

# Apunts

[Descarregar aquests apunts](#)

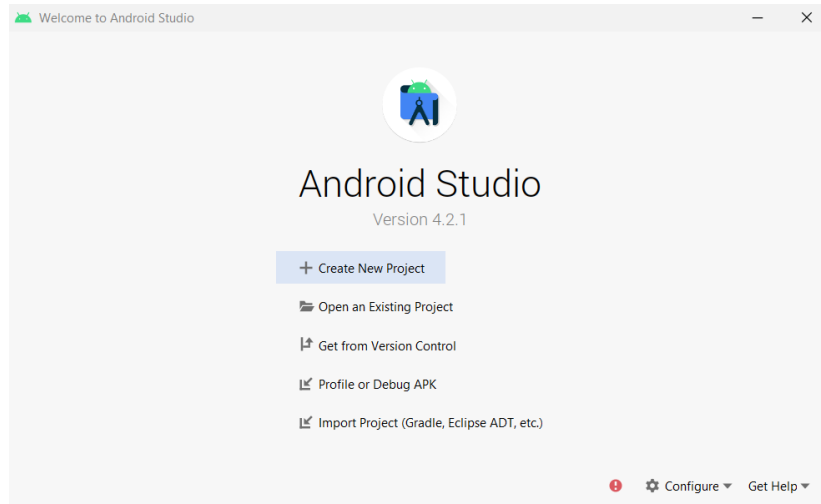
## Tema 2 . Estructura d'un projecte Android Studio

### Índex

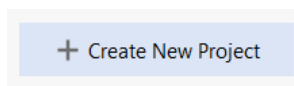
1. [Creant el primer projecte d'Android](#)
2. [Estructura d'un projecte en Android Studio](#)
3. [Recursos en Android](#)
  1. [La carpeta Layout](#)
  2. [Les carpetes Mipmap i Drawable](#)
  3. [La carpeta values](#)
  4. [Accedir als Recursos en Android](#)
4. [Android Manifest](#)
5. [Gradle](#)
  1. [Projectes, tasques i accions](#)
  2. [Arxius de configuració de Gradle](#)

## Creant el primer projecte d'Android

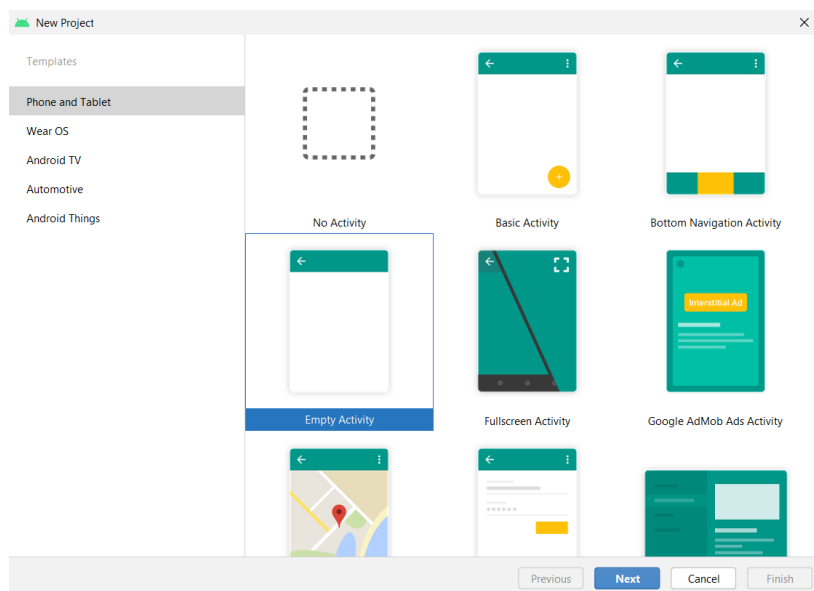
Realitzarem el nostre primer exemple en Android, ens servirà bàsicament per a conèixer l'entorn Android Studio i les carpetes de les quals consta un projecte Android creades amb aquest entorn. Una vegada instal·lada l'aplicació i si no s'ha creat un projecte amb anterioritat, en obrir l'entorn ens trobarem amb una finestra com la que segueix:



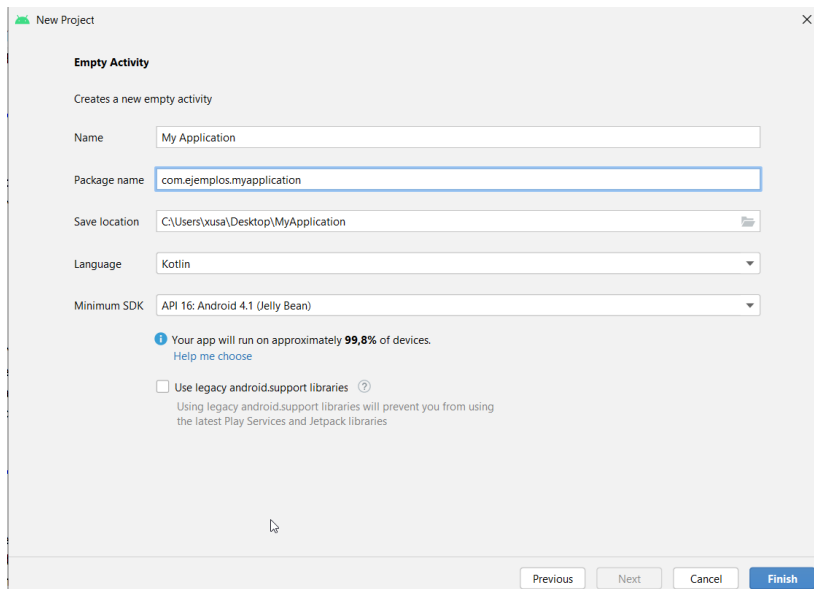
Com podem veure, aquesta finestra ens permet crear un projecte nou, obrir un existent o realitzar altres opcions, premerem en **Create a New Project**.



Haurem de seleccionar el tipus d'activitat que es crearà com a inici del nostre projecte, l'opció que necessitem al llarg del curs és una **Empty Activity**, que ens crearà una activitat buida.

