

UD 01. Programació d'aplicacions per a dispositius mòbils. Android.

Primeres passes amb Android Studio.



Continguts

1	Android Studio	3
1.1	Instal·lació	3
1.1.1	Creant una entrada al menú	9
2	Primers passos amb Android Studio: Hola món	10
2.1	La interfície d'Android Studio	13
2.2	La vista de fitxers del projecte	16
2.3	La vista Android	18
2.4	Execució de l'aplicació	19
2.4.1	Activant la virtualització a l'emulador	21
3	Fitxers d'Activities i Layouts	22
3.1	Enllaçant Layout i Activities	25
3.1.1	Exemple	26
3.1.2	Exercicis	28
3.2	Enllaçant Layouts i Activities amb View Binding	28
3.2.1	Habilitar el Data Binding	29
3.2.2	Ús del View Binding	29

1 Android Studio

Android Studio és un IDE basat en JetBrains IntelliJ Idea per al desenvolupament en Android.

Les principals característiques d'aquest IDE són:

- És distribuït sota llicència Apache 2.0 (lliure)
- Està desenvolupat en Java, C i C++
- Fa ús de Gradle per a la construcció de paquets
- Refactorització de codi
- Entorn WYSIWYG (What you see is what you get)
- Ofereix suport per a aplicacions Android Wear/Wear OS, per a dispositius corporals (wearables),
- Ofereix emuladors de dispositius virtuals per executar i depurar aplicacions

1.1 Instal·lació

El lloc web d'Android Studio (<https://developer.android.com/studio>) ens ofereix la possibilitat de descarregar i instal·lar l'IDE en qualsevol plataforma.

Tot i que existeixen altres formes alternatives d'instal·lació, com pota ser mitjançant Ubuntu Make o des de la tenda d'aplicacions del sistema, en forma de paquet Snap o Flatpak, el procediment oficial és relativament senzill, i és l'aconsellable.



El procés d'instal·lació i que seguirem, per als diferents sistemes operatius es troba descrit a la web de documentació:

- <https://developer.android.com/studio/install>

Anem a veure el procediment centrant-nos en sistemes GNU/Linux basades en Ubuntu, com pota ser Justix, però fent referència també a com ho faríeu en Windows.

El primer que cal fer és descarregar algunes llibreries de sistema necessàries amb l'ordre:

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1  
↳ libbz2-1.0:i386
```



Com veiem es tracta de llibreries del sistema de 32 bits. En cas que l'ordre anterior done error i no trobe els paquets, probablement és perquè no tenim habilitada aquesta arquitectura (recordeu que Ubuntu ja fa temps que treballa només amb arquitectures de 64 bits).

Per habilitar l'arquitectura de 32 bits al nostre sistema haurem de llançar l'ordre:
`dpkg --add-architecture i386.`

Una vegada instal·lades, descarreguem el paquet corresponent al nostre sistema des de la web de descàrregues (<https://developer.android.com/studio>), i el descomprimim en algun directori dins la nostra carpeta personal. Podeu fer-ho on vulguen. Per a la resta de document, anem a treballar a la carpeta: `~/ .local/share`.

Una vegada descomprimit, se'ns crea el directori `~/ .local/share/android-studio`. Accedim a ell, i llancem l'assistent d'instal·lació:

```
:~/ .local/share/android-studio/bin$ ./studio.sh
```

El primer que ens demana aquest instal·lador és si ja teníem alguna instal·lació prèvia i volem importar els ajustos. Indicarem que no:

Instal·lació d'Android Studio

Ara veurem la finestra d'Splash de l'aplicació:

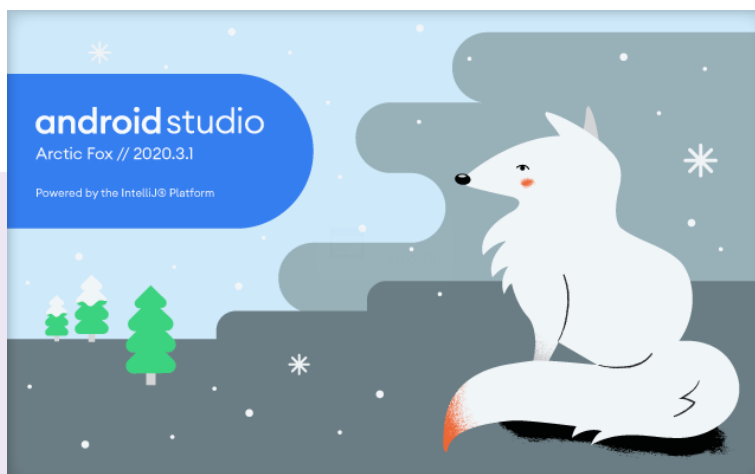


Figura 1: Instal·lació d'Android Studio

Després d'aquesta, se'ns sol·licita permís per recopilar informació per a Google sobre la instal·lació. Podeu indicar si voleu o no.

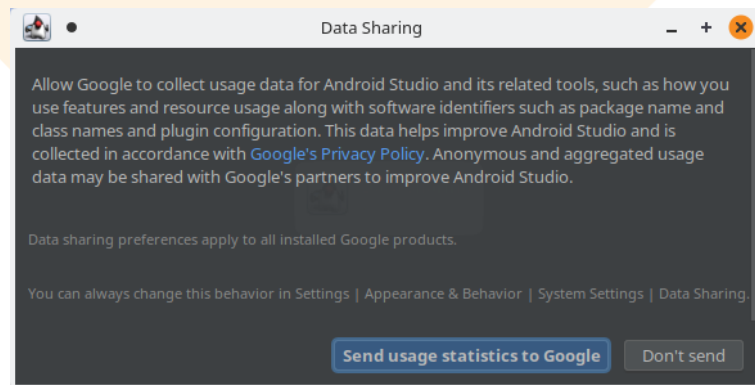


Figura 2: Instal·lació d'Android Studio

Ara se'ns mostra la primera pantalla en sí de l'assistent. Polsem en *Seguent*:

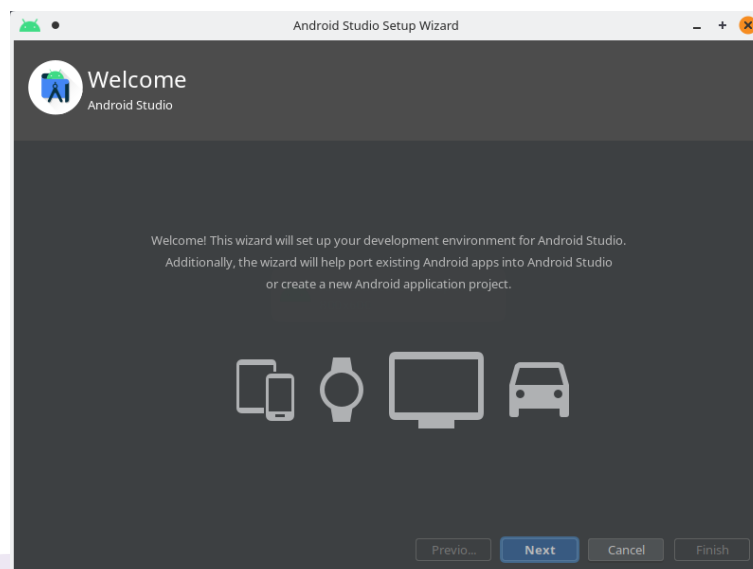


Figura 3: Instal·lació d'Android Studio

Establim el tipus d'instal·lació: Personalitzada estàndard. Seleccionem aquesta segona.

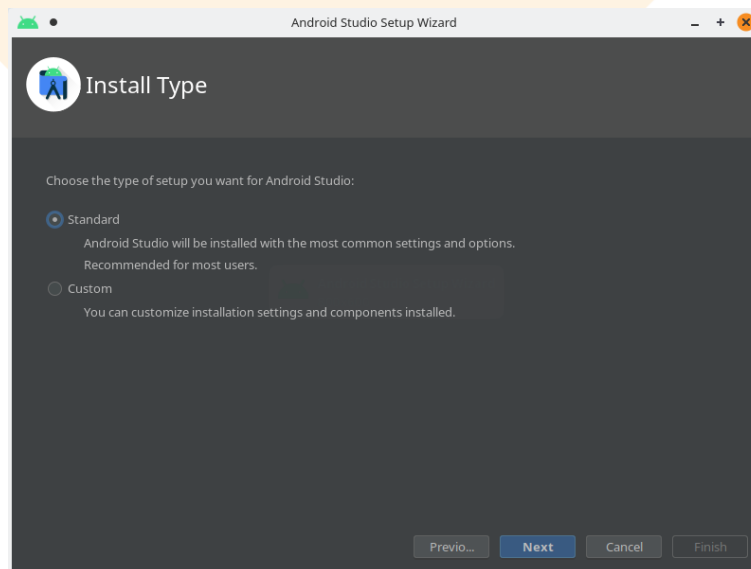


Figura 4: Instal·lació d'Android Studio

Ara el tema que més ens agrada, podent triar entre un tema clar i un fosc:

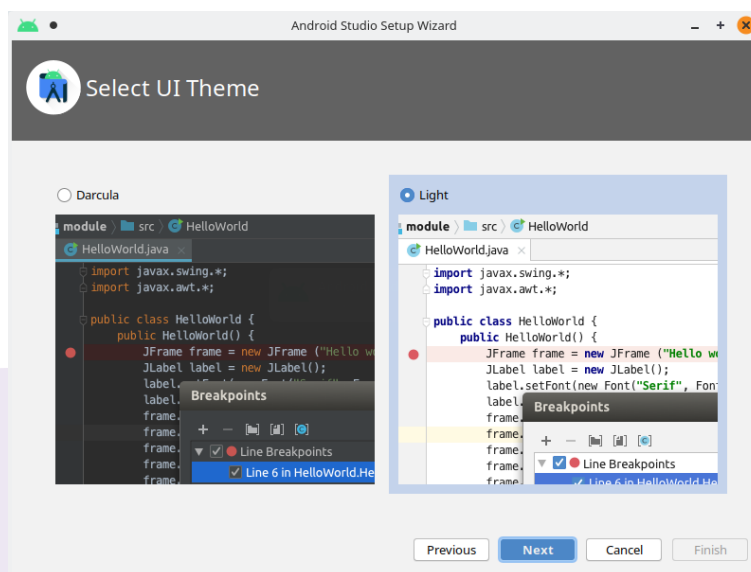
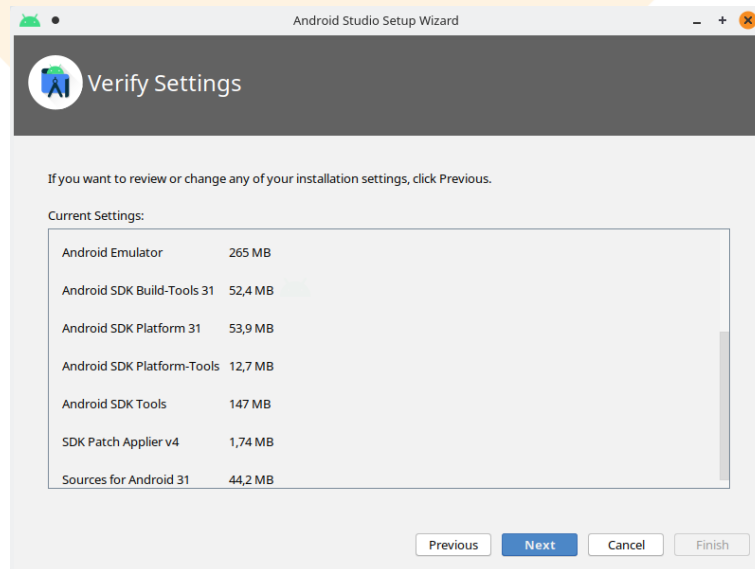
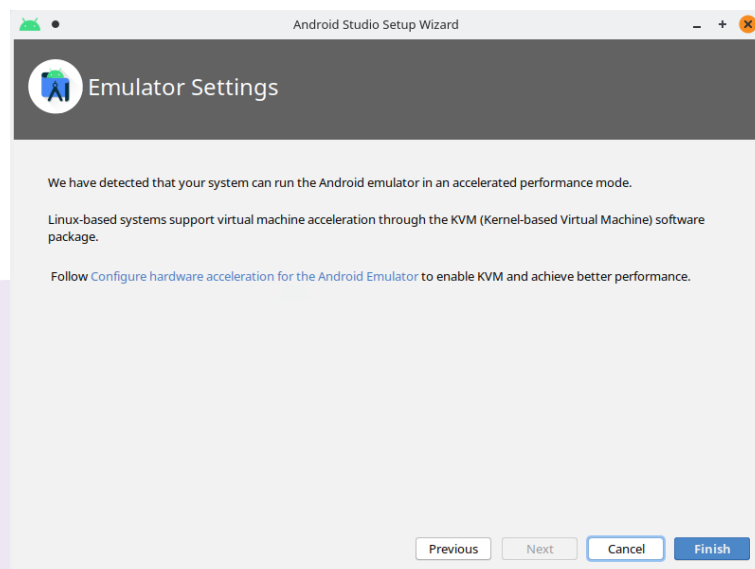


Figura 5: Instal·lació d'Android Studio

Ara ens mostrarà un resum de la instal·lació estàndard. Com veiem, instal·larà diferents components de l'SKD d'Android, així com l'emulador (Android Emulator). Com que anem a desenvolupar programari per a tipus de dispositiu diferent al que estem desenvolupant, necessitem bé un dispositiu extern, o bé, com és el cas, un emulador.

