Programació multimèdia
i dispositius mòbils

UD 02. Programació d'aplicacions per a dispositius mòbils. Android.

Visió general del procés de desenvolupament. El cicle de vida de les activitats.

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2021-22

Continguts

1 Visió general del desenvolupament d'apps en Android **3**
1.1 Components d'una aplicació 3
1.1.1 Activació de components. Intents. 5
1.2 El fitxer de Manifest 5
1.3 Recursos de l'aplicació 7

2 Activitats, Layouts, Vistes i Fragments **8**
2.1 Vinculació o Bind de vistes 8
2.2 Activitats i el seu cicle de vida 10
2.2.1 Gestionant el cicle de vida de les activitats 10
2.2.2 Logs 13

1 Visió general del desenvolupament d'apps en Android

Les aplicacions Android es programen en Kotlin, Java o C++, i són compilades per l'Android SDK, junt amb fitxers amb dades i recursos (interfícies, imatges)… per generar un fitxer `.apk`. Aquests fitxers inclouen l'aplicació i la informació que necessita Android per instal·lar-la.

Android implementa el principi del *mínim privilegi* en les aplicacions, amb el que aporta certa seguretat a les aplicacions. Aquest principi es tradueix en:

- Android és un sistema operatiu multiusuari, ja que es basa en Linux. La peculiaritat és que pe a Android, cada aplicació serà un usuari diferent.
- Cada app té un ID d'usuari conegut només pel sistema, i estableix els permisos necessaris perquè aquesta puga accedir als seus recursos,
- Cada procés té la seua màquina virtual, de manera que el codi s'executa de forma independent, i cada app tindrà, en principi, el seu propi procés.

Amb açò, cada aplicació té només accés als components que necessita. De tota manera, una aplicació pot compartir dades amb altres aplicacions i accedir a serveis del sistema, bé fent que dos aplicacions compartisquen l'ID d'usuari o bé sol·licitant permís a l'usuari per accedir a dades i recursos del dispositiu (càmera, micròfon, connexió bluetooth, targeta SD, contactes, etc.)

Exemple inicial
Durant aquesta part de la unitat, seguirem treballant amb l'exemple que vam fer del comptador. És interessant que mantingueu aquest exemple obert per poder fer bé el seguiment i algunes modificacions que veurem.

Documentació oficial: *Conceptes bàsics sobre aplicacions Android*
https://developer.android.com/guide/components/fundamentals?hl=es-419

1.1 Components d'una aplicació

Les aplicacions Android tenen quatre tipus de components: activitats, serveis, receptors d'emissions i proveïdors de continguts.

- **Activitats *(Activities)*:** Representen les diferents pantalles de l'aplicació i recullen la interacció amb l'usuari. Cada activitat pot ser un punt d'entrada a l'aplicació, i pot ser requerida per altres

aplicacions. Pensem per exemple quan fem una foto i volem compartir-la directament per una aplicació de missatgeria instantània. L'activitat de l'aplicació que se'ns obri és directament la d'enviar missatges.

Les activitats, a més de mostrar el procés per pantalla a l'usuari, tindran en compte aspectes com la gestió dels processos recents (activitats aturades), ajudar en la finalització de processos de manera que es puga retornar amb l'estat restaurat, i permetre que el sistema coordine el flux de dades entre usuaris (aplicacions), com per exemple, *compartir*. Les activitats s'implementen com a subclasses de la classe `Activity`. Si ens fixem a l'exemple de l'*Hola Món*, definíem la nostra activitat com a `class MainActivity : AppCompatActivity()`, on `AppCompatActivity` és una subclasse d'`Activity`.

- **Serveis *(Services)*:** Els serveis són punts d'entrada general que ens permeten mantenir l'execució de l'aplicació en segon pla i que no proporcionen interfície d'usuari, com poguera ser un servei de reproducció d'àudio en segon pla o la sincronització de dades des de la xarxa, permetent que l'usuari es trobe interactuant amb altra activitat. Les activitats o altres components poden iniciar serveis i permetre que s'executen o enllaçar-se a ells per tal d'interactuar.