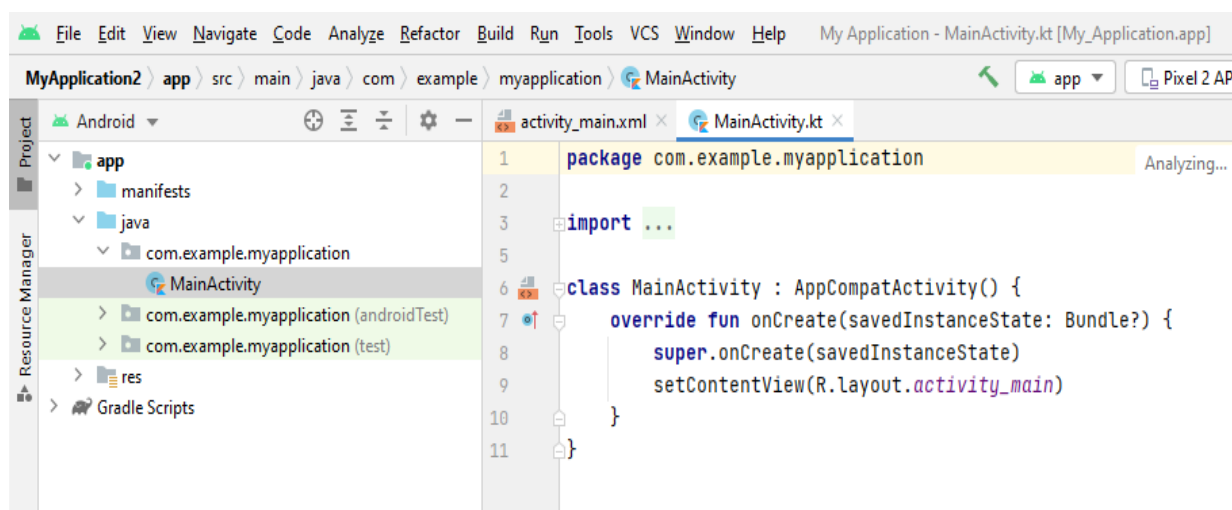


A partir d'ací haurem de seleccionar el nom de paquet que donarem a la nostra aplicació, la localització i el nom de l'aplicació, a més també haurem de seleccionar el Llenguatge amb el qual volem treballar (Java o Kotlin). És millor no editar el nom de paquet, es generarà només en modificar el nom de domini i el nom d'aplicació. En **Save location** podem seleccionar on volem deixar la nostra aplicació.



L'última opció permetrà seleccionar el **SDK mínim** amb el qual vulgueu que còrriga la vostra aplicació. El màxim no fa falta seleccionar-ho perquè el projecte agafarà el SDK més actualitzat que tingues instal·lat. Al costat del selector del sdk, podeu veure una explicació i un enllaç en el qual es poden veure les estadístiques actualitzades del percentatge de mòbils que podran córrer la vostra aplicació, segons el mínim SDK seleccionat.

A partir d'ací haurem d'esperar una mica fins que es genere el nostre projecte i aparega una finestra semblant a la següent:



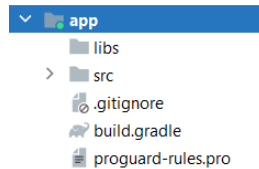
## Estructura d'un projecte en Android Studio

Una vegada arribats ací, s'haurà obert l'arbre de carpetes de l'aplicació a l'esquerra (encara que aquesta finestra és movable. Per si us resulta més útil en un altre lloc de la finestra d'Android Studio, sapieu que podeu moure-la prement i arrossegant la pestanya amb nom **1: Project** a qualsevol part. També podeu moure la resta de les finestres que estan obertes.

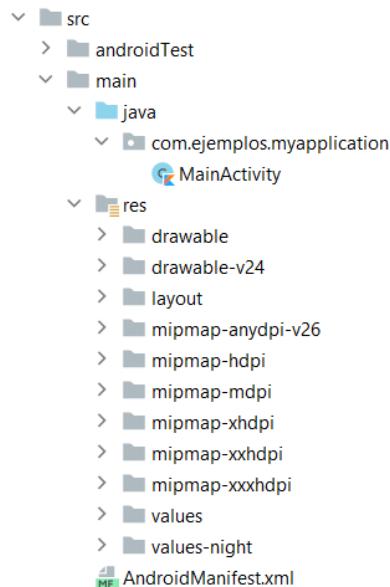
En l'arbre de directoris teniu diverses maneres de llistar les carpetes i arxius de l'aplicació. Les dos principals són **Android, Project, i Package**, però hi ha unes altres que podeu usar, per exemple: **Problems** mostrarà només els arxius que tenen errors.

La **carpeta app** és la que conté tot el relacionat amb el teu projecte, és la carpeta que ens interessa de moment i on inclourem els arxius necessaris perquè la nostra aplicació siga empaquetada. Si

desplegues el seu contingut voras tres carpetes: build, libs i src. Ara com ara ignorarem els altres arxius.

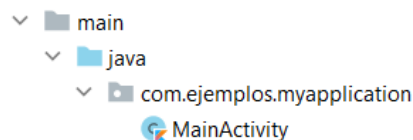


Normalment la major part del temps i forces les usarem en la carpeta **src**(Source). Dins d'ella se situa la carpeta main, la qual conté tots els arxius font Java i Kotlin per a la nostra aplicació. La carpeta **res** (Resources) que conté els recursos del projecte (icones, so, dissenys, etc.) i l'arxiu **AndroidManifest.xml** , més endavant aprofundirem en ell.

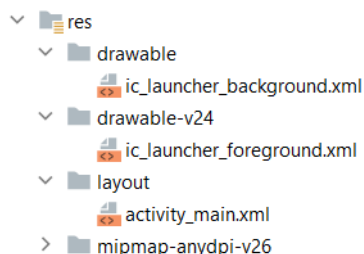


La Activity (o Activitys) que hem creat es troben en

**Nom de la vostra App->src->main->java->Paquet de la vostra App .**



Alguns dels altres arxius imprescindibles de tota aplicació Android, són **els recursos** que es troben en la carpeta **Nom de la vostra App->src->main->res** . Una vegada dins podeu veure, entre altres, les carpetes **layout** , **drawable** i **mipmap**



## Recursos en Android

Els recursos són arxivaments o dades externes que suporten el comportament de la nostra aplicació Android per exemple imatges, strings, colors, estils, etc.

Formalment en un projecte, aquests elements es troben en la carpeta **res** nomenat anteriorment. Allí

es troben subdirectoris que agrupen els diferents tipus de recursos.

