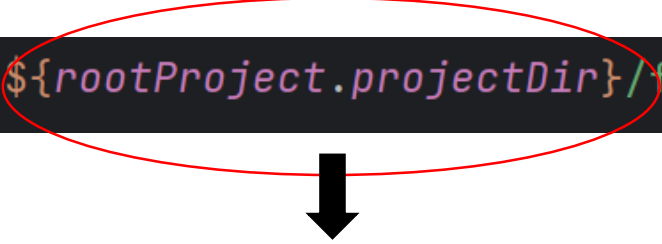
```
android {
    flavorDimensions ("ProductTypes")

    productFlavors {
        create("free") {
            dimension = "ProductTypes"
            applicationIdSuffix = ".free"
        }
        create("pro") {
            dimension = "ProductTypes"
            applicationIdSuffix = ".pro"
        }
    }
}
```

الخطوة الثالثة : Include the custom script in your build file :

- لو عندي أكثر من module في ال project بدل ما أضطر إني أكتب الكود ده في كل module كل اللى هعمله إني هعمل apply لل gradle script اللى عملته في كل module بالشكل ده :

```
apply(from = "${rootProject.projectDir}/flavors.gradle")
```



ده عبارة عن ال root الخاص بال project وده لاني عملت ال gradle script في ال root ويمكن أعمله في ملف ال gradle نفسه ولكن في الحالة دي هعمل ال apply بالشكل ده :

```
apply(from = "${rootProject.projectDir}/gradle/flavors.gradle")
```

Gradle Tasks

■ إيه هي ال Gradle Tasks ؟

- ✓ عبارة عن small unit of work بنكتبة في ال gradle build script كل task بيكون موجود في ال gradle بيعرض action معين عايزة يتنفذ زي مثلاً لو عايز أطلع apk من ال project او أعمل install لل app على ال emulator او ممكن أفحص الكود بتاع ال app وأشوف المشاكل الموجودة فيه وكمان ايه الإقتراحات الموجودة علشان أحلها و tasks تانية كتيرة وكمان ممكن أخلى task يكون معتمد على task تاني زي ما هنشوف كمان شوية.
- ✓ ال tasks دي انا مش مضطر اني أفتح ال project على الأندرويد استديو علشان أشغلها بالعكس انا ممكن أشغله على ال terminal زي ال bash او اي cmd

فيه نوعين من ال Gradle tasks

Custom task

Built-in tasks

أولاً : built –in tasks

- ✓ دي عبارة عن tasks جاهزة في الأندرويد ممكن أستخدمها بشكل مباشر
- ✓ لو عايز أشوف كل ال tasks اللى موجودة هستخدم ال command ده

```
C:\Users\EEdbeshir\AndroidStudioProjects\MyApplication>gradlew tasks
```

هلاقيه ظهر كل ال tasks الموجودة وهلاحظ إنهم متقسمين لكذا نوع كل نوع منهم ليه ال tasks الخاصة بيه بالشكل ده :

1-Build tasks

- دى ال tasks المسؤله عن ال compiling وال packaging

```
Build tasks
-----
assemble - Assemble main outputs for all the variants.
assembleAndroidTest - Assembles all the Test applications.
assembleDebug - Assembles main outputs for all Debug variants.
assembleFree - Assembles main outputs for all Free variants.
assemblePro - Assembles main outputs for all Pro variants.
assembleRelease - Assembles main outputs for all Release variants.
assembleUnitTest - Assembles all the unit test applications.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildKotlinToolingMetadata - Build metadata json file containing information about the used Kotlin tooling
buildNeeded - Assembles and tests this project and all projects it depends on.
bundle - Assemble bundles for all the variants.
bundleDebug - Assembles bundles for all Debug variants.
bundleFree - Assembles bundles for all Free variants.
bundlePro - Assembles bundles for all Pro variants.
bundleRelease - Assembles bundles for all Release variants.
clean - Deletes the build directory.
compileFreeDebugAndroidTestSources
compileFreeDebugSources
compileFreeDebugUnitTestSources
compileFreeReleaseSources
compileFreeReleaseUnitTestSources
compileProDebugAndroidTestSources
compileProDebugSources
compileProDebugUnitTestSources
compileProReleaseSources
compileProReleaseUnitTestSources
```

