

# Resum per a l'ús propi de Juanjo


## Índex

1. Definicions
2. Toast
3. Getion básica de vistas o componentes visuales
4. Activities
  1. Cicle de vida d'una Activity
  2. Intents
5. Requerir permisos a l'usuari
6. Intent Filter
7. Treballant amb Parcelables

## Tema 3 . Components d'una aplicació Android

### Definicions

- **Aplicació:** Finestra de components que poden interactuar entre ells.
- **Context:** Interfície entre l'aplicació i el sistema operatiu, la qual descriu la informació que representa la teua aplicació dins de l'ambient del sistema operatiu, accés a recursos i interacció con la resta de aplicacions. Es representa amb la classe abstracta `Context` . Tenim dues tipus:
  - **Context d'aplicació.** Aquest context engloba a tots els altres i per tant es únic, i cobreix tot el cicle de vida de l'aplicació.

 **Important:** Es pot accedir des d'una `Activity` o un `Service` amb `application` o des de qualsevol que herete de `Context` amb `applicationContext` .
  - **Context de 'Activity' o 'Service'.** Viu tant com l'element al qual pertany. Així doncs, un `Context` d'una `Activity` viurà el mateix temps que viu aquesta la activity i sempre viurà menys temps que el de l'Aplicació.
- **Components d'una aplicació**
  - **Activity:** Les '*activities*' són els components visibles de l'aplicació, tenen una interfície d'usuari que permet interactuar amb elles.
  - **Services:** Un servei és una entitat que executa instruccions en segon pla sense que l'usuari el note en la interfície. Per exemple guardar la informació en la base de dades, escoltar música mentre s'executa l'aplicació, administrar connexions de xarxa, etc.
  - **Vista:** una vista és una classe base per a tots els elements de la interfície d'usuari. Per tant, cobreix moltes classes i conceptes diferents, inclosos Widgets, ViewGroups i Layouts. Hi ha una vista arrel adjunta a una instància de finestra que forma la base de la jerarquia de vistes. En general, la paraula Vista sol usar-se per a descriure elements de la interfície d'usuari en general, o per a referir-se a classes d'interfície d'usuari abstractes o bàsiques, com ViewGroups.
  - **Widget:** Hi ha diverses definicions per a aquest terme, però la majoria es refereix a un element d'interfície d'usuari "*llest per a usar*", ja siga un botó, ImageView, EditText, etc. Tindrem en compte que algunes persones consideren que els Widgets són elements d'interfície d'usuari complets (no abstractes). En altres paraules podem dir que són els controls de usuari.
  - **Fragment:** Els fragments (fragment) es poden entendre com a seccions o parts (habitualment reutilitzables) de la interfície d'usuari d'una aplicació. D'aquesta manera, una activitat podria contindre diversos fragments.
  - **Content provider:** Amb aquest component, qualsevol aplicació en Android pot emmagatzemar dades en un fitxer, en una base de dades SQLite o en qualsevol altre format que considere. A més, aquestes **dades poden ser compartits entre diferents aplicacions**. Per defecte, el API d'Android porta amb si *Content Providers* predefinits per a intercanviar informació d'àudio, vídeo, imatges, e informació personal. Però si volem una estructura personalitzada has de crear la teua pròpia subclasse heretada de la classe `ContentProvider` .
  - **Intents:** Un Intent **és un objecte de missatgeria** que pots usar per a sol·licitar una acció d'un altre component d'una app. Aquests dades s'organitzen en parelles de clau-valor.
  - **Broadcast receiver:** El '*receptors d'emissió*', **reaccionen a intents específics podent al seu torn executar una acció** o iniciant una activity o poden retornar un altre intent al sistema perquè seguísca l'execució. Són capaços de processar els missatges del sistema operatiu.