La idea de l'ús de recursos és dividir el codi de la teua app per a mantindre independència. Tot amb la finalitat d'agregar variacions dels arxius per a adaptar l'aplicació a diferents tipus de pantalles, idiomes, versions, dimensions, configuracions d'orientació, etc. Per exemple: *no és el mateix l'espai d'un telèfon mòbil a una tauleta (tablet). No obstant això es poden crear variacions dels recursos perquè l'aplicació s'adapte a la densitat de pantalla de cadascun.*

Els grups de recursos es divideixen en subdirectoris. Cadascun d'ells conté arxius que compleixen una funció específica dins de l'aplicació. Has de respectar aquesta estructura de documents per a no tindre problemes en l'execució. En la següent [Taula de Recursos](#), podem veure un resum dels més usats.

Nom del subdirectori	Contingut
drawable	Recursos gràfics que puguin ser projectats en pantalla. Generalment trobaràs arxius d'imatge com .png, .jpg o .gif, no obstant això és possible usar altres com: arxius nine-patch, llistes d'estat, drawables amb múltiples nivells, drawables amb figures 2D definides en XML, i moltes més.
mipmap	Conté la o les icones de l'aplicació per a evitar distorsió entre diverses densitats de pantalla.
layout	Arxius XML que contenen definicions de la interfície d'usuari.
menu	Arxius XML que estableixen les característiques per als menús usats en la interfície. Normalment contenen definicions sobre els ítems albergats en un menú i les agrupacions entre ells.
values	Arxius XML que contenen dades simples com a sencers, strings, booleans, colors.

## Recursos Alternatius En Android

Un recurs alternatiu és una variació d'un recurs, que s'ajusta a una característica de configuració en el dispositiu mòbil on s'executa l'aplicació. Això s'aconsegueix a través d'una [taula](#) de qualificadors creada per Google que estandarditza totes les configuracions possibles que es puguin presentar. Un qualificador és un mecanisme gramatical que especifica el propòsit d'un recurs. La seua sintaxi és la següent: `<nom_recurs>-<qualificador>`

Per exemple: **mipmap-hdpi** contindrà la variació del recurs `ic_launcher.png` per a dispositius amb densitat de pantalla de 240dpi. Aquesta condició és especificada pel qualificador **hdpi**.

👉 tots els arxius que corresponguen al mateix recursos hauran de tindre el mateix nom, s'identificaran gràcies al qualificador afegit a l'identificador de carpeta.

Si cal indicar més d'un qualificador, se separen per guions i l'ordre de preferència és el que se segueix en la taula, el primer qualificador serà el de més preferència. Per exemple: *dispositius en anglés dels Estats Units i en orientació horitzontal*, tindran un nom de recurs alternatiu en el **drawable**, de la següent manera **drawable-en-rUS-land**.

👉 Una vegada que guardes els recursos alternatius en directoris denominats amb aquests qualificadors, Android aplicarà automàticament els recursos en la teua aplicació d'acord amb la

## La carpeta Layout

En la carpeta layout trobaràs els arxius que contenen les vistes de les activitats o fragments. En crear el projecte conté l'arxiu `activity_main.xml`. Aquest arxiu representa el disseny de la interfície de la meua activitat principal.

Construir la interfície a través de nodes XML és molt més senzill que la creació a través de codi Java o Kotlin. Addicionalment Android Studio ens ha dotat d'un panell de disseny estil **Drag and Drop**, la qual cosa és una benedicció per als desenvolupadors, ja que facilita la creació de la interfície d'usuari.

Aquest arxiu de disseny comença amb un node arrel anomenat `<ConstraintLayout>`. Un Layout és el contenidor principal que defineix l'ordre i seqüència en què s'organitzaran els widgets en la nostra activitat. Existeixen diversos tipus de Layouts, com per exemple el

`LinearLayout`, `GridLayout`, `FrameLayout`, `ConstraintLayout`, etc.

Android Studio crea per defecte un `ConstraintLayout` perquè permet crear un grup de components amb ubicacions relatives i a més permet tot el control a través de l'Editor de Disseny. Si obris l'arxiu `activity_main.xml` de la teua aplicació veuràs alguna cosa com això:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

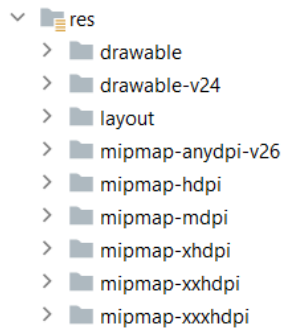
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Més endavant aprofundirem sobre els elements de disseny tipus Layouts.

## Les carpetes Mipmap i Drawable

Com ja has vist, hi ha una sèrie de carpetes que comencen pel qualificador mipmap i drawable. Aquestes carpetes es relacionen directament amb la mena de densitat de pantalla on s'executarà la nostra aplicació.



La majoria de dispositius mòbils actuals tenen un d'aquests 4 tipus de densitats:

- **Mediun Dots per Inch(mdpi)**: Aquest tipus de pantalles tenen una densitat de 160 punts per polzada.
- **High Dots per Inch(hdpi)**: En aquesta classificació trobarem telèfons la resolució dels quals és de 240 punts per polzada.
- **Extra high dots per inch(xhdpi)**: Resolucions majors a 340 punts per polzada
- **Extra Extra high dots per inch(xxhdpi)**: Rangs de resolucions majors a 480 punts per polzada.
- **Extra Extra Extra high dots per inch(xxxhdpi)**: Rangs de resolucions majors a 640 punts per polzada.

Mirem una xicoteta imatge il·lustrativa:



La imatge mostra alguns exemples de dispositius mòbils la densitat dels quals es troba dins dels rangs establits. Al final podem veure una categoria extra anomenada xxxhdpi. Aquesta denominació descriu a la densitat de televisors d'última generació que executen Android.

Android Studio 3 crea un **icona adaptativa** per a la seua aplicació que només està disponible en SDK 26 i posteriors. Aquestes icones adaptatives s'inclouen en la carpeta `mipmap-anydpi-v26`, són arxius .xml que s'adapten a la tecnologia del dispositiu. Per exemple: *una icona de selector adaptable es pot mostrar amb una forma circular en un dispositiu OEM i amb un quadrat amb cantonades arredonides en un altre.*

La carpeta **drawable** serveix per a col·locar totes les imatges que vaja a utilitzar la nostra aplicació, seguint el mateix esquema de sufixos que mipmap. Mentre que en la carpeta mipmap únicament col·locarem la icona de l'app. Com podem observar, quan creuem un projecte en Android Studio, per defecte, ja ens col·loca ací la icona amb el nom de `ic_launcher`.