- يعنى مثلاً ال assemble ده وظيفة يعمل build لل apk وكمان لو كنت مستخدم flavors هلاقية عمل generate لكل ال apks المتاحة سواء كان free او pro مثلاً او اياً كان ال flavors الموجودة

2-Build Setup tasks

- ده عادة بيشير إلى ال tasks اللى وظيفتها تجهز ال build environment أو ال configurations قبل عملية ال build لل project نفسة و ده ممكن يشمل ال dependencies او ال configurations وغيرهم

```
Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.
```

- ممكن أعمل project من جديد باستخدام ال command ده init

3- Verification tasks

- دى ال tasks المسؤله عن ال test سواء كان unit test او ال instrumented test

Verification tasks

```
-----
check - Runs all checks.
checkJetifier - Checks whether Jetifier is needed for the current project
connectedAndroidTest - Installs and runs instrumentation tests for all flavors on connected devices.
connectedCheck - Runs all device checks on currently connected devices.
connectedFreeDebugAndroidTest - Installs and runs the tests for freeDebug on connected devices.
connectedProDebugAndroidTest - Installs and runs the tests for proDebug on connected devices.
deviceAndroidTest - Installs and runs instrumentation tests using all Device Providers.
deviceCheck - Runs all device checks using Device Providers and Test Servers.
lint - Runs lint on the default variant.
lintFix - Runs lint on the default variant and applies any safe suggestions to the source code.
lintFreeDebug - Print text output from the corresponding lint report task
lintFreeRelease - Print text output from the corresponding lint report task
lintProDebug - Print text output from the corresponding lint report task
lintProRelease - Print text output from the corresponding lint report task
lintVitalFreeRelease - Print text output from the corresponding lint report task
lintVitalProRelease - Print text output from the corresponding lint report task
test - Run unit tests for all variants.
testFreeDebugUnitTest - Run unit tests for the freeDebug build.
testFreeReleaseUnitTest - Run unit tests for the freeRelease build.
testProDebugUnitTest - Run unit tests for the proDebug build.
testProReleaseUnitTest - Run unit tests for the proRelease build.
updateLintBaseline - Updates the lint baseline using the default variant.
```

- فى ال task ده lint مسؤل إنه يطلع كل المشاكل الموجودة فى ال project ولكن على شكل ملفات html لو نفذت ال command ده وبعد كدة فتحت ملف ال build عند ال reports هلاقى ملف ال html لو فتحتة هلاقى فيه كل المشاكل الموجودة فى ال project وشرحها

4- Install tasks

- دى tasks مسؤله عن ال deploying وال installing لل apk

Install tasks

```
-----
installFreeDebug - Installs the Debug build for flavor Free.
installFreeDebugAndroidTest - Installs the android (on device) tests for the FreeDebug build.
installProDebug - Installs the Debug build for flavor Pro.
installProDebugAndroidTest - Installs the android (on device) tests for the ProDebug build.
uninstallAll - Uninstall all applications.
uninstallFreeDebug - Uninstalls the Debug build for flavor Free.
uninstallFreeDebugAndroidTest - Uninstalls the android (on device) tests for the FreeDebug build.
uninstallFreeRelease - Uninstalls the Release build for flavor Free.
uninstallProDebug - Uninstalls the Debug build for flavor Pro.
uninstallProDebugAndroidTest - Uninstalls the android (on device) tests for the ProDebug build.
uninstallProRelease - Uninstalls the Release build for flavor Pro.
```

5-Android tasks

- ال tasks دي خاصة بال android بشكل عام زى مثلاً ال task الأول بيعرض كل ال dependencies الموجودة في ال project وال task الثاني مسؤل عن ال signing configurations وال task الثالث بيعرض كل ال source set الموجودة في ال project