### Toast

Per a mostrar un `Toast`

```
Toast.makeText(  
    applicationContext,  
    "Text que mostrarà el toast",  
    Toast.LENGTH_SHORT  
) .show()
```

### Getion básica de vistas o componentes visuales

```
// Buscar el objecte associat.  
var vista = findViewById<TipoVista>(R.id.idVistaEnLayout)  
  
// Gestió de events  
vista.setOnClickListener {  
    // Accions al clicar  
}
```

## Activities

### Cicle de vida d'una Activity

- **Creació** `onCreate()` :  
Una activitat s'ha creat quan la seua estructura es troba en memòria, però aquesta **no és visible encara**. Quan l'usuari pressiona sobre la icona de l'aplicació en el seu dispositiu, el mètode `onCreate()` és executat immediatament per a carregar el layout de l'activitat principal en memòria.
- **Execució-Represa** `onStart()` `onResume()` :  
Després d'haver sigut carregada l'activitat s'executen en seqüència el mètode `onStart()` i `onResume()`. Encara que `onStart()` fa visible l'activitat, és `onResume()` qui li transfereix el focus perquè interaccue amb l'usuari.
- **Pausa** `onPause()` :  
Una activitat està en pausa quan es troba en la pantalla parcialment visible. Un exemple: *quan s'obrin diàlegs que prenen el focus superposant-se a l'activitat*. El mètode anomenat per a la transició cap a la pausa és `onPause()`.
- **Detenció** `onStop()` :  
Una activitat està detinguda quan no és visible en la pantalla, però encara es troba en memòria i en qualsevol moment pot ser represa. Quan una aplicació és enviada a segon pla s'executa el mètode `onStop()`. En reprendre l'activitat, es passa pel mètode `onRestart()` fins a arribar a l'estat d'execució i després al de represa.
- **Destrucció** `onDestroy()` :  
Quan l'activitat ja no existeix en memòria es troba en estat de destrucció. Abans de passar a destruir l'aplicació s'executa el mètode `onDestroy()`. És comú que la majoria d'activitats no implementen aquest mètode, llevat que hagen de destruir processos com a serveis en segon pla.

### Intents

- **Explicit:** S'especifica exactament el component a llançar. Se sol utilitzar quan es vol executar els **components interns d'una aplicació**. Per exemple, passem dades a un altra activitat definida en la nostra aplicació.

Possibles:

1. Iniciem la activitat B des de A sense dades:

```
// this es la és la activitat cridadora que hereta de Context
// també podríem passar applicationContext
var intento = Intent(applicationContext, ActivityB::class.java)
startActivity(intento)
```

2. Passem dades en forma de clau-valor a la activitat B:

```
// mitjançant una interfície fluida
var i = Intent(applicationContext, MainActivity::class.java)
    .putExtra("CLAU1", "VALOR2")
    .putExtra("CLAU2", "VALOR2")

// o també ...
var i = Intent(applicationContext, MainActivity::class.java).apply {
    putExtra("CLAU1", "VALOR2")
    putExtra("CLAU2", "VALOR2")
}

startActivity(intento)
```

El objecte `Intent` en la `Activity` de destí el tindrem referenciat amb la propietat `intent` (**No deferíem utilitzar aquest id**)

```
val? valor1 = intent?.getStringExtra("DAT01")
```

3. Passem dades i esperem resposta segon resultat del Activity llançat.

Hem de separar la resposta de la crida per que després d'acabar la activitat es destruirà.

- **Pas 1:** Crearem un objecte `ActivityResultLauncher` mitjançant el mètode...


```
public final <I, O> ActivityResultLauncher<I> registerForActivityResult(
    @NonNull ActivityResultContract<I, O> contract,
    @NonNull ActivityResultCallback<O> callback)
```

A on ...

- `ActivityResultContract` es un '**contracte**' que especifica que una activitat es pot cridar amb una entrada de tipus `I` i produir una eixida de tipus `O`. etc. Tot i que es poden crear **contractes personalitzats**, la API proporciona contractes predeterminats per a accions de intent bàsiques, com **prendre una foto, sol·licitar permisos**, etc.

Per exemple, `ActivityResultContracts.StartActivityForResult()` és un Intent que produirà un `ActivityResult` com a resultat.