## La carpeta values

Dins de la carpeta **values** es creen per defecte tres arxius **colors**, **strings**, **themes**, comentarem els dos primers i deixes el de themes per a més endavant, ja que necessita una explicació més detallada. Dins de l'arxiu de colors **colors.xml** els elements han de ser declarats amb l'etiqueta **<color>**. S'ha d'assignar un nom únic amb l'atribut name i el seu contingut és un color en forma hexadecimal que comença pel símbol numeral '#'. Es pot accedir a la paleta de colors en prémer sobre qualsevol dels quadres de color de l'esquerra.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <color name="purple_200">#FFBB86FC</color>
4     <color name="purple_500">#FF6200EE</color>
5     <color name="purple_700">#FF3700B3</color>
6     <color name="teal_200">#FF03DAC5</color>
7     <color name="teal_700">#FF018786</color>
8     <color name="black">#FF000000</color>
9     <color name="white">#FFFFFFFF</color>
10 </resources>
```

Un dels recursos més rellevants és l'arxivament **strings.xml**. Aquest fitxer emmagatzema totes les cadenes que es mostren en els widgets (controls, formes, botons, vistes, etc.) de les nostres activitats. Per a declarar el nostre recurs de strings usarem el node arrel **<resources>**. Per a declarar les cadenes usarem l'etiqueta **<string>** i establim l'atribut name com a identificador. Dins d'aquesta etiqueta posarem el text que **es visualitzarà en el component d'interfície**.

```
<resources>
    <string name="app_name">My Application</string>
</resources>
```

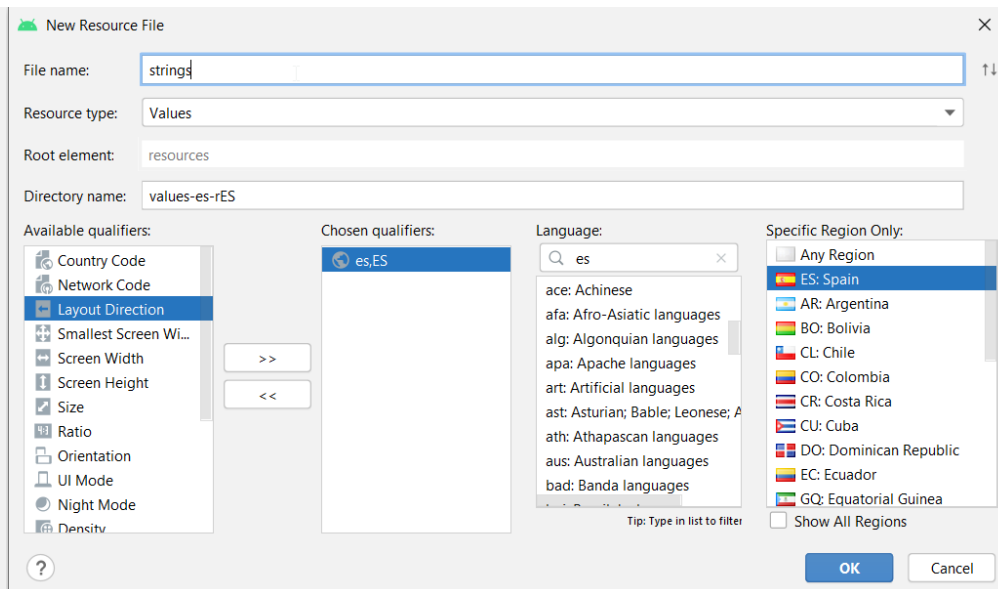
Per exemple, si tingueres un botó el títol del qual és "Pressiona ací", és recomanable incloure aquesta cadena en el teu arxiu strings.xml.

L'etiqueta app\_name conté el nom de l'aplicació. En aquest cas és "My Application".

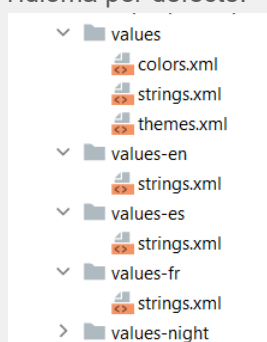
L'arxiu strings.xml és molt útil per als desenvolupadors. Una de les seues grans utilitats és facilitar l'ús de múltiples idiomes en l'aplicació. Això es deu al fet que pots externalitzar les cadenes del codi i el sistema s'encarregarà de seleccionar la versió de l'arxiu strings.xml amb el llenguatge necessitat, segons la configuració del dispositiu.

🎓 Definirem la nostra aplicació per a poder configurar-la en tres idiomes: espanyol, anglés i francès, per a una aplicació multi-idioma seria necessari incloure recursos de tipus cadena per a cada idioma que tinga l'aplicació, per a això crearíem tres versions del fitxer strings.xml i ho guardariem en els recursos addicionals de values segons l'idioma, per a saber les extensions dels **idiomes**, encara que Android Studio ens proporciona l'opció de crear recursos fàcilment. Per exemple, si dins de la carpeta res, selecciones **new -> Android Resources File** i seleccionem value, podrem crear els recursos addicionals per localitat:





En afegir els tres recursos addicionals, les carpetes quedarien de la següent manera. La carpeta values sense qualificador, seria per a l'idioma per defecte:



Si a cadascun d'ells li afegim el següent contingut:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
    <string name="hola_mundo">Hola Mundo!!</string>
</resources>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
    <string name="hola_mundo">Hello World!!</string>
</resources>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
    <string name="hola_mundo">Bonjour monde!!</string>
</resources>
```

Ens quedaria associar aquesta cadena al TextView que hi ha en el layout principal. Parell això obrim l'arxiu `activity_main` de la carpeta `layout` i en l'atribut text seleccionem el recurs string `hola_mundo`.

```

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hola_mundo"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

```

En executar l'aplicació i depenent de la configuració d'idioma del dispositiu, el text apareixerà en un idioma o un altre.

Personalment et recomane que uses aquest arxiu sempre, inclou tot el teu text i no deixes res per fora, la teua aplicació t'ho agrairà en el futur.

## Accedir als Recursos en Android

Ara que s'ha entès que són els recursos, entendrem com poden ser accedits en el codi Kotlin o en el codi XML. Per a obtenir la referència d'un recurs és necessari que se li assigne un identificador que el diferencie i li indique al programador on es troba.