Android tasks

```
-----  
androidDependencies - Displays the Android dependencies of the project.  
signingReport - Displays the signing info for the base and test modules  
sourceSets - Prints out all the source sets defined in this project.
```

6-Help tasks

- دي tasks بتظهر information عن ال project

Help tasks

```
-----  
buildEnvironment - Displays all buildscript dependencies declared in root project 'My Application'.  
dependencies - Displays all dependencies declared in root project 'My Application'.  
dependencyInsight - Displays the insight into a specific dependency in root project 'My Application'.  
help - Displays a help message.  
javaToolchains - Displays the detected java toolchains.  
kotlinDslAccessorsReport - Prints the Kotlin code for accessing the currently available project extensions and conventions.  
outgoingVariants - Displays the outgoing variants of root project 'My Application'.  
projects - Displays the sub-projects of root project 'My Application'.  
properties - Displays the properties of root project 'My Application'.  
resolvableConfigurations - Displays the configurations that can be resolved in root project 'My Application'.  
tasks - Displays the tasks runnable from root project 'My Application' (some of the displayed tasks may belong to subprojects).
```


ثانياً : custom tasks

- دلوقتي إحنا ممكن نعمل task معين خاص بينا زى ما هنشوف دلوقتي ولكن لان الموضوع مهم فهنبداً بحاجات صغيرة الأول وبعد كدة هنتكلم عن ال tasks اللى ممكن نحتاجها واحنا شغالين وبتوفر علينا شغل كتير
- ال tasks دى ممكن نكتبها فى ملفات ال build.gradle الخاصة بال project وممكن نعملها gradle script خاص بيها وبعد كدة نعمل apply فى ال build.gradle علشان يتنفذ

```
92 ▶ tasks.register( name: "task1"){  
93     println("hello world")  
94 }
```

- ده task صغير كل وظيفته يطبع كلمة hello world دلوقتي عايزين نفهم كل حاجة فى الكود ده
- ال function اللى هى register وظيفتها إنها بتحدد إن فيه task جديد هيتعمل وبتأخد two parameters الأول هو إسم ال task وهنا حددتة إنه هيكون task1 والثانى هو ال task نفسه يعنى ال action اللى هيتنفذ اللى هو
println("hello world")
- لو عملت run هلاقى ال output بالشكل ده :

```
hello world  
> Task :app:task1 UP-TO-DATE  
  
BUILD SUCCESSFUL in 4s  
  
Build Analyzer results available  
8:58:40 PM: Execution finished 'task1'.
```


■ ازای عمل run لا custom task ؟

- ال custom task ممكن أعمله run عن طريق ال terminal زى ما كنت بعمل run لا built-in tasks

```
c:\Users\EEdeshir\AndroidStudioProjects\MyApplication>gradlew task1
hello world

BUILD SUCCESSFUL in 3s
```

- وممكن أعمله run عن طريق ال android studio



```
92  ▶ tasks.register( name: "task1"){
93      println("hello world")
94  }
```

Configuration phase Vs Execution phase

أولاً : Configuration phase

- ده بيحصل لما ال gradle بيعمل build لا project أثناء ال build بيحصل Registration لكل ال tasks فى وقت ال Registration ده الكود اللى فى ال task بيتنفذ

ثانياً : Execution phase

- ده بيحصل لما بعمل run ل task معين الكود اللى بيتنفذ فى ال phase دى هو اللى بيقون داخل ال doFirst{ } و doLast{ }

علشان نفهم الكلام ده هناخد مثال صغير

```
77 ▶ tasks.register( name: "task1") {  
78     println("this is task one")  
79     doFirst {  
80         println("this is task one inside first")  
81     }  
82     doLast {  
83         println("this is task one inside last")  
84     }  
85 }
```

- لو عملت run هلاقى ال output ظهر بالشكل ده :

```
this is task one
```

```
> Task :app:task1  
this is task one inside first  
this is task one inside last
```

```
BUILD SUCCESSFUL in 2s
```

```
1 actionable task: 1 executed
```

```
Build Analyzer results available
```

```
9:42:34 PM: Execution finished 'task1'.
```