  Els serveis s'implementaran com a subclasses de la classe `Service`.

- **Receptors d'emissions *(Broadcast receivers)*:** Es tracta de components que recullen certs esdeveniments del sistema fora del flux habitual de les aplicacions, i permeten fins i tot enviar emissions a aplicacions que no estan en execució. Exemples d'emissions llançades pel propi sistema poden ser l'anunci que s'apaga de la pantalla, que queda poca bateria, o que s'ha fet una captura de pantalla. Les aplicacions també poden iniciar emissions, com per exemple per avisar que s'han descarregat dades al dispositiu. Aquests components tampoc tenen interfície gràfica, però poden crear notificacions en la barra d'estat o servir com a porta d'entrada a altres components.

  Aquests receptors s'implementen com a subclasses de `BroadcastReceiver`, i cada receptor d'emissió es lliura com un objecte `Intent`.

- **Proveïdors de continguts *(Content providers)*:** Administren dades compartides de l'aplicació, que es poden emmagatzemar bé al sistema de fitxers, en una base de dades SQLite, al núvol… Un clar exemple són els contactes de l'usuari. Android proveeix a qualsevol aplicació amb els permisos corresponents l'accés als contactes. Els proveïdors de continguts s'implementen com a subclasses de `ContentProvider`.

Un dels aspectes més interessants del disseny d'Android, i a diferència d'altres sistemes, és que **les aplicacions no tenen un únic punt d'entrada o funció principal**, sinó que podem tindre diferents punts d'entrada a cada component de l'aplicació. A més, **qualsevol aplicació pot iniciar un component d'altra aplicació**, sense necessitat d'incorporar o vincular l'altra aplicació al nostre

codi. L'exemple més clar és poder fer una foto amb l'aplicació de la càmera, de manera que en la nostra aplicació, només ens cal iniciar l'activitat corresponent en l'aplicació de la càmera. En aquest moment, s'iniciarà (si no estava en execució) el procés per a l'aplicació de la càmera, creant una instància de les classes necessàries.

Com hem dit, en Android cada aplicació té un procés amb un usuari diferent, i per tant uns permisos que limiten l'accés a altres aplicacions. Per tant, com podem des d'una aplicació accedir a components d'una altra? La resposta és que les nostres aplicacions no poden fer-ho, però Android sí, de manera que per activar un component en altra aplicació, haurem d'enviar un missatge al sistema per demanar-li un component específic i que el sistema l'active per nosaltres. Aquests missatges seran els *Intents* o *intencions*.

1.1.1 Activació de components. Intents.

Les activitats, els serveis i els receptors d'emissió s'activen mitjançant missatges asíncrons coneguts com *Intents* (objectes `Intent`), i que vinculen en temps d'execució diferents components. Aquests missatges poden servir per activar un component específic (intent explícit) o un *tipus de component* més genèric (intent implícit).

Documentació oficial: *Activació de Components*
https://developer.android.com/guide/components/fundamentals?hl=es-419#ActivatingComponents

1.2 El fitxer de Manifest

Per tal de poder iniciar un component d'una aplicació, Android necessita saber que aquest existeix. Per a això, només cal que ho indiquem al fitxer de manifest (`AndroidManifest.xml`) de l'aplicació.

Així doncs, al fitxer de Manifest s'indiquen els components d'aquesta, i a més, també podem:

- Indicar els permisos de l'usuari que requereix l'aplicació (accés a dispositius, contactes, Internet…)
- Indicar l'API mínima de l'aplicació,
- Indicar les característiques de programari i maquinari que requereix l'aplicació (càmera, bluetooth, pantalla multitàctil, etc.)
- Indicar les biblioteques de l'API a les que l'aplicació estarà vinculada (com per exemple Google Maps).

Analitzem el fitxer de Manifest del nostre primer projecte d'Hola Món.