Cada identificador se situa en una classe anomenada **R** a través d'una constant sencera. Aquesta classe és generada automàticament per una eina del SDK anomenada **appt**, per la qual cosa no és recomanable editar-ho manualment. Cada tipus de recurs existent dins de la carpeta **res** és una classe niada (*anidada*) estàtica de l'arxiu **R.class**. Per exemple, el subdirectori **drawable** pot ser accedit com a **R.drawable**.

### Accedir desde Kotlin

Ara, si desitges obtenir l'identificador d'un recurs des de codi Kotlin, llavors uses l'operador punt per a accedir als membres de cada classe interna **Paquet.R.tipusRecurs.nomRecurs**.

Per exemple, si tens una imatge l'identificador de la qual és **ic\_launcher\_background**. Per a accedir al seu contingut només navegues de la següent forma **R.drawable.ic\_launcher\_background**.

```

package com.ejemplos.myapplication

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        R.drawable.ic_launcher_background
    }
}

```

Si es vol accedir a la classe **Resources** que inclou tots els recursos de android, podem obtenir una instància **dins d'una activitat** amb el mètode **getResources()**. A través d'ella podem accedir als valors de qualsevol mena de recurs.

```

val color=resources.getColor(android.R.color.holo_blue_bright,theme)

```

En aquest cas hem accedit als recursos del sistema, per a això s'ha accedit primer a

```
android.R.tipusRecurs.nomRecurs
```

## Accedir des de XML

Aquesta forma de referència es dona quan dins d'un recurs definit en un arxivament XML necessitem usar el valor d'un altre recurs. Per a això usem la sintaxi **@[Paquet:]tipusRecurs/nomRecurs**

Per exemple, quan a un component `<TextView>` dins d'un layout li assignem el valor d'un string definit en l'arxiu `strings.xml`.

```
android:text="@string/hola_mundo"
```

Per als recursos del sistema des de xml seria `@android:tipusRecurs/nomRecurs`

## Android Manifest

El **Manifest** es pot dir que és l'arxiu més essencial de la nostra aplicació, si es produeix un error en ell el més probable és que l'App no es puga executar. És un arxiu XML que conté nodes descriptius sobre les característiques de l'App. Característiques com els building blocks existents, la versió de SDK usada, els permisos necessaris per a executar alguns serveis, activitats i seves de l'aplicació i moltes més.

En poques paraules l'Android Manifest mostra una descripció de tota la nostra aplicació. La seua configuració pot realitzar-se a través d'una interfície gràfica, però és recomanable conèixer la sintaxi ja que en moltes ocasions serà més fàcil i ràpid fer-ho des del propi xml.

Si obris l'arxiu en l'editor veuràs un codi similar a aquest:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ejemplos.myapplication">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

El nom ha de romandre intacte, ja que s'usa com a referència per al parsing de la nostra aplicació. El node arrel d'aquest document es representa amb l'etiqueta `<manifest>` i per obligació ha de contindre un fill de tipus `<application>`.

En el codi anterior podem observar que manifest posseeix dos atributs, **xmlns:android** i **package**. El primer no hem de canviar-ho mai, ja que és el namespace de l'arxiu.

L'atribut package indica el nom del paquet Java/Kotlin que suporta a la nostra aplicació. El nom del paquet ha de ser únic i un diferenciador a llarg termini.

L'etiqueta `<application>` representa com estarà construïda la nostra aplicació. Dins d'ella definirem

nodes referents a les activitats que conté, les llibreries incloses, els Intents, Providers, i altres components.

## Etiqueta application

Alguns atributs de l'etiqueta `<application>` són:

- **allowBackup:** Aquest atribut pot prendre els valors de true o false. Indica si l'aplicació serà persistent en tancar nostre AVD.
- **icon:** Indica on està situat la icona que s'usarà en l'aplicació. Hem d'indicar la ruta i el nom de l'arxiu que representa la icona. En aquest cas apuntem a les carpetes mipmap on es troba `ic_launcher.png`. Icona per defecte que ens proporciona Android Studio. Es mostrarà en l'Android Market i en el llançador de l'aplicació del nostre dispositiu. També serà la icona per defecte per a totes les activitats que definim dins de l'etiqueta.
- **label:** És el nom de l'aplicació que veurà l'usuari en el seu telèfon. Normalment apunta a la cadena "app\_name" que es troba en el recurs strings.xml. El contingut d'aquest atribut apareixerà en la barra de títol de l'activitat si la tinguera. També s'usarà en el llançador de l'aplicació si definim aquesta activitat com el punt d'entrada de la nostra aplicació. Si no ho definim, s'utilitzarà el label definit en l'etiqueta `<application>`. Serà el títol per defecte de les activitats que definim dins de l'etiqueta application.
- **theme:** Aquest atribut apunta a l'arxiu de recursos styles.xml, on es defineix la personalització de l'estil visual de la nostra aplicació.
- **supportsRtl:** Declara si l'aplicació està disposada a admetre dissenys de dreta a esquerra (RTL). Si s'estableix en true i la targetSdkVersion s'estableix en 17 o superior, en aqueix cas el sistema activarà i utilitzarà diverses API RTL perquè l'aplicació pugui mostrar dissenys RTL. En un altre cas s'ignorarà la configuració de regió de l'usuari.

Dins de `<application>` trobarem expressada l'activitat principal que definim en crear el projecte. Usarem `<activity>` per a representar un node tipus activitat.

## Etiqueta activity

Cada etiqueta `<activity>` representa una de les activitats de l'aplicació. Alguns atributs són:

- **label:** Fa referència al text que es mostrarà en la capçalera de l'activitat. Si no s'indica cap atribut label, la activity tindrà el mateix nom que l'Aplicació.  
Dins de `activity` ens podem trobar:
- **android:name:** Especifica el nom de l'activitat, en l'exemple s'ha posat `.MainActivity` ja que tenim el nom del package definit com a atribut de l'element manifest.
- **android:screenOrientation:** (portrait o landscape) Defineix l'orientació de l'aplicació vertical o apaïxada. Si no definim aquest atribut, s'usarà el que el dispositiu tinga predeterminat, inclòs el variable en funció de l'acceleròmetre. Quan la pantalla canvia d'orientació l'activitat es destrueix.
- **android:configChanges** En Android, el canvi d'orientació o l'aparició del teclat es considera un canvi de configuració, la qual cosa fa que la pantalla es redibuïxi per a adaptar-se als canvis. En aquest atribut definim els esdeveniments que volem manejar nosaltres, amb la finalitat d'evitar que es destrüísca l'activitat. Els diferents efectes se separen amb la barra vertical o pipe `|`, podríem posar `keyboard|keyboardHidden|orientation`, per a l'aparició desaparició de la pantalla i per al canvi d'orientació del dispositiu.