■ هلاحظ إن الكود إتنفذ كالتالى :

- أولاً : **this is task one**

ودة يعتبر ال main task ودة بيتنفذ لما ال task يحصله registration
يعنى ده معناه إنه run كجزء من ال task configuration phase مش فى
ال execution time

- ثانياً : **this is task one inside first**

وده لاننا إستخدمنا { doFirst{ } الكود الى مكتوب داخل { doFirst{ } بيتنفذ
بعد ال task الأساسى وقبل أى task تانى وده إتنفذ فى وقت ال execution
مش فى ال configuration

- ثالثاً : **this is task one inside last**

ده هيتنفذ فى الآخر لاننا إستخدمنا { doLast{ } وكمان ده بيتنفذ بعد ال
main task وبعد أى task تانى

dependsOn{ }

- الميثود دى مهمة فى ال task ومستخدمه بشكل كبير وعن طريقها
بحدد إن فيه task معين هيتنفذ قبل task تانى

```
77 ▶ tasks.register( name: "task1"){  
78     println("this is task one")  
79 }  
80 ▶ tasks.register( name: "task2") {  
81     println("this is task two")  
82     dependsOn(tasks.named( name: "task1"))  
83 }
```

فى الكود ده هنا أنا حددت إنه عشان ينفذ task2 لازم الأول هيروح ينفذ task1
وبعد كده ينفذ task2

- لو عملت run ل task2 هلاقى ال output ظهر بالشكل ده :

```
this is task two
this is task one
> Task :app:task1 UP-TO-DATE
> Task :app:task2 UP-TO-DATE

BUILD SUCCESSFUL in 1s

Build Analyzer results available
5:41:48 AM: Execution finished 'task2'.
```

This is task two
This is task one

- المفروض إن task1 يتنفذ الأول
وبعد كدة يتنفذ task2 ولكن اللى
ظهر عكس كده :

- اللى حصل هو انه فى البداية عمل register ل task2 وليس execute فننفذ
السطر ده : `println("this is task two")`

- بعد كدة عمل register ل task1 وليس execute فننفذ السطر ده برضو :
`println("this is task one")`

- لما جه ينفذ الميثود `dependOn{}` راح عمل execute ل task1 وبعد كده عمل
execute ل task2 فكداه فعلياً هو نفذ task1 الاول وبعد كدة نفذ task2
- ال message بتاعت up-to-date دى معناها إن ال gradle حسب task1 و
task2 وإن ال outputs بتاع الإثنين متغيرش

- لو حصل تغير مثلاً فى task1 وعملت run ل task2 بما إن task2 بيعتمد على
task1 ده معناه إن التغير ده ه reflect على task2 لانه هينفذ task1 الأول
- ال task التانى اللى معتمد عليه ال task الاول مش شرط يكون custom ممكن
عادى يكون built-in task

- دلوقتي عايز اعمل task وهو عبارة انه لما اعمل generate لـ apks يحطهم في folder معين في الـ app اسمة مثلاً my_apks

```
77 > tasks.register<Copy>( name: "moveApk") {  
78     from( ...sourcePaths: "${layout.buildDirectory.get()}/outputs/apk")  
79     into( destDir: "my_apks")  
80 }  
81 > tasks.named( name: "assemble") {  
82     finalizedBy( ...paths: "moveApk")  
83 }
```

- هنا انا حددت ان فيه task معين من نوع Copy اسمة moveApk النوع ده فيه الميثود دى :

files -- From() <-- ده بحدد فيه الـ path الى هنقل منه الـ

files -- Into() <-- بحدد الـ path الى هنقل ليه الـ

- لو عملت run للـ assemble task هينفذ في البداية الـ assemble وبعد كده هينفذ الـ moveApk task ودى فائدة الميثود { } finalizedBy اني بقدر احدد ان فيه task معين هيتنفذ في الآخر بعد task تاني وهلاقي الـ apk اتنقلت في folder جديد اسمة my_apks وفيه الـ apks الخاصة بالـ project