**Figura 6:** Instal·lació d'Android Studio

A la següent pantalla, l'assistent detecta que estem en un sistema GNU/Linux, i ens comenta que podem activar la virtualització basada en el kernel (KVM) per tal d'executar l'emulador d'Android en el mode de rendiment amb acceleració. De moment fem clic en següent:

**Figura 7:** Instal·lació d'Android Studio

Fet açò, ens descarregarà ja tots els components necessaris per fer la instal·lació:

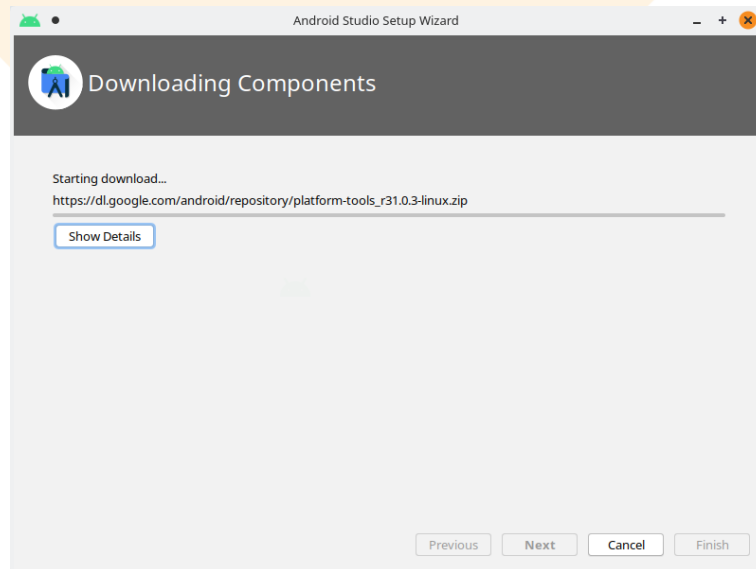


Figura 8: Instal·lació d'Android Studio

I farem clic en Finalitzar una vegada acabe:

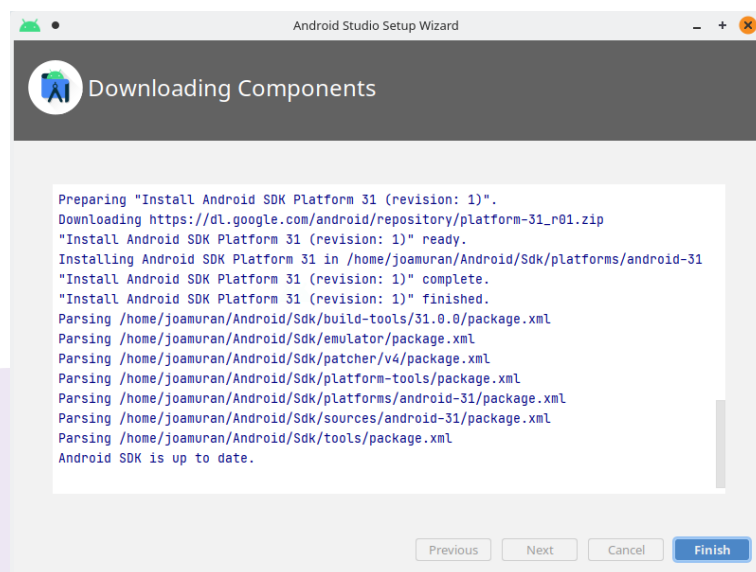


Figura 9: Instal·lació d'Android Studio

Amb això, ja tenim Android Studio Instal·lat al nostre equip, que ens mostrarà la següent pantalla de benvinguda.

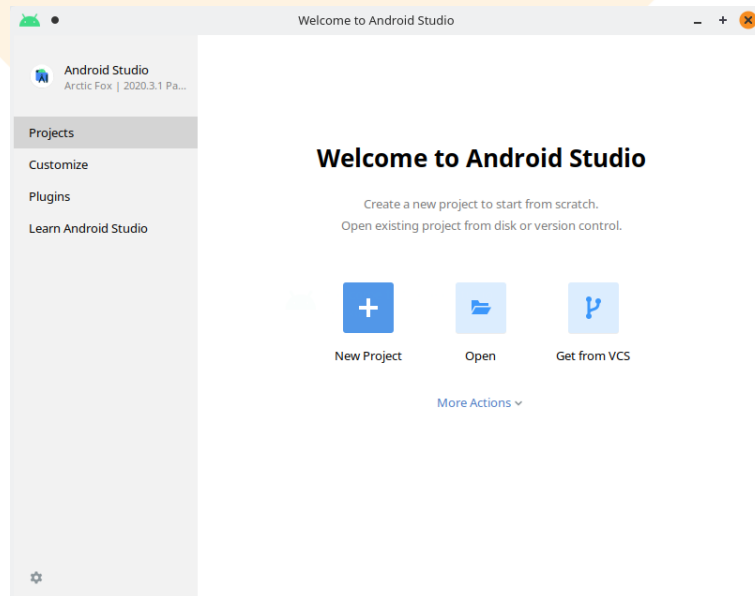


Figura 10: Instal·lació d'Android Studio

1.1.1 Creant una entrada al menú

Tal i com l'hem instal·lat, l'executable per llançar Android Studio serà el fitxer `~/.local/share/android-studio/bin/studio.sh`.

Per tal de no haver de buscar cada vegada aquest llançador, podem crear una entrada en el menú, de forma manual.

Per a això, creem el fitxer `~/.local/share/applications/androidstudio.desktop`, sent `~` el vostre directori personal, i fegirem el següent contingut (reemplaçant *usuari* pel teu nom d'usuari):