En Android no existeix un únic punt d'entrada per a la nostra aplicació. Podem iniciar-la a través de múltiples activitats o services que poden ser iniciats a partir de intents específics que pot enviar el sistema o una altra aplicació. Per a dir a Android davant quin intent ha de reaccionar la nostra aplicació i com, existeix el `<intent-filter>`.

Amb l'etiqueta `intent-filter` i el seu element `<action>` estem indicant que aquesta activity serà un punt d'entrada per a la nostra aplicació. Només pot haver-hi una activity que reaccione a aquest intent.

Amb element `<category>` li diem a Android que volem que aquesta activity siga afegida al llançador de l'aplicació. Poden haver-hi diverses activitats que reaccionen a aquest intent.

D'aquesta manera Android sap que en ser premuda (pulsada) la icona de l'aplicació, ha d'iniciar aquesta activity. Si no haguérem establert el `<intent-filter>`, aquesta activity només es podria cridar des de dins de l'aplicació. Fixem-nos que tots dos elements només tenen l'atribut `name` sent el seu valor el nom del intent.

```
<activity android:name=".MainActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

### Etiqueta uses-permissions

Android compta amb un model de seguretat molt elaborat. Cada aplicació s'executa en el seu propi procés i màquina virtual. Android restringeix l'ús dels recursos del sistema: targeta SD, WIFI, HW d'àudio, etc. Mitjançant aquest Tag especifiquem els permisos que necessitarà la nostra aplicació per a poder executar-se, a més són els que haurà d'acceptar l'usuari abans d'instal·lar-la. Per exemple, si es desitja utilitzar funcionalitats amb Internet o el vibrador del telèfon, cal indicar que la nostra aplicació requereix aqueixos permisos. Algun d'aquests permisos són:

- `android.permission.RECORD_AUDIO`, per a l'accés al hw d'àudio i enregistrament.
- `android.permission.INTERNET`, concedeix permís a totes les API's de xarxa.
- `android.permission.WRITE_EXTERNAL_STORAGE`, per a emmagatzematge extern.
- `android.permission.WAKE_LOCK`, ens permet antibloqueig (que si el terminal està inactiu aquest no es bloquege).

### Permisos en moment d'execució

A partir dels dispositius amb instal·lació Android 6.0 (nivell de API 23) o superior, és necessari demanar els [permisos en el moment d'execució](#) de l'aplicació. S'ha de tindre en compte que els permisos es demanaran just en el moment en el qual l'usuari comence a interactuar amb la funció que els requereix, si aquests són rebutjats per l'usuari és important que el flux de l'App pugui continuar funcionant.

Un model a seguir podria ser el següent:

1. Declara els permisos necessaris en el Manifest.
2. Esperar que l'usuari invoque l'acció que requereix els permisos.
3. Comprova que els permisos encara no han sigut atorgats. **Línia 5**
4. Mostra justificació (Dialogue) per la qual es requereixen els permisos, en cas que es veja necessari. **Línia 6 i 10**
5. Sol·licita els permisos. **Línia 14**

6. Verifica la resposta de l'usuari. En cas afirmatiu s'accedirà als recursos, proporcionant una eixida elegant en cas que l'usuari es negue a proporcionar-los.

El següent codi és la base funcional, a partir de la qual podem afegir el necessari per a les nostres diferents aplicacions. Fixa't que en l'exemple es demanen dos permisos.

```
val RESPUESTA_PERMISOS= 111
@RequiresApi(Build.VERSION_CODES.Q)
fun solicitarPermisos()
{
    5    if (checkSelfPermission(READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_DENIED
        || checkSelfPermission(READ_CONTACTS) == PackageManager.PERMISSION_DENIED) {
    7    if (shouldShowRequestPermissionRationale(READ_EXTERNAL_STORAGE)) {
        //Si se decide explicar los motivos de los
        //permisos con algo similar a un dialogo
    10   }
        if (shouldShowRequestPermissionRationale(READ_CONTACTS)) {
        //Si se decide explicar los motivos de los
        // permisos con algo similar a un dialogo
    14   }
        //Con requestPermissions se pide al usuario que
        //permita los permisos, en este caso dos
        requestPermissions(
            arrayOf(READ_EXTERNAL_STORAGE, READ_CONTACTS),
            RESPUESTA_PERMISOS)
    }
}
```