- L'etiqueta `<application>` defineix als seus atributs aspectes bàsics de l'aplicació, com els recursos de les icones a utilitzar `android:icon` i `android:roundIcon` o el nom de l'aplicació `android:label`.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <application
        ...
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        ... >
```

  Dins d'*aplication* cal registrar tots els components de l'aplicació, mitjaçant els elements corresponents `<activity>`, `<service>`, `<receiver>` o `<provider>`. Els components que no incloguem aci, no estaran visibles per al sistema, i per tant, no es podran executar.

- Veiem l'etiqueta `<activity>` que conté el nostre fitxer. En aquesta, l'atribut `android:name` especifica el nom de la classe `Activity` plenament qualificat, i admetria l'atribut `android:label` si volguérem especificar una etiqueta de l'activitat de cara a l'usuari.

```
<activity android:name=".MainActivity"...>
    ...
</activity>
```

- I dins d'*activity*, trobem l'etiqueta `<intent-filter>`. Com hem comentat abans, els *Intents* són misstges que s'utilitzen per activar els diferents components, com activitats, serveis i receptors. Els Intents poden explicitar el component objectiu, fent ús de la classe del component, o bé indicar el component de forma implícita, indicant el tipus d'acció (i de forma opcional les dades amb què es realitzarà aquesta).

  En cas que indiquem el tipus de component, serà el mateix sistema el qui busque un component que puga realitzar l'acció i iniciar aquesta. En cas que hi haja més d'un component que realitze l'acció que requereix l'*Intent*, es demanarà a l'usuari que seleccione l'acció que vol.

  Per tal d'indicar al sistema quins components de cada aplicació poden respondre a un *Intent*, fem ús dels filtres d'intents (element `<intent-filter>`) dins del mateix component.

  Així doncs, al nostre cas, dins de l'*activity* tenim:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
```

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

Amb l'`<action android:name="android.intent.action.MAIN" />`, especifiquem que l'activitat és el punt d'entrada a l'aplicació, de manera que quan carreguem l'aplicació, és esta l'activitat que es crea. Per la seua banda, especificant en `<category android:name="android.intent.category.LAUNCHER" />`, indiquem que aquesta acció estarà disponible com un llançador a nivell d'aplicació (i per tant, ens apareixerà a la pantalla d'aplicacions). Si especifiquem per exemple `android.intent.category.DEFAULT`, estarem indicant que es tracta d'una acció que respon a intents implícits.

Documentació Oficial
Podeu trobar més informació sobre els *Intents* i els filtres d'*Intents* a:
https://developer.android.com/guide/components/intents-filters?hl=es-419.

Finalment, al fitxer `AndroidManifest.xml` podem especificar altres etiquetes, per tal d'indicar els recursos que puga necessitar l'aplicació, tant de maquinari com de programari. Per exemple, podríem haverem escrit:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

Per indicar que anem a utilitar la càmera i que necessitem la versió mínima de l'SDK 7 (Android 2.1)

1.3 Recursos de l'aplicació

A banda del codi, les aplicacions poden necessitar d'altres recursos, ja siguen imatges, fitxers de so, animacions, menús, estils, etc. Com ja vam veure a l'exemple de l'*Hola Món*, la interfície d'usuari de les activitats s'especifiquen com a un recurs també, en format XML.

Dins la carpeta *resources*, tindrem organitzats els diferents tipus de recurs que podem tindre.

Si ens fixem en la nostra aplicació de base, tenim els següents subdirectoris dins els recursos:

- `drawable/`: Conté el contingut *dibuixable* de l'aplicació. És a dir, els fitxers d'imatges, però també diversos elements de disseny, com llistes d'estats, formes o elements relacionats amb les animacions.
- `mipmap/`: Conté diferents elements de disseny, com per exemple les icones de l'aplicació, en diferents densitats (ppp).
- `layout/`: Conté els fitxers XML amb la interfície de l'usuari.
- `values/`: Conté fitxers XML amb valor simples, com cadenes, valors enters o colors.

Per cadascun d'aquests recursos que utilitzem al projecte, l'SDK defineix un *ID de recurs* amb un número enter únic dins l'aplicació, que podem utilitzar per fer referència al recurs en qüestió des del nostre codi. Per exemple, la interfície d'usuari definida en `res/layout/activity_main.xml` és accessible a través de l'ID anomenat `R.layout.activity_main` generat automàticament per l'SDK. Recordeu que la classe R, generada automàticament per Android, conté les definicions de tots els recursos d'un paquet, i es troba a l'espai de noms d'aquest.