```
[Desktop Entry]
Name=Android Studio
Comment=IDE for Android
GenericName=Android Studio
Exec=/home/usuari/.local/share/android-studio/bin/studio.sh
Icon=/home/usuari/.local/share/android-studio/bin/studio.svg
Type=Application
StartupNotify=false
StartupWMClass=studio.sh
Categories=Utility;Development;IDE;
Actions=new-empty-window;
```

```
Keywords=vscode;  
X-Desktop-File-Install-Version=0.24
```

2 Primers passos amb Android Studio: Hola món

Després de la instal·lació, i sempre que tornem a obrir Android Studio sense projectes recents, se'ns presenta una finestra de benvinguda semblant a la següent:

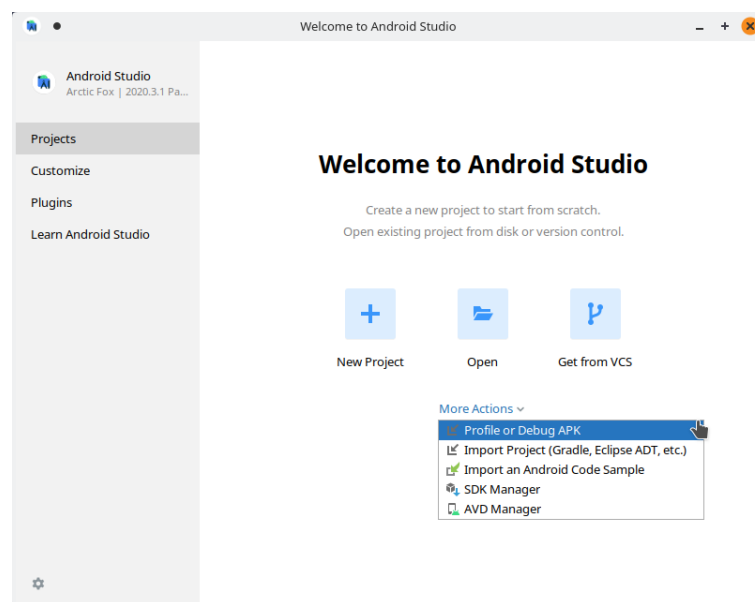


Figura 11: Finestra Inicial d'Android Studio

Des de la que podrem fer varies coses, tals com:

- Iniciar un projecte nou
- Obrir un projecte que ja existeix al nostre ordinador
- Obtenir un projecte des d'un sistema de control de versions
- Depurar aplicacions
- Importar altres projectes creats amb altres IDEs
- Importar codis d'exemple
- Gestionar l'SDK o l'AVD per als dispositius virtuals

Per tal de fer un xicotet exemple d'Hola Món, seleccionarem la primera *New Project* per començar un projecte nou.

Després ens demanarà que triem la plataforma per a la que desenvoluparem el nostre projecte i el tipus de projecte. Android Studio ens permet desenvolupar aplicacions per a diferents dispositius, tals com telèfons i tauleta, wereables, televisors, cotxes o IoT (Internet of Things).

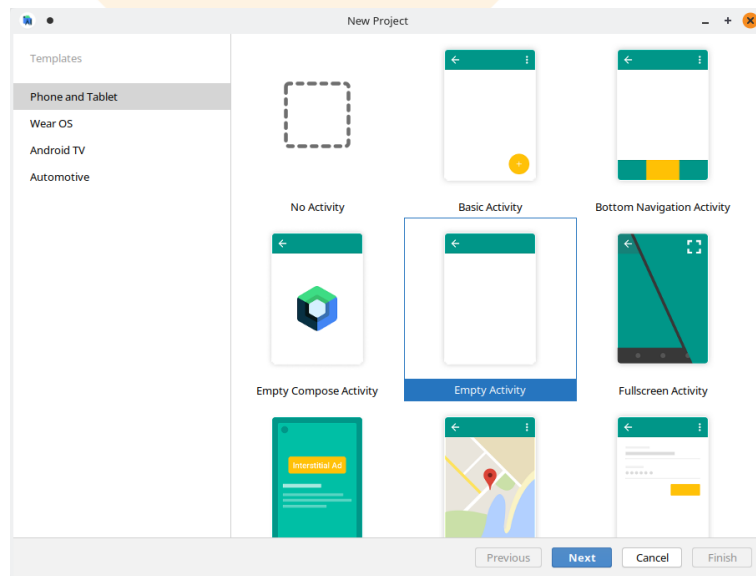


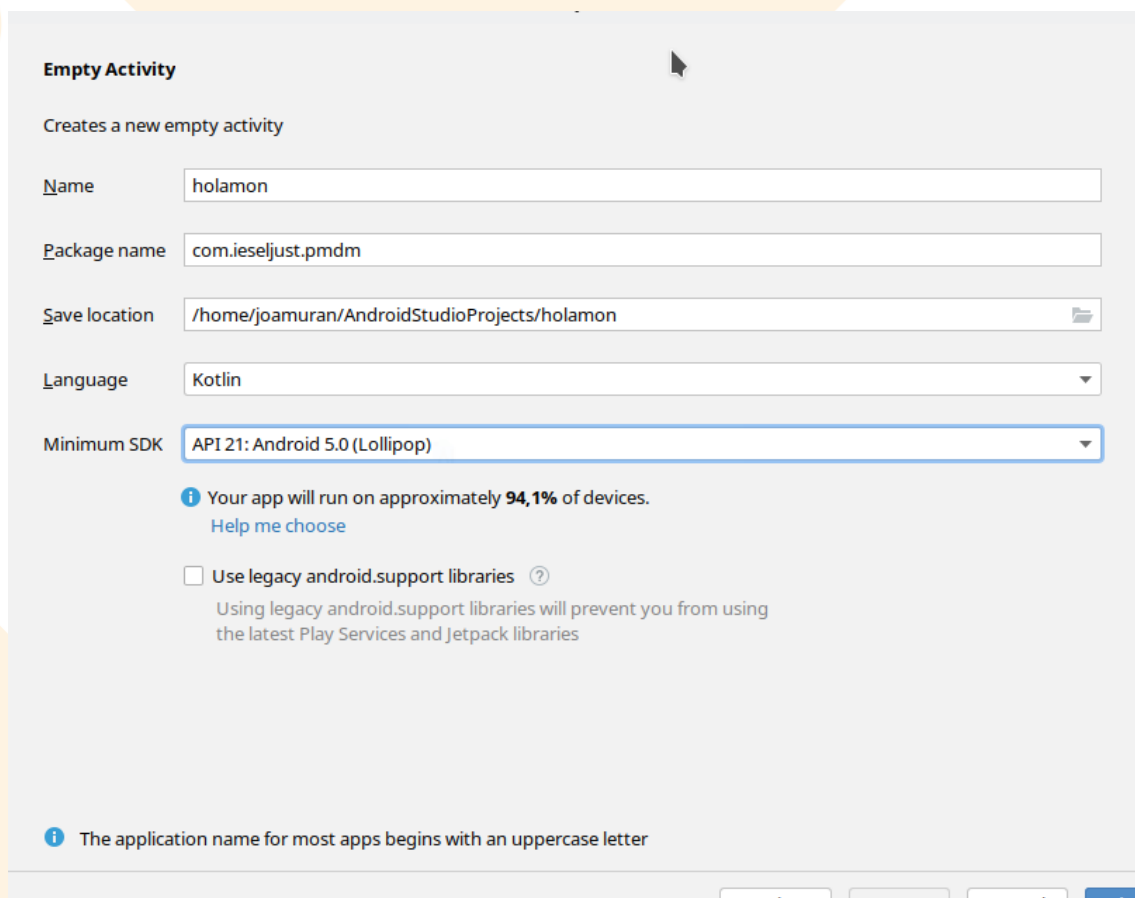
Figura 12: Selecció de la plataforma i tipus de projecte

Pel que fa al tipus de projecte, que triarem a la part de baix, se'ns ofereix una sèrie de plantilles predissenyades per començar. Per fer una aplicació de tipus *Hola Món* senzilla seleccionarem *Empty Activity*, per tal que siga una aplicació consistent en una única pantalla.



En Android, les diferents *pantalles* que conformen una aplicació es coneixen com *Activities* o *Activitats*.

En la pròxima pantalla se'ns demanen algunes dades del projecte, com el llenguatge i el SDK mínim que suportarem. Triarem el llenguatge Kotlin i la el SDK mínim Android 5.



Empty Activity

Creates a new empty activity

Name

Package name

Save location

Language

Minimum SDK

i Your app will run on approximately **94,1%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries **?**
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

i The application name for most apps begins with an uppercase letter

Previous Next Cancel Finish

Figura 13: Afegint informació de l'aplicació



Quin SDK escollir?

Escollir un SDK mínim dependrà de les funcionalitats que necessitem a la nostra aplicació i de la quantitat de dispositius a què volem arribar.

Si modifiquem el SDK, veurem que a la part de baix ens va canviant el percentatge de dispositius per als què funcionaria la nostra aplicació. Com més baixet triem l'SDK arribarem a més dispositius, però no tindrem totes les funcionalitats de les últimes versions de l'API de l'SDK. En canvi, si triem un SDK amb major nivell d'API, tindrem més funcionalitats, però la nostra aplicació serà suportada per menys dispositius.