```
override fun onRequestPermissionsResult(
    requestCode: Int, //codigo de identificación del resultado
    permissions: Array<out String>, //array con los nombres de los permisos
    grantResults: IntArray //array de 0 y -1 (permitido, no permitido) en orden
)
{
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    when {
    9    RESPUESTA_PERMISOS == requestCode && grantResults.isNotEmpty() -> {
    10    for (i in 0..permissions.size - 1) {
    11    if (grantResults[i] == PackageManager.PERMISSION_GRANTED)
        Toast.makeText(
            applicationContext, "Permiso Concedido " +
            permissions[i], Toast.LENGTH_SHORT
        ).show()
    }
    }
    }
}
```

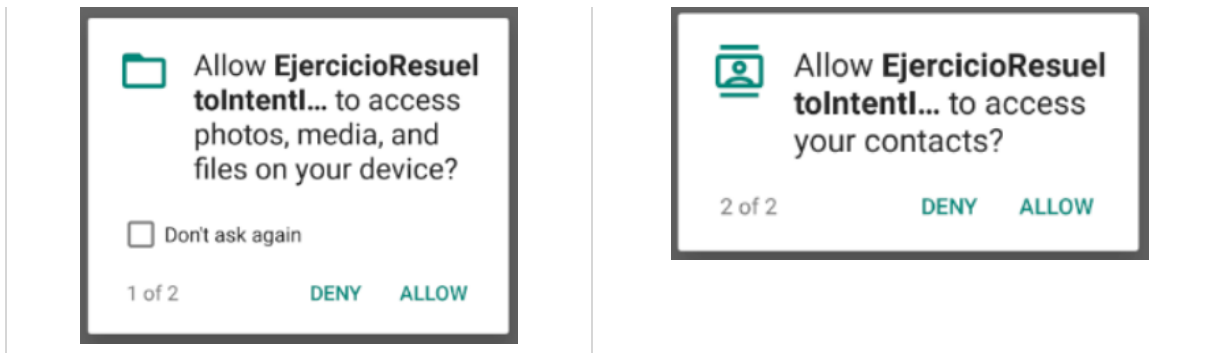
✎ Mètode que s'executa quan l'usuari respon als permisos. Al mètode li arriba el codi de la petició, el array amb els noms dels permisos, i el array d'enters indicant si el permís ha sigut denegat o concedit.

**Línia 9** es comprova si el codi d'entrada és l'establert i si hi ha resultats.

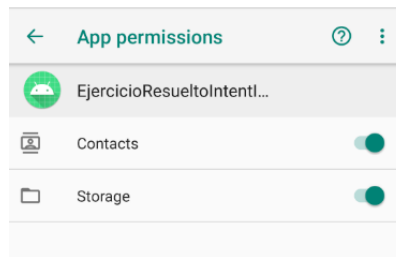
**Línia 10 i 11** es recorren tots els permisos i es mostra amb un toast els permisos concedits.

L'eixida per pantalla seria similar a la següent a l'hora de demanar permisos a l'usuari:

--	--



A més si entrem dins dels permisos de l'aplicació i l'usuari els ha concedits, podem veure'ls.



 **Prova el codi anterior en una aplicació i comprova el seu funcionament**


### Etiqueta uses-features


Permet especificar les característiques maquinari que requereix la nostra aplicació pantalla multitàctil, suport OpenGL, etc. per a poder instal·lar-se. Algunes restriccions maquinari per a la nostra aplicació podrien ser:

- `<uses-feature android:name="android.hardware.microphone" android:required="false" />`
- `<uses-feature android:name="android.hardware.camera" android:required="false" />`
- `<uses-feature android:name="android.hardware.camera.autofocus" android:required="false" />`
- `<uses-feature android:name="android.hardware.camera.flash" android:required="false" />`
- `<uses-feature android:name="android.hardware.touchscreen.multitouch" android:required="true" />`

Manifest complet d'exemple:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.xusa.miaplicacion"
    android:versionCode="11"
    android:versionName="1.5.1">
    <application android:icon="@drawable/icon">
        <activity
            android:name=".UnActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@style/Theme.NoBackground">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="OtroActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@style/Theme.NoBackground">
        </activity>
        <!-- Elementos para la publicidad -->
        <meta-data android:name="com.mobclix.APPLICATION_ID" android:value="insert-your-application-i
        <activity android:name="com.mobclix.android.sdk.MobclixBrowserActivity"/>
    </application>
    <supports-screens
        android:anyDensity="true"
        android:largeScreens="false"
        android:normalScreens="true"
        android:smallScreens="false">
    </supports-screens>
    <!-- Permisos que se exigen para mostrar la publicidad -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.GET_TASKS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <!-- Permisos en General -->
    <uses-permission android:name="android.permission.VIBRATE"/>
    <!-- Version mínima de android soportada por la aplicacion -->
    <uses-sdk android:minSdkVersion="3"/>
</manifest>
```

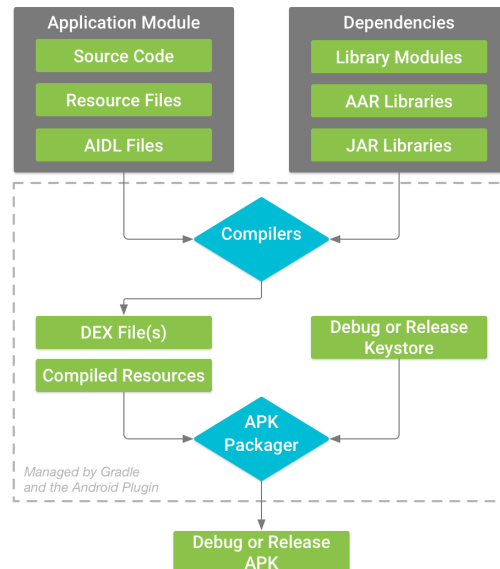
 **Analitza l'arxiu de manifest anterior, buscant en Internet la informació que no trobes en el tema. Després fes una descripció de com creus que serà l'aplicació que conté aqueix manifest.**

 **Modifica el Manifest de l'exercici HolaMundo de manera que com a requisits del mateix volem que s'execute en manera apaisada i que no afecte el canvi d'orientació del dispositiu. Canvia la icona i l'etiqueta de la activity principal (provar que si no la té, posa per defecte la de l'aplicació) i dona permisos d'internet. A més s'hauran d'afegir dos llenguatges més per al string Hola Món.**

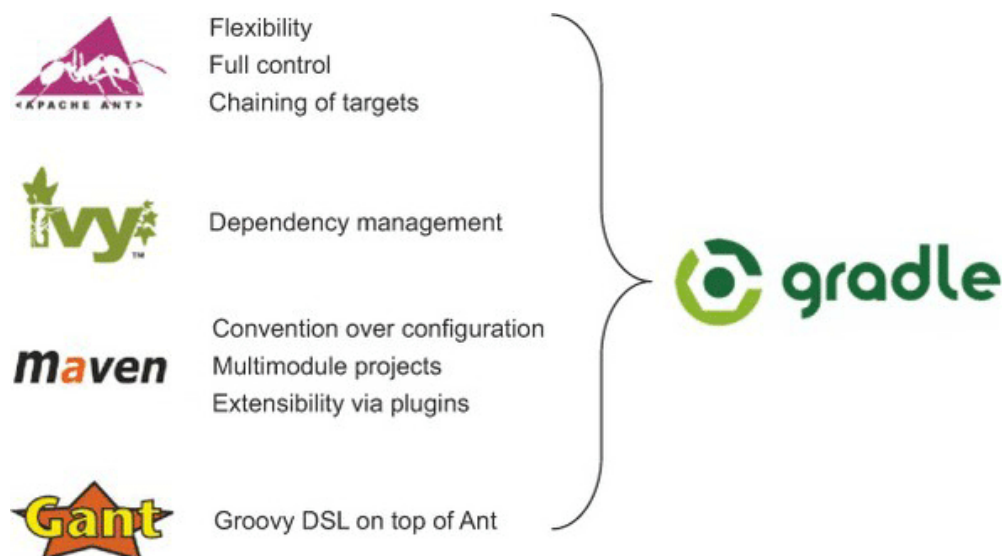
## Gradle



Una vegada creada i executada la nostra primera aplicació, haurem d'aprofundir una mica en el que ocorre en el **procés de compilació**, temps transcorregut des del moment que li donem l'ordre de compilació **play**, fins que el AVD mostra per pantalla l'eixida de la nostra app. Qui s'encarrega d'aqueix procés en Android Studio?, la resposta és **Gradle, un complement d'Android** que converteix el projecte en un paquet d'aplicacions per a Android (APK) o un **Android App Bundle (AAB)**. A l'ésser un complement necessita de les actualitzacions específiques de Gradle, independents de les d'Android Studio.