Documentació oficial
Podem trobar molta més informació sobre aquesta carpeta de recursos a la documentació d'Android en:

- https://developer.android.com/guide/topics/resources/providing-resources?hl=es-419.

2 Activitats, Layouts, Vistes i Fragments

Com hem vist, les activitats són un dels components principals de les aplicacions en Android i representen les diferents pantalles on recollim la interacció de l'usuari.

Aquestes activitats representen la interfície amb recursos de tipus Layout, on s'especifiquen els diferents elements d'interfície (botons, textos, etc.). Aquests elements seran elements especialitzats de la classe `View`.

2.1 Vinculació o Bind de vistes

Els diferents elements o vistes que conformen la interfície d'usuari als fitxers XML s'han d'enllaçar d'alguna manera al nostre codi. Aquest procés es coneix com a vinculació de vistes, i ja l'hem tractat al primer apartat de la unitat.

La vinculació de vistes consisteix a enllaçar els diferents elements de la interfície d'usuari especificada als fitxers de disseny XML amb el nostre codi.

Aquesta vinculació es pot fer de diverses formes, com hem vist, tot i que l'opció recomanada per Google és el *View Binding* a través de les llibreries *Data Binding*.

Recordeu que per activar el *View Binding* calía dir-ho explícitament al fitxer `build.gradle` amb:

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

Una vegada enllaçat, Android s'adona que s'han produit canvis al fitxer de compilació i ens demana sincronitzar-se. En aquest moment, es generaran l'objecte `databinding`.

Amb això, al fitxer `MainActivity.kt` el que vam fer va ser importar els bindings per al fitxer de la interfície `activity_main.xml`, que es crea al component `ActivityMainBinding` del `databinding`, de la següent manera:

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

I una vegada importat, calia *unflar*, en crear-se la vista, aquest amb amb el Layout, amb la següent ordre:

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

Una vegada *unflat*, tenim tots el components de l'activitat al Binding, de manera que establíem ara el contingut dela vista amb:

```
setContentView(binding.root)
```

I accedíem a les diferents vistes directament a través del binding.

2.2 Activitats i el seu cicle de vida

Les activitats i la forma en què aquestes s'inicien és part fonamental del model d'aplicació d'Android. Si recordem, una aplicació d'Android pot tindre diferents punts d'entrada, que són les activitats.

La classe `Activity` facilita aquest model, de manera que des d'una aplicació es poden invocar activitats d'altres aplicacions (recurdeu l'exemple de l'App de la càmera). Les nostres activitats s'implementen com a subclasses d'questa classe `Activity`.

En general, cadascuna d'aquestes activitats implementaran les diferents pantalles que componen la nostra aplicació. Com que la majoria d'aplicacions contindran diverses pantalles, implementaran diverses activitats. El més habitual és que hi haja una activitat que siga l'activitat principal, que serà la que apareixerà quan l'usuari inicie l'aplicació, i serà aquesta la que inicie altres activitats.

Tot i que les activitats treballen en conjunt, per donar una experiència coherent a l'usuari, les activitats tenen poca relació entre elles, el que facilita la invocació d'activitats entre diferents apps.

Cada activitat haurà d'estar registrada en el fitxer de Manifest, i a més, haurà de gestionar els seus cicles de vida de forma correcta.

2.2.1 Gestionant el cicle de vida de les activitats

Les activitats, al llarg de la seua vida útil, passen per diferents estats. En el canvi d'un estat a altre, es disparen certs esdeveniments, que podem capturar i gestionar mitjançant funcions de *callback*.