Per ajudar-nos en la selecció, Android Studio ens ofereix si fem click a *Help me choose* un resum dels diferents SDKs que suporta, junt amb el nivell de l'API i les principals característiques d'aquest, així com el percentatge de dispositius que la suportarien.

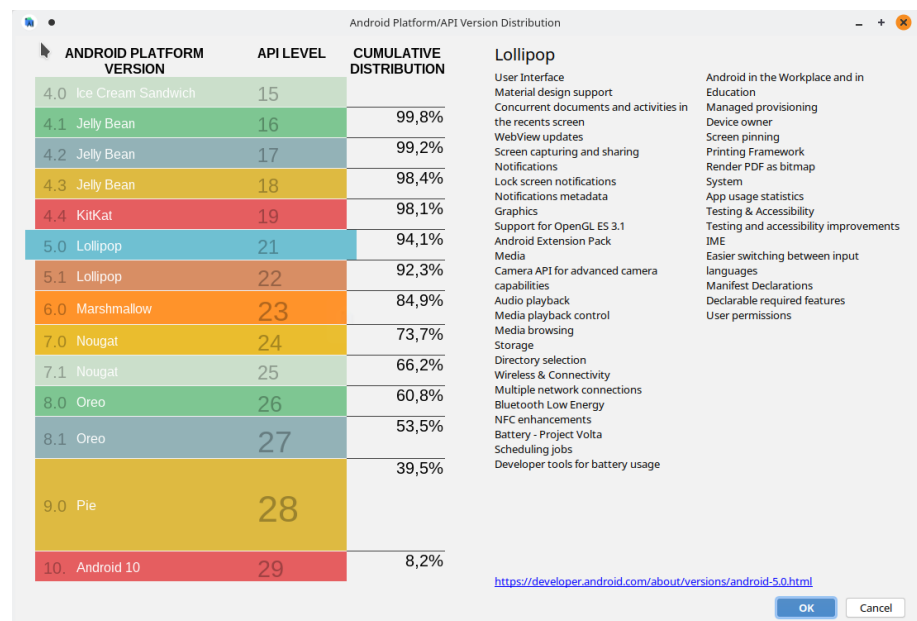


Figura 14: Versions de la plataforma d'Android

Una vegada tenim les característiques del projecte definides, donem clic a finalitzar per tal que ens genere tot el projecte.

2.1 La interfície d'Android Studio

Una vegada generat el projecte se'ns obri la següent finestra:

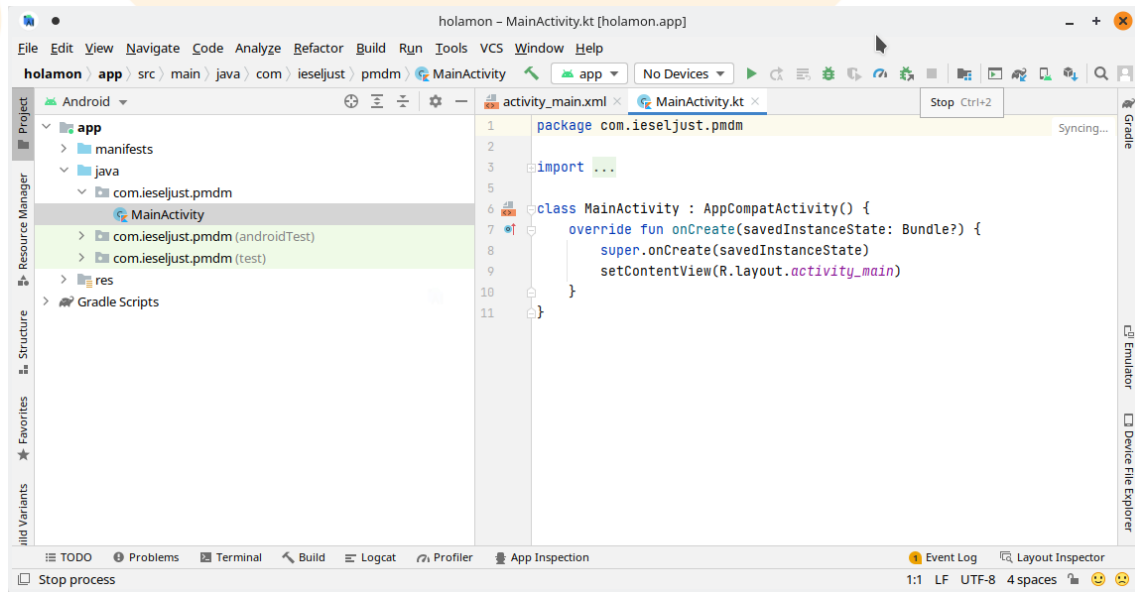


Figura 15: El nostre projecte en Android

Poc a poc anirem descobrint detalls d'aquesta interfície, però com veiem, no dista molt dels IDEs que ja coneixem. A la següent imatge podem veure les parts principals d'aquesta interfície:

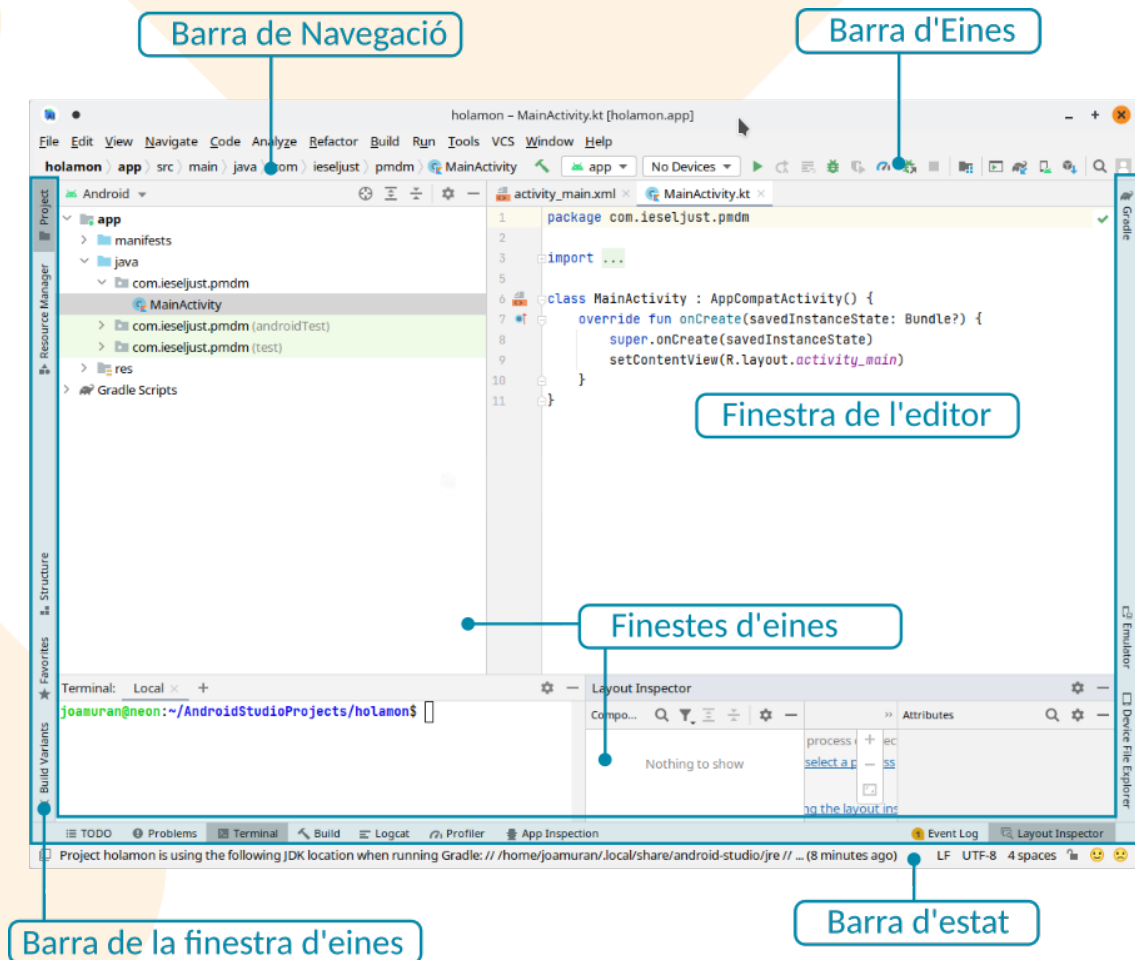


Figura 16: Interfície d'Android Studio

En ella distingim:

- La **barra d'eines** per guardar, obrir, construir o executar el programa, entre d'altres opcions,
- La **barra de navegació**, per explorar el projecte,
- La **finestra de l'editor**, on tenim el codi o el disseny de l'aplicació,
- La **barra de la finestra d'eines**, que envolta tot l'IDE i conté botons per expandir o contraure finestres d'eines individuals,
- Les **Finestres d'Eines**, amb diverses utilitats, com l'administració de projectes, recerca de text, control de versions, finestra de depuració, etc. A la finestra de l'esquerra d'Administració del projecte, podem veure que apareixen diverses *vistes* del nostre projecte. Les que més utilitzarem serà la pròpia del projecte (Project), la de fitxers (Project Files) o la d'Android.

Podem trobar més informació sobre la interfície a la web de desenvolupadors d'Android: <https://developer.android.com/studio/intro?hl=es-419>

2.2 La vista de fitxers del projecte

Centrant-nos ara en aquesta finestra lateral d'eines per veure com s'ha estructurat el projecte generat i com es mostra a les diferents vistes. Anem a seleccionar la vista de *Project Files* per veure més clar tota l'estructura generada:

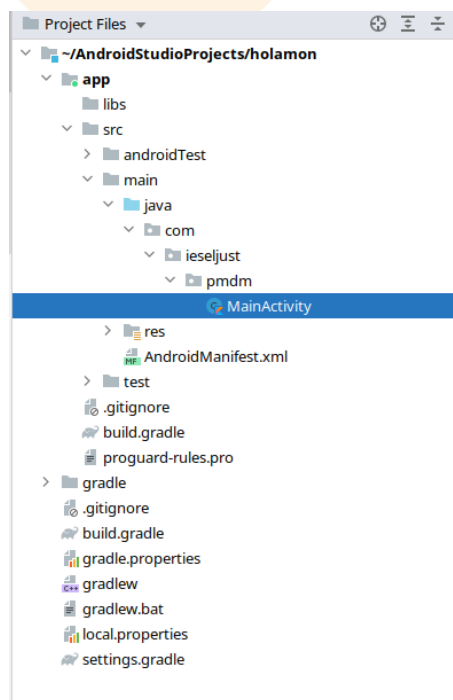


Figura 17: Fitxes del projecte

Com veiem, es tracta d'una estructura molt semblant a la dels projectes en Gradle que ja hem vist. En la carpeta arrel, disposem dels wrappers (gradlew i gradlew.bat), alguns fitxers de configuració, de git, i el més important: el fitxer `build.gradle`, de configuració del projecte. Si l'obrim veurem que està expressat en el format *Groovy* que ja coneixem i que inclou alguns repositoris i dependències, però no inclou cap classe per llançar l'aplicació.

Recordeu, que Gradle permet disposar de diverses aplicacions o mòduls per projecte. Aquest fitxer `build.gradle` contindrà informació comuna a tots els mòduls de què es compose el projecte.

Si accedim a la carpeta `app`, trobarem la carpeta `src` amb el codi font, més una carpeta buida per a llibreries, i altre fitxer `build.gradle`, aquest sí, amb la informació del mòdul o aplicació:

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