Gradle és una eina de **automatització de la construcció de codi** que beu de les aportacions que han realitzat eines com **Ant** i **Maven**, però intenta portar-ho tot un pas més allà.



Per a començar es recolza en [Groovy](#) i en un [DSL \(Domain Specific Language\)](#) per a treballar amb un llenguatge senzill i clar a l'hora de construir el build, comparat amb Maven que usa **XML**.

D'altra banda, disposa d'una gran flexibilitat que permet treballar amb altres llenguatges i no sols **Java**.

Un altre avantatge clau de **Gradle** sobre els sistemes de compilació anteriors de **Android Studio**, és l'administració de dependències. Abans, per a usar una biblioteca d'Android en un projecte, calia descarregar el codi font, agregar-lo al projecte i passar a la compilació. Gradle automatitza aquest procés, ja que amb agregar una sola línia a l'arxiu `build.gradle` del mòdul, s'encarregarà de descarregar la biblioteca compilada d'un repositori públic i l'agregar-la al projecte.

## Projectes, tasques i accions

Quan creguem un nou projecte en android, aquest haurà de tindre almenys un projecte de Gradle (per defecte es crearà només `build.gradle` de nivell superior). Un projecte ha de tindre almenys una tasca i una tasca consta d'algunes accions (podríem veure aquestes com una col·lecció de codi, semblança a una funció).

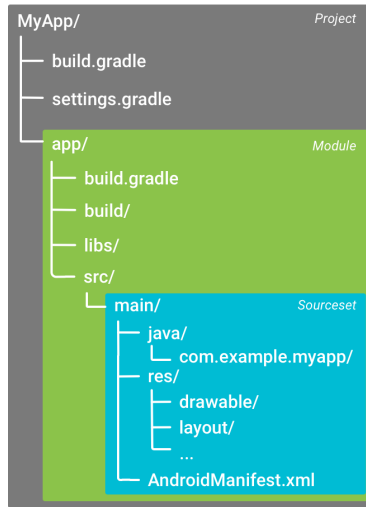
Una tasca indica una unitat d'execució lògica, Gradle té creades moltes tasques per defecte que són suficients per a treballar en els nostres projectes. Per exemple: quan es compila el projecte es llancen una sèrie de tasques de Gradle que automatitzen el treball, podem veure part d'aquestes en la següents línies.

```
> Task :app:preBuild UP-TO-DATE
> Task :app:preDebugBuild UP-TO-DATE
> Task :app:compileDebugAidl NO-SOURCE
> ...
> Task :app:compileDebugKotlin UP-TO-DATE
> Task :app:compileDebugJavaWithJavac UP-TO-DATE
> Task :app:compileDebugSources UP-TO-DATE
> Task :app:mergeDebugJavaResource UP-TO-DATE
> Task :app:dexBuilderDebug UP-TO-DATE
> Task :app:mergeDexDebug UP-TO-DATE
> Task :app:packageDebug
> Task :app:assembleDebug
```

Si es necessitara realitzar accions específiques, es podrien crear tasques propies utilitzant el DSL de Gradle.

## Arxius de configuració de Gradle

Cada vegada que es crea un projecte en **Android Studio**, el sistema genera automàticament tots els arxius de **configuració necessaris de Gradle**. Alguns es mostra en la següent imatge.



Els projectes d'Android Studio consten d'1 o més mòduls, que són components que pot compilar, provar i depurar de manera independent. Cada mòdul té el seu propi arxiu de compilació, per la qual cosa cada projecte d'Android Studio conté **2 tipus** d'arxius de compilació de Gradle.

- **Arxiu de compilació de nivell superior**, ací trobarà les opcions de configuració que són comunes a tots els mòduls que conformen un projecte.
- **Arxiu de compilació a nivell de mòdul**, cada mòdul té el seu propi arxiu de compilació Gradle que conté configuracions de compilació específiques del mòdul. Aquests són els que necessiten més temps d'edició per part del programador.

Per a accedir ràpidament a aquests **build.gradle arxius**, obrirem el panell **Projecte d'Android Studio** (seleccionant la pestanya Android), dins de la carpeta Gradle Scripts els trobarem. Els primers dos elements en la carpeta Gradle Scripts són els arxius de compilació de Gradle a nivell de projecte i a nivell de mòdul

## Arxiu de compilació Gradle de nivell superior

Cada projecte d'Android Studio conté un únic arxiu de compilació de Gradle de nivell superior. Aquest **build.gradle** arxiu és el primer element que apareix en la carpeta Gradle Scripts i està clarament marcat com a Projecte. La majoria de les vegades, no es necessitarà fer canvis en aquest arxiu, però continua sent útil per a comprendre el seu contingut i el paper que exerceix en el projecte.

```
// Top-level build file where you can add configuration options common to all sub-projects/modules
buildscript {
    ext.kotlin_version = "1.5.21"
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.2.2"
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        mavenCentral()
        jcenter() // Warning: this repository is going to shut down soon
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

## Arxius de compilació Gradle a nivell de mòdul

A més de l'arxiu de compilació de Gradle a nivell de projecte, cada mòdul té un arxiu de compilació **build.gradle** propi. A continuació es mostra una bàsica d'un arxiu de compilació de Gradle a nivell de mòdul.

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

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.ejemplos.b2.myapplication"
        minSdkVersion 19
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled 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'
    }
}

dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

## Altres arxius de Gradle

A més dels arxius `build.gradle`, la carpeta Gradle Scripts conté alguns altres arxius Gradle. La majoria de les vegades no haurà de ser editats manualment, ja que s'actualitzaran automàticament en fer canvis en el projecte.

- **gradle-wrapper.properties (Versió Gradle)**. Aquest arxiu verifica si està instal·lada la versió correcta de Gradle i descàrrega la versió necessària si no és així.
- **settings.gradle**. Aquest arxiu fa referència a tots els mòduls que componen el projecte.
- **gradle.properties (Propietats del projecte)**. Aquest arxiu conté informació de configuració per al projecte.
- **local.properties (ubicació del SDK)**. Aquest arxiu informa el complement Android Gradle on pot trobar la instal·lació d'Android SDK.