- `ActivityResultCallback` es un SAM '*consumidor*' al que le arriba un objecte de tipus `O`, per a el nostre contracte un `ActivityResult` i fa una determinada acció sense retornar o '*produir*' rés. `result : ActivityResult -> void`

 **Important:** Los `registerForActivityResult` deuen sempre ser-se avanç del `onStart()` això és en el `onCreate()`

```

val activityResultLauncher =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
        when (result.resultCode) {

            // Si el resultat es correcte
            // aleshores tindrem la resposta en result.data
            Activity.RESULT_OK -> {
                Toast.makeText(
                    applicationContext,
                    result.data?.getStringExtra("DATORETURN"),
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
    }
}

```

- **Pas 2:** Crearem un **Intent** amb les dades y el nom de la **Activity** com sempre i després amb el objecte **activityResultLauncher** llaçarem la **Activity** passant el Intent al mètode **.launch(i : Intent)** del objecte.

```

findViewById<Button>(R.id.buttonID).setOnClickListener {
    openPostActivity.launch(
        Intent(applicationContext, ActivityToLaunch::class.java)
        .putExtra("DATO", "Dato que se manda")
    )
}

```

- **Pas 3:** Per tornar dades ho farem creant un nou **Intent** que posarà el estat adequat en el objecte **ActivityResult**

```

textView.setOnClickListener {
    // Creem el nou intent amb les dades.
    val intentoResultado = Intent().putExtra("DATORETURN", "Este es el dato")
    setResult(Activity.RESULT_OK, intentoResultado)
    // Finalitza el activity i torna al creador.
    finish()
}

```

Exemple de còdi.

```

private lateinit var lanzadorActividad2: ActivityResultLauncher<Intent>

// Los registros deben hacerse al crear la Activity.
fun registraRespuestasActividades() {
    lanzadorActividad2 = registraRespuestaActivity2()
}

fun registraRespuestaActivity2(): ActivityResultLauncher<Intent> {
    return registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            Toast.makeText(
                applicationContext,
                result.data?.getStringExtra("DATORETURN"),
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}

fun manejadoresEventos() {
    manejadoreClickBoton()
}

fun manejadoreClickBoton() {
    var boton = findViewById<Button>(R.id.button)
    boton.setOnClickListener {
        var edit = findViewById<EditText>(R.id.datos)
        var intento = Intent(this, Activity2::class.java)
        .putExtra("DATOS", edit.text.toString())
        lanzadorActividad2.launch(intento)
    }
}

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

    registraRespuestasActividades()
    manejadoresEventos();
}

```

- **Implicit:** Aquells que li pregunten al sistema quin servei o component és el més adequat per a realitzar la petició, és a dir, no especifica un component especialment. Sinó que volem fer i led dades per a fer-ho seguint els següents passos:
  1. Declarem el intent amb l'acció apropiada (ACTION\_VIEW, ACTION\_WEB\_SEARCH, etc).
  2. Adjuntem informació adicional necessària depenent de l'acció del intent.


3. Llancem el intent deixant que el sistema trobe l'activitat adequada.

```
// Obrir els contactes.
val intento = Intent(Intent.ACTION_VIEW, Uri.parse("content://contacts/people/"));
// resolveActivity si està disponible el que volem fer.
if (intento.resolveActivity(packageManager) != null)
    startActivity(intento);

// Altres exemples de intents típics:
// 1. Obrir una ubicació en maps
val intento = Intent(Intent.ACTION_VIEW, Uri.parse("geo:38.3619129, -0.4905111?z=18"));
// 2. Fer una trucada per telèfon
val intento = Intent(Intent.ACTION_CALL, Uri.parse("tel:676188432"));
```

Però des de el **API 30** per a que `intento.resolveActivity(packageManager)` ens diga si està disponible la activitat implícita que volem obrir hem de incloure al `manifest.xml` la etiqueta `<queries>` indicant que la nostra activitat voldrà preguntar per la activity o el servei a llançar mitjançant el intent. Per exemple, per a `Intent.ACTION_VIEW` posaríem.

```
<manifest>
...
    <queries>
        <intent>
            <action android:name="android.intent.action.WEB_SEARCH" />
        </intent>
    </queries>
</manifest>
```

 **Important:** A més, si el intent precisa de permisos com accedir a la càmera. Hauríem de especificar-los al manifest i avanç de llançar el intent fer una petició de sol·licitud del mateixos.

Per a executar un altra aplicació, no fa falta posar la una *'query'* però sí hem de conèixer el nom del paquet i el nom de la classe de la activitat que volem obrir.