```
}

android {
    compileSdk 31

    defaultConfig {
        applicationId "com.ieseljust.pmdm"
        minSdk 21
        targetSdk 31
        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 '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:2.1.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Com veiem, apareix una nova secció `android`, i el plugin `com.android.application`, junt amb algunes coses més. Android Studio afeg un complement d'Android per als projectes Gradle, que ofereix algunes capacitats específiques per a Android. Els projectes en Gradle, permeten aquesta flexibilitat de tindre un fitxer de compilació `build.gradle` de nivell superior per a tot el projecte, i fitxers de compilació `build.gradle` a nivell de mòduls independents. Quan importem un projecte existent, Android Studio generarà automàticament aquests fitxers de compilació.

Un altre detall curiós és que a diferència d'un projecte Kotlin, la carpeta `src/main` conté una subcarpeta `java`, encara que el codi que conté siga kotlin.

Per altra banda, la carpeta de recursos `resources`, ara s'anomenarà `res`, i recordem que contindrà tots els recursos de la nostra aplicació. En projectes Android, aquesta carpeta rep encara més rellevància, ja que a banda d'imatges, emmagatzema també el codi XML de les interfícies d'usuari.

Dins aquesta carpeta `src/main`, també tenim el fitxer `AndroidManifest.xml`, amb informació sobre l'aplicació i els seus recursos (nom, icona, tema, activitat principal, etc.)

2.3 La vista Android

Si canviem a la vista d'Android, veurem els diferents mòduls de què consta el nostre projecte.

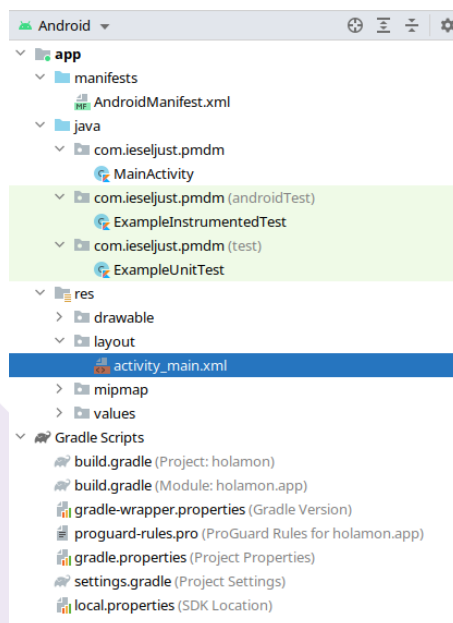


Figura 18: Vista Android

**Estructura d'un projecte Android Studio**

Cada projecte Android Studio inclou un o més mòduls amb fitxers de codi font i recursos. Aquests tipus de mòduls poden ser:

- Mòduls d'Apps per a Android
- Mòduls de biblioteca
- Mòduls de Google App Engine

Al nostre cas, només disposem d'un mòdul anomenat app, i dins d'ell distingim tres carpetes:

- La carpeta manifest amb el fitxer `AndroidManifest.xml`,
- La carpeta java, amb els diferents fonts i tests,
- La carpeta res, amb els diferents recursos de l'aplicació.

A més també tenim accés als diferents scripts de Gradle i fitxers de configuració.

2.4 Execució de l'aplicació

Tinguem en compte que encara no hem afegit cap línia de codi a banda del que ens genera automàticament el projecte.

Llançarem ara l'aplicació, de manera que puguem veure aquesta *empty activity* de què consta.

Per a això, farem clic en el *Play* de la barra d'eines, i observem què passa:

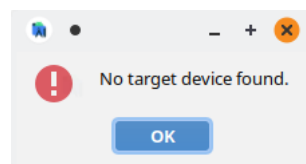


Figura 19: Error de dispositiu

Què ha passat? Doncs que no tenim cap *dispositiu* sobre el qual llançar l'aplicació.

Per configurar doncs un dispositiu tenim dues opcions:

- Fer ús d'un emulador, o bé
- Executar-lo en un dispositiu físic

Anem a fer ús d'un emulador. Per a això, ens fixem en la barra d'eines, al desplegable que de moment marca *No Devices*:

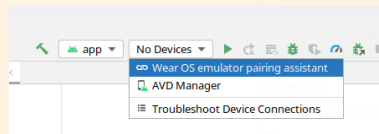


Figura 20: Obrir AVD Manager

Fem clic en ell, per obrir ara el *AVD Manager*, i dins d'aquest, crearem un dispositiu nou (*Create Virtual Device*).

És important escollir un dispositiu que dispose de Google Play, com per exemple el Pixel 3. Després triarem el sistema operatiu que volem en ell. En el nostre cas, hem triat Android 10. Tingueu en compte que caldrà descarregar el sistema abans de continuar, el que pot ser un procés bastant costós, segons tot el què estiguem instal·lant (Sistema operatiu, playstore, etc.)

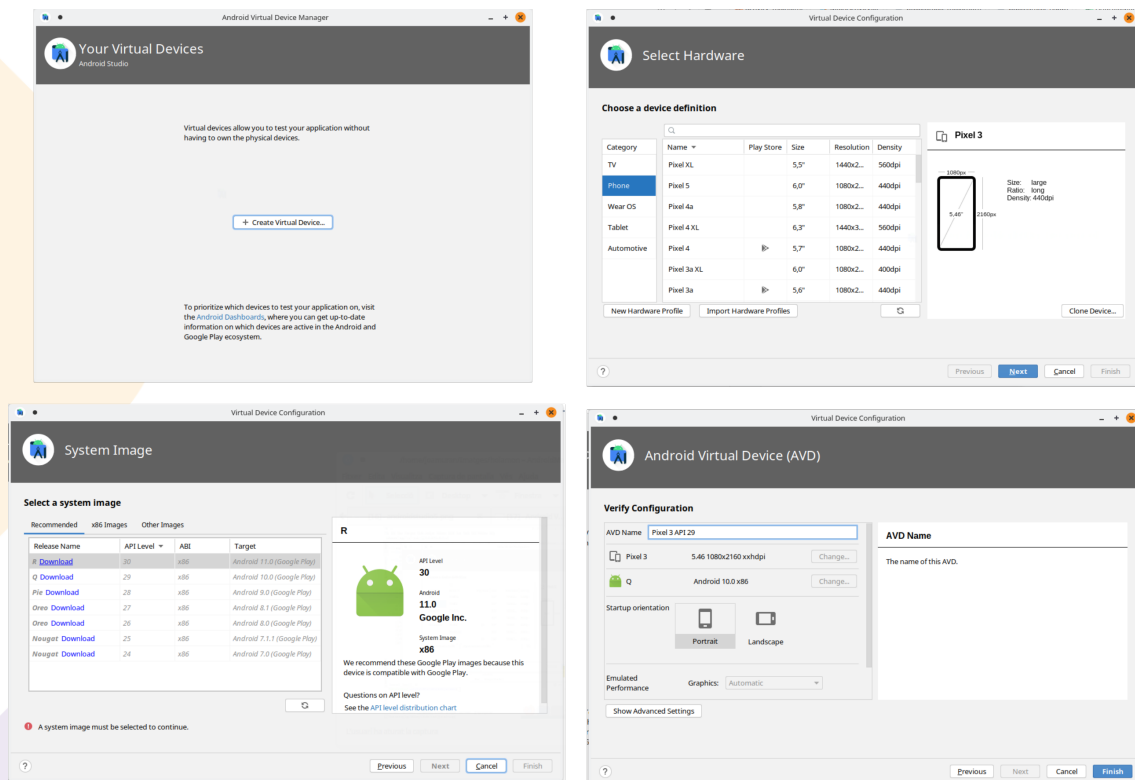


Figura 21: Creació del dispositiu

I ara ja, finalment, podem llançar l'aplicació en el dispositiu emulat, i com veurem, mostra el missatge *Hola Mon*.



Figura 22: Hola Món

2.4.1 Activant la virtualització a l'emulador

Com hem comentat, en GNU/Linux és possible activar la virtualització basada en el nucli per millorar el rendiment de l'emulador.