Veiem quines són aquestes:

- **`onCreate()`**: S'activa quan es crea l'activitat, i en ella inicialitzem els components essencials d'aquesta, enllaçarem les vistes i vincularem dades. Aci generalment utilitzem `setContentView()` per establir el *layout* de la interfície de l'activitat.
- **`onStart()`**: Es dispara sempre quan acaba `onCreate()` i l'activitat passa a estat *Iniciada*, passant a primer pla i tornant-se visible per a l'usuari.
- **`onResume()`**: Aquest callback es dispara just abans que l'activitat comence a interactuar amb l'usuari, i es troba a dalt de tot de la pila d'activitats, capturant tot el que introdueix l'usuari.
- **`onPause()`**: S'invoca quan l'activitat perd el focus i pasas a l'estat de *Detesa* (s'ha polsat *enrere* o *Recents*). Quan el sistema invoca `onPause()`, l'activitat encara està parcialment visible. Les activitats *deteses* poden continuar actualitzant la UI si l'usuari espera que això passe (com per exemple, actualitzar un mapa de navegació). Aquest callback no s'ha d'utilitzar per guardar dades de l'aplicació, de l'usuari, fer crides de xarxa o realitzar transaccions sobre bases de dades.

- **`onStop()`**: S'invoca quan l'activitat ja no és visible per a l'usuari, bé perquè s'estiga eliminant l'activitat, perquè se n'inicia una de nova o perquè es reanude altra que la cobrisca. Després d'ella, es pot llançar `onRestart()` o `onDestroy()`.
- **`onRestart()`**: S'invoca quan una activitat que estava *Detesa* passa de nou a estar en primer pla. Aquest callback, restaura l'estat de l'activitat en el moment en què esta es va parar. Després d'aquest callback, s'invoca sempre `onStart()`.
- **`onDestroy()`**: S'invoca aquest callback abans que s'elimine l'activitat, i s'usa per garantir que els seus recursos s'alliberen amb ella, i el procés que la conté s'elimine.

Gràficament, podem vore aquest cicle de vida de la següent forma general:

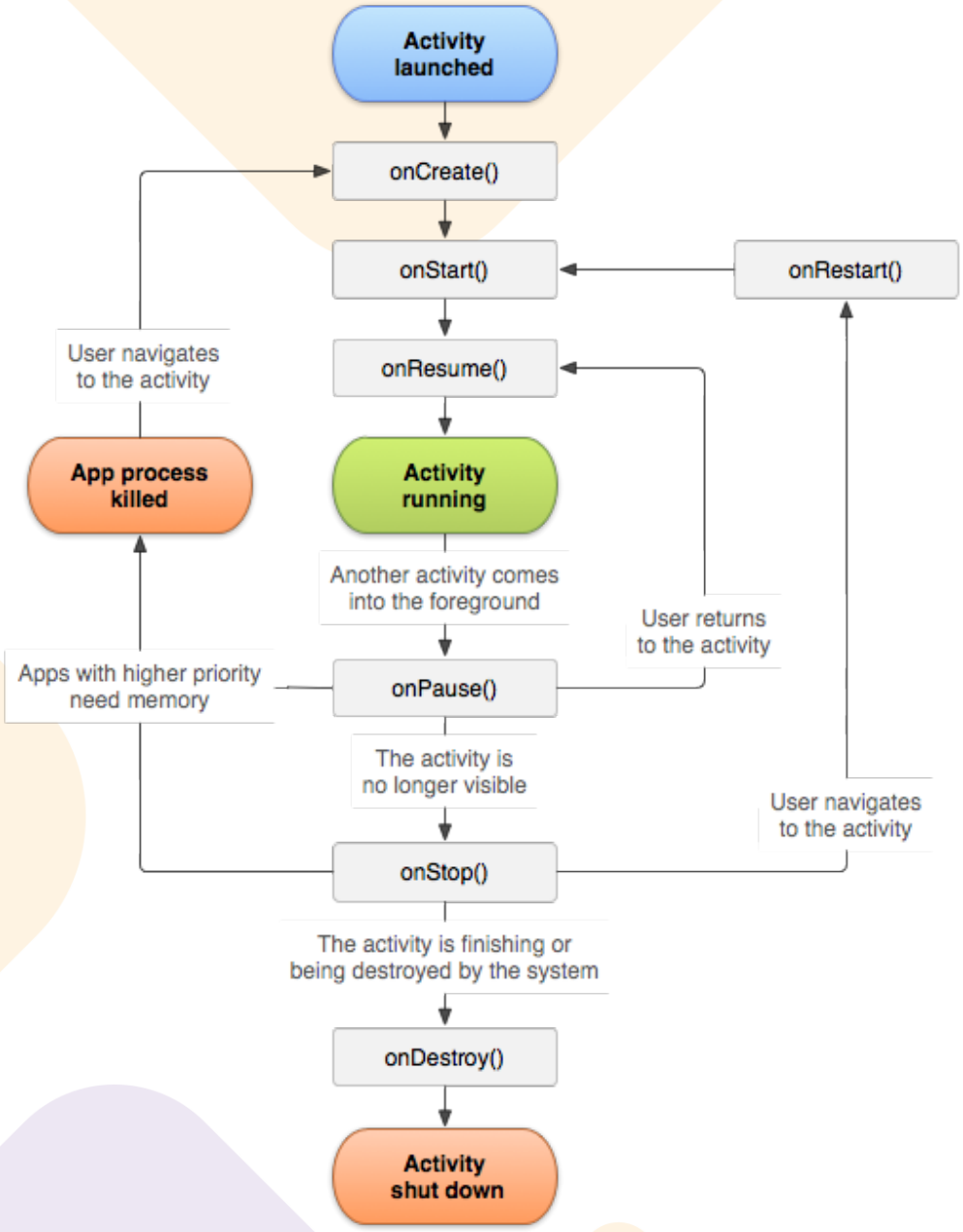

Figura 1: Cicle de vida d'una activitat

I centrant-nos en els diferents callbacks i estats:

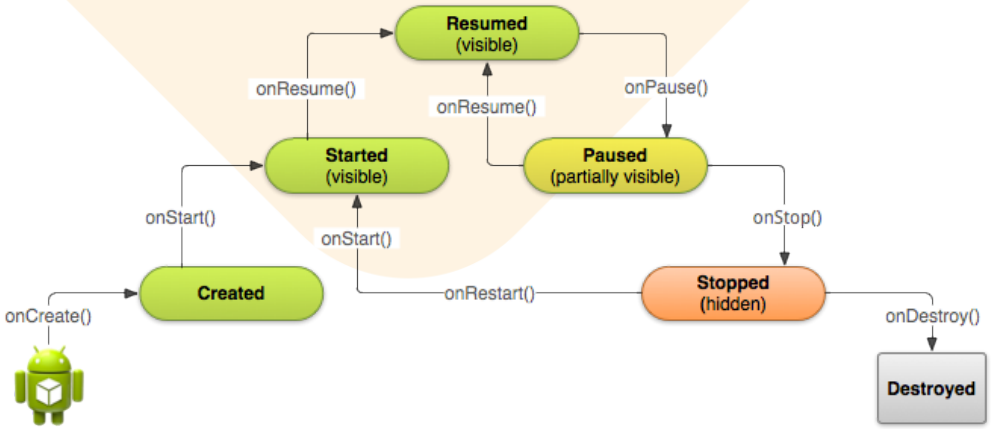

Figura 2: Cicle de vida d'una activitat

> ***Documentació oficial***
>
> Podem trobar molta més informació sobre el cicle de vida de les activitats en:
>
> https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es.
>
> I sobre la preservació d'estats durant el cicle de vida:
>
> https://developer.android.com/topic/libraries/architecture/saving-states?hl=es

2.2.2 Logs

Android ens ofereix, a través de la classe Log una API per enviar informació d'eixida a través d'un registre.

L'IDE d'Android Studio ens ofereix la finestra *Logcat*, on es mostren els diferents missatges del sistema, però també pot mostrar-nos missatges des de la nostra aplicació, si fem ús d'aquesta classe Log. Aquests missatges es mostraran en temps real, i es guardaran en un historial.

Aquesta finestra és bastant potent, i ens permet crear filtres de cerca, establir nivells de prioritat, mostrar només els missatges de l'app, buscar en el registre…

Quan la nostra app genera una excepció, aquesta finestra ens mostra un misstge seguit del seguiment de la pila (*Stack Trace*).