 **Nota:** Els noms de paquet els podem consultar a la carpeta `data/data` dins del emmagatzematge del nostre dispositiu. Per exemple ...

```
val intent = Intent();
intent.setComponent(ComponentName("com.example.ejerciciosuelto2activity",
    "com.example.ejerciciosuelto2activity.MainActivity"))
if (intent.resolveActivity(packageManager) != null) {
    try {
        startActivity(intent)
    } catch (e: ActivityNotFoundException) {
        Toast.makeText(this,
            "No existe la actividad aejerciciosuelto2activity",
            Toast.LENGTH_LONG).show()
    }
}
```

## Requerir permisos a l'usuari

Tenim que seguir els següents passos ...

- **Pas 1:** En l'arxiu de **manifest** de l'app, declara els permisos que necessites.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
</manifest>
```

- **Pas 2:** Esperar que l'usuari invoque la tasca o acció de l'app que requereix accés.

```

1 // Id que identifica la sol·licitut de permisos requerits al sistema.
2 private val RESPUESTA_PERMISOS = 111
3
4 @RequiresApi(Build.VERSION_CODES.Q)
5 fun solicitarPermisos() {
6     // android.Manifest.permission.READ_EXTERNAL_STORAGE
7     val permisosNecesarios = listOf<String>(READ_EXTERNAL_STORAGE)
8     val permisosQueFaltan = mutableListOf<String>()
9
10    // Mirarem si la App ja té els permisos concedits.
11    for (permiso in permisosNecesarios) {
12        if (checkSelfPermission(permiso) == PackageManager.PERMISSION_DENIED) {
13            // Si no els té concedit l'afegirem per ha sol·licitar-ho
14            permisosQueFaltan.add(permiso)
15        }
16    }
17
18    // Es llama a al mètode que que demana el permisos per part del sistema
19    // i que al finalitzar executa el callback onRequestPermissionsResult
20    // Nota: Hem anat afegint tots els que necessitàvem
21    if (permisosQueFaltan.size > 0) {
22        requestPermissions(permisosQueFaltan.toTypedArray(), RESPUESTA_PERMISOS)
23    } else {
24        // Si ja els teniem tots concedits llançaríem la funcionalitat.
25        // lanzarFuncionalidadQueRequierePermisos()
26    }
27 }

```

- **Pas 3:** `requestPermissions` crida a `onRequestPermissionsResult` a on verificarem si l'usuari ja ha concedit el permís/sos de temps d'execució que requereix l'app.

```

override fun onRequestPermissionsResult(
    requestCode: Int,           // código 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 (requestCode) {
        // Si rebem resposta del Id de sol·licitut que hem fet nosaltres.
        RESPUESTA_PERMISOS -> {

            val permisosQueFaltan = mutableListOf<String>()
            for (i in 0..grantResults.size) {
                if (grantResults[i] == PackageManager.PERMISSION_DENIED) {
                    permisosQueFaltan.add(permissions[i])
                }
            }

            if (permisosQueFaltan.isEmpty()) {
                lanzarFuncionalidadQueRequierePermisos()
            } else {
                Toast.makeText(
                    applicationContext,
                    "Lo siento no se puede realizar la "
                        + "funcionalidad porque faltan los siguiente permisos: "
                        + permisosQueFaltan.joinToString(", "),
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
    }
}

```

## Intent Filter

Al manifest, les activitats i serveis poden declarar el tipus d'accions que poden dur a terme i els tipus de dades que poden gestionar.

Com s'ha explicat en el punt anterior, quan es crea un intent implícit el sistema Android busca el component apropiat que pugui resoldre la petició donada en el intent. La cerca la realitza analitzant els **intent-filter definits per a eixa activitat o servici** en el arxiu de manifest. Per exemple...

```

<intent-filter>
  <!-- La acció es mostrar dades -->
  <action android:name="android.intent.action.VIEW"/>
  <!--
    Indica de de a on es llaçada la activitat (launcher, altra app, altra activitat)
    En aquest cas DEFAULT indica que és la activitat que es llançarà
    per a intent implícits des de altres aplicacions.
  -->
  <category android:name="android.intent.category.DEFAULT"/>
  <!-- El tipus de dades que el intent manejarà -->
  <data android:mimeType="vnd.android.cursor.dir/person"/>
</intent-filter>

```

## Treballant amb