Al lloc web https://developer.android.com/studio/run/emulator-acceleration?utm_source=android-studio#vm-linux se'ns explica com podem habilitar KVM al nostre sistema.

Per comprovar si tenim KVM habilitat, podem fer ús de l'ordre `kvm-ok` del paquet `cpu-checker`:

```
$ kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

Si és així, una vegada tenim l'emulador descarregat, podem activar l'acceleració de la màquina virtual.

Per a això, només cal que, des de la línia d'ordres activem el paràmetre `-accel-check` de l'emulador. Si hem instal·lat el SDK en `~/Android/Sdk/emulator` podem activar-ho amb:

```
~/Android/Sdk/emulator/emulator -accel-check
```

```
accel:
0
KVM (version 12) is installed and usable.
accel
```



Disposeu de més informació sobre l'acceleració de la màquina virtual i gràfica al següent enllaç:

- <https://developer.android.com/studio/run/emulator-acceleration?hl=es>

3 Fitxers d'Activities i Layouts

Tot i que hi haurà alguna variació, podem dir que cada pantalla de la nostra aplicació és una *Activity*. Aquestes *Activitats* van a tindre una part gràfica i altra lògica, representades en els seus respectius fitxers: el fitxer que es correspon a la interfície gràfica, que es troba com un recurs dins la carpeta `res/layout`, i el fitxer amb el codi font de la part lògica, ubicat a la carpeta `src/main/java`

Per a la finestra principal de la nostra aplicació *MainActivity*, disposem del següent fitxer font *MainActivity.kt*

```
package com.ieseljust.pmdm.holamon

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
```

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

La sintaxi d'aquest codi ens és familiar. En primer lloc, defineix el paquet com `.ieseljust.pmdm.holamon` i importa un parell de classes:

- `androidx.appcompat.app.AppCompatActivity`: Classe base per a activitats que utilitzen funcionalitats de la plataforma més noves en dispositius més antics, com la barra d'accions o el canvi integrat entre temes clars. Podeu consultar aquesta classe en la referència d'Android
- `android.os.Bundle`: Classe per mantindre l'estat de l'activitat quan aquesta es recarrega, i per ajudar a la comunicació entre activitats.

Després, defineix la classe *MainActivity* com a una classe descendent de la classe *AppCompatActivity*, sobreescriu el seu mètode `onCreate`. Aquest mètode rep un objecte de tipus *Bundle* anomenat `savedInstanceState`, que pot ser nul (?). A més, dins d'ell, invoca el mètode `onCreate` de la seua classe pare, passant-li el mateix objecte. Poc a poc anirem veient què és tot açò.

Per altra banda, als recursos del sistema tenim, dins la carpeta `layout` el fitxer `activity_main.xml`. Tot i ser un fitxer XML, Android Studio ens mostra una interfície WYSIWYG (*What you see is what you get*), de manera que veiem el *disseny* de la pantalla, junt amb diferents elements d'interfície que podríem afegir, simplement, arrossegant i soltant en aquesta.

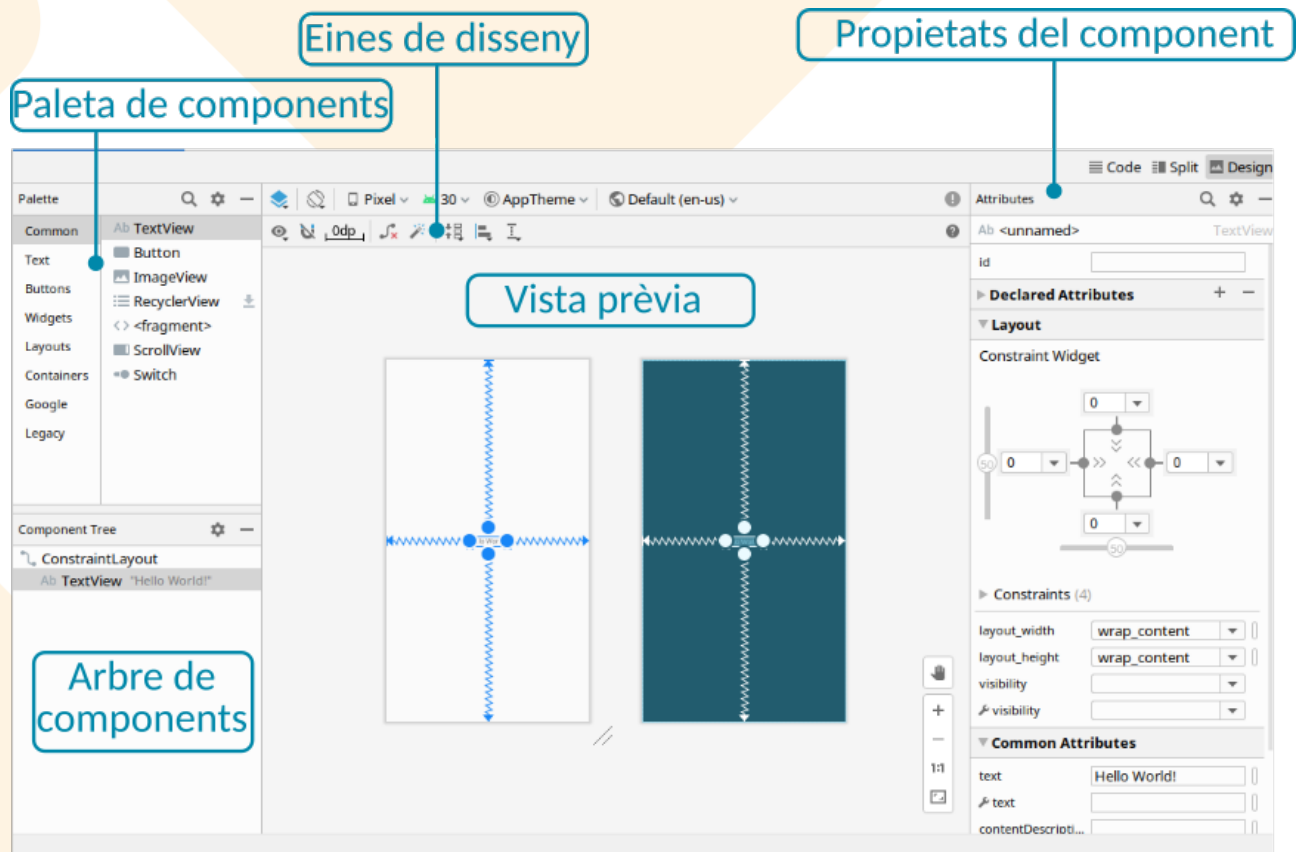


Figura 23: Dissenyador d'interfícies

Com veiem, aquesta vista de disseny del Layout ens permet modificar tant les propietats dels objectes que tenim a l'activitat, a través de les *Propietats del component* i les seues *Eines de Disseny*, així com afegir nous elements mitjançant la paleta de components i veure com aquests s'organitzen mitjançant l'*Arbre de components*. Si ens fixem en aquest arbre de components, veurem que tenim un component de tipus *TextView* dins un *ConstraintLayout*.

Per altra banda, si volem accedir a consultar el codi d'aquesta interfície, ho podem fer amb la barra de botons ubicada a la part superior dreta de la vista de disseny, i on podem alternar entre el codi XML, la vista de disseny o una vista híbrida:

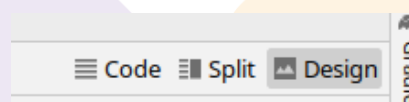


Figura 24: Dissenyador d'interfícies

Proveu a modificar la propietat *Text* del *TextView*, i torneu a llançar l'aplicació en l'emulador.

Adoneu-vos que en aquest cas, no hem definit el missatge al codi del programa principal Kotlin, sinó que el missatge està definit dins la mateixa interfície. No obstant això, aquest missatge es pot modificar des del mateix codi Kotlin.

Per altra banda, podem consultar l'XML de la interfície, per veure'n quin format té i com s'expressen aquests atributs:

```
<?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>
```

3.1 Enllaçant Layout i Activities

Ja hem vist quin aspecte tenen el fitxer de definició de la interfície en XML de la finestra principal (*Layout*) i el fitxer que gestiona la seua lògica.

Anem a veure ara com s'estableix la relació entre aquest *Layout* i el codi font.

Fixem-nos en el mètode *onCreate* de la classe *MainActivity*:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}
```

Aquest mètode, heretat de la classe `AppCompatActivity` s'invoca com a resposta a l'esdeveniment de creació de l'activitat (funciona per tant com un *callback*). En ell, veiem que primer s'invoca al constructor de la classe pare, i després s'invoca el mètode `setContentView`, que és qui s'encarrega de *carregar* el *Layout*. Per tal de fer referència a aquest *Layout* ho fa a través de `R.layout.activity_main`.