Tot i que a partir de la versió 2.2 d'Android Studio els missatges de la nostra aplicació també es mostren a la finestra *Run*, és aconsellable seguir utilitzant *logcat*, ja que ofereix més possibilitats en quant a gestió dels missatges.

Per accedir a la finestra *Logcat*, podem fer-ho bé a través del menú *View* > *Tool Windows* > *Logcat*, o bé en la pestanya *Logcat* en la barra de finestres d'eines, a la part de baix.

Per tal d'escriure misstges en aquesta finestra, la classe *Log* ens ofereix diversis mètodes, segons tipus de misstge que volguem escriure. De major a menor prioritat, estos mètodes són:

Mètode	Tipus de misstge
`Log.e(String, String)`	Misstge d'error
`Log.w(String, String)`	Missatge d'advertència (Warning)
`Log.i(String, String)`	Missatge d'informació
`Log.d(String, String)`	Misstge de depuración
`Log.v(String, String)`	Eixida detallada del registre

Com veiem, tots aquests mètodes necessiten dos arguments. El primer és una etiqueta que tindrà el missatge, per tal de poder-les filtrar, i el segon serà ja el text del missatge. Quan es tracta de misstges del sistema, l'etiqueta és un string curt que indica el component des del que s'ha originat el missatge.

És una pràctica habitual fer ús d'una constant TAG en les nostres classes per utilitzar-les com a etiqueta.

> **Documentació oficial**
>
> Disposeu de més informació sobre aquest finestra, les seues opcions a i més informació sobre els logs en: https://developer.android.com/studio/debug/am-logcat.html

A mode d'exemple per utilitzar logs i per aprofundir en el cicle de vida de les activitats, anem a fer que en cada canvi d'estat de l'activitat principal de l'app amb què estem treballant se'ns mostre un missatge de depuració. Per a això, ens situem en la nostra classe principal `MainActivity.kt`, i fem el següent:

- En primer lloc, incloem una constant al nostre codi per indicar el *tag* que anem a utilitzar. La podem definir com a atribut de la classe *MainActivity*:

```
val tag:String ="Lifecycle Events"
```

- En segon lloc, afegim un missagte al registre com a primera instrucció del mètode `onCreate()`, que ja tenim al nostre codi.

```
override fun onCreate(savedInstanceState: Bundle?) {
        Log.d(tag, "Entre al mètode onCreate")
        ...}
```

Veure que la classe *Log* no la reconeix, i si ens posem al damunt ens suggereix el missatge: `android.util.Log?  Alt+Enter`. Premem `Alt+Enter` per tal d'afegir l'import corresponent (`import android.util.Log`).

- Finalment, afegim nous mètodes de *Callback* a la classe, per capturar els diferents canvis d'estat:

```
override fun onStart() {
        super.onStart()
        Log.d(tag, "Entre al mètode onStart")
    }

override fun onResume() {
        super.onResume()
        Log.d(tag, "Entre al mètode onResume")
    }

    override fun onPause(){
        super.onPause()
        Log.d(tag, "Entre al mètode onPause")
    }

    override fun onStop() {
        super.onStop()
        Log.d(tag, "Entre al mètode onStop")
    }

    override fun onRestart() {
        super.onRestart()
        Log.d(tag, "Entre al mètode onRestart")
    }
    override fun onDestroy(){
        super.onDestroy()
        Log.d(tag, "Entre al mètode onDestroy")
    }
```

Una vegada fet açò, llanceu l'aplicació i fixeu-vos en l'evolució dels logs mentre feu les següents operacions:

1. Inici de l'aplicació
2. De la barra de navegació d'Android[1], fem clic al botó d'inici (central) per tornar a la pantalla principal o bé al botó de menú (dreta) per mostrar totes les aplicacions actives.
3. Restaurar l'aplicació, si haviem fet clic al botó d'inici premem el botó de menú i seleccionem la nostra aplicació, o bé si havíem fet clic al menú, directament cliquem a l'aplicació.
4. Tornem a fer clic al botó de menú.
5. Arrosseguem l'aplicació cap amunt, per tal de tancar-la.

Podem vore aquet procés i les diferents eixides del log a la següent imatge:

[1] Recordeu que la barra de navegació d'Android és la barra que apareix a sota les aplicacions, i en la que tenim disponibles tres botons: Enrere, inici i menú.

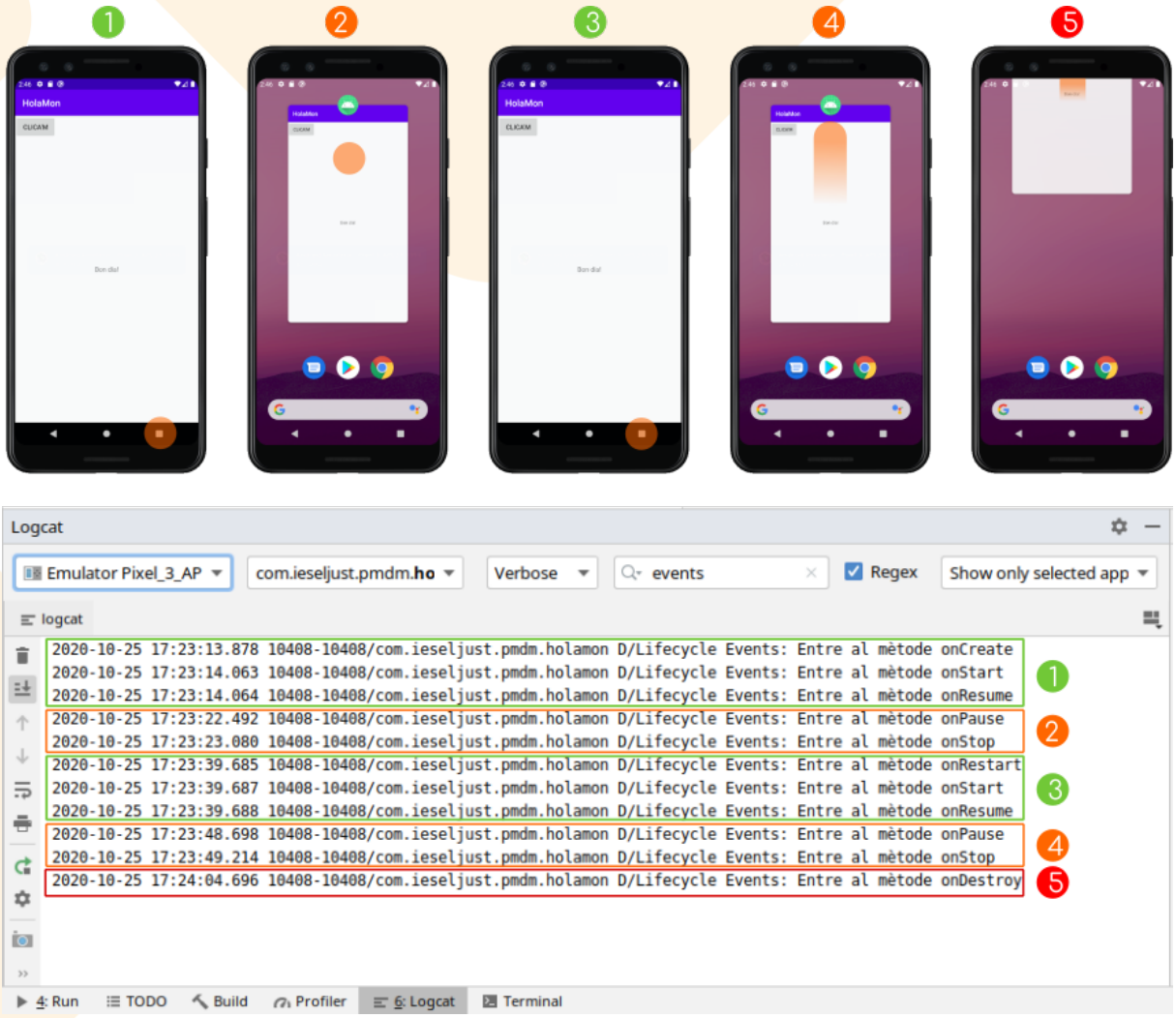

Figura 3: Registrant els canvis d'estat