Com veurem més endavant, una *Activity* té un cicle de vida, al llarg del qual ocorren determinats esdeveniments. En aquest cas, estem associant una funció de *callback* que es llançarà quan es cree (`onCreate`) l'activitat. Tot i que ho veurem més endavant, podem trobar informació sobre el cicle de vida d'una *Activity* en <https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es>



Què és la classe R?

La classe R (de **R**esources) conté les definicions de tots els recursos d'un paquet, i es troba a l'espai de noms d'aquest.

Per exemple, si el nostre paquet s'anomena `com.ieseljust.pmdm`, tindrem una classe R amb tots els símbols dels recursos, que s'anomenarà `com.ieseljust.pmdm.R`.

A més, també disposem de la classe R dels recursos del framework, a `android.R`.

Aixó doncs, fent ús de la classe R, busquem el recurs de tipus *layout* i dins d'aquest el recurs *activity_main* (que es correspon al fitxer *activity_main.xml*).

3.1.1 Exemple

Anem a fer una xicoteta modificació en aquest *Hola món*, per tal que quan es fa click en ell, canvie el text.

Per a això, treballarem la interacció igual que amb qualsevol interfície gràfica: associant una acció a quan es produisca un esdeveniment.

El primer que necessitarem doncs serà afegir-li un identificador al text, de manera que puguem referir-nos a ell fàcilment. Això podem fer-ho tant des del dissenyador d'interfícies com editant directament el fitxer XML.

Si accedim al dissenyador d'interfícies i fem clic sobre el `TextView` que conté el missatge d'*Hola Món*, a la part dreta veurem que podem consultar i editar les seues propietats. Així doncs, fem clic a l'*id* i escrivim el nom d'identificador `textHolla`:

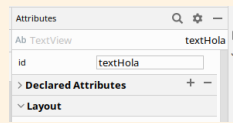


Figura 25: Assignant un identificador al text

Si tenim la vista partida, o accedim a la vista de codi de la interfície, veurem que ens ha aparegut un nou atribut al textview, amb nom `android:id`, que és l'identificador, i amb valor `"@+id/textHola"`. Aquest `@+id` fa referència a que estem definint un nou identificador als recursos de l'aplicació:

```
<TextView
    android:id="@+id/textHola"
    ...
/>
```

Ara tornem al codi principal de l'activitat, i modifiquem el mètode `onCreate` amb el següent codi:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    var textHola= findViewById<TextView>(R.id.textHola)

    textHola.setOnClickListener {
        textHola.text = "Benvingut a Android!"
    }
}
```

Veiem què hem fet en aquestes noves línies:

```
var textHola= findViewById<TextView>(R.id.textHola)
```

Amb açò hem definit una variable anomenada `textHola`, i l'hem inicialitzada per a que faci referència al recurs de tipus `TextView` `R.id.textHola`. Per a això hem fet ús del mètode `findViewById` que ens busca una vista (element gràfic) que es corresponga amb l'identificador `textHola`, que és l'identificador que li hem afegit al component.

```
textHola.setOnClickListener {  
    textHola.text = "Benvingut a Android!"  
}
```

Una vegada tenim la referència al *TextView* gràcies a l'identificador, hem associat una acció (en forma d'expressió lambda) a quan es produeix l'esdeveniment *click* sobre aquest. Això ho aconseguim amb el mètode `setOnClickListener`. Fixeu-vos que no li passem cap argument, i que aquesta expressió s'aplica sobre la pròpia vista (*it:View!*). Dins aquesta funció, el que fem és modificar l'atribut *text* del *textView* *textHola*, amb la cadena *Benvingut a Android!*.



Recordeu que tot i que estem accedint a través de `.text`, realment estem accedint al mètode `set` que Kotlin ha generat per a aquest atribut.

3.1.2 Exercicis

Exercici 1

- Sobre l'exemple anterior, executa'l i amb la rotació automàtica del dispositiu activada, prova a canviar el text fent clic sobre ell i després a rotar el dispositiu i observa què passa.

Exercici 2

- Amb el dissenyador gràfic, afeg un nou botó a la interfície (*Palette > Buttons Button*), de manera que el text canvie quan es fa click sobre el botó, en lloc de sobre el text.

Exercici 3

- Afeg ara dos botons, un `+` i un `-`, inicialitza un comptador a 0 i fes que es mostre aquest en lloc del missatge d'Hola Món. A més, cada vegada que es fa click al `+` o al `-`, aquest comptador s'incrementarà o es decrementarà en funció de botó que haja polsat.

3.2 Enllaçant Layouts i Activities amb View Binding

Amb la versió 3.6 d'Android es va llençar el mecanisme del *View Binding* per accedir a les vistes d'un XML des del codi.



Quan parlem de vistes en una Activitat o Fragment d'Android, fem referència als diferents elements de la interfície: Botons, *textViews*, etc.

La llibreria Data Binding, ens permet accedir de forma senzilla a la vista, enllaçant variables del nostre codi Kotlin o Java amb els components de l'XML.

Tot i que el mecanisme és complex, ja que implica la possibilitat d'afegir codi dins els XMLs, existeix els objectes Bindings, que simplifiquen aquest mecanisme i ens poden ajudar en la nostra tasca. Aquests *Bindings*, contenen totes les vistes de l'XML convertides als tipus correctes, i tenint en compte qualsevol possibilitat de valor nul de cada vista.

3.2.1 Habilitar el Data Binding

En primer lloc, haurem de modificar el fitxer `build.gradle` de l'aplicació per habilitar el View Binding. Afegirem el següent codi a la configuració d'Android:

```
android {  
    buildFeatures {  
        viewBinding = true  
    }  
}
```

3.2.2 Ús del View Binding

Ara caldrà modificar l'activitat per fer ús del View Binding, i modificar la forma en què *unflem* la vista.

En primer lloc, afegirem la llibreria `ActivityMainBinding`:

```
import com.ieseljust.pmdm.databinding.ActivityMainBinding
```

Aquesta llibreria ens l'ha generada el sistema automàticament quan hem inclòs el View Binding en el projecte. Si oblidem posar-la, Android Studio ens marcarà un error quan la utilitzem i ens suggerirà la seua importació.

En segon lloc, en el mètode `onCreate`, modificarem la línia que estableix el contingut d'aquesta:

```
setContentView(R.layout.activity_main)
```

Per:

```
val binding = ActivityMainBinding.inflate(layoutInflater)
setContentView(binding.root)
```

En lloc de generar la vista passant-li al `setContentView` l'identificador del *Layout*, el que fem és *unflar* la vista amb el *View Binding* i establim el contingut proporcionant l'arrel d'aquest *binding*.

Aquest *binding*, contindrà doncs tots els elements (vistes) de la interfície que hem carregat, de manera que ara, per associar els esdeveniments i modificar el valor farem:

```
binding.textHola.setOnClickListener {
    binding.textHola.text = "Benvingut a Android!"
}
```



Teniu més informació sobre el View Binding a:

- *View BINDING – El método DEFINITIVO para acceder a las vistas en ANDROID*, de Antonio Leiva

3.2.2.1 Exercicis

- Modifica els exercicis 2, 3 i 4 anteriors fent ús del View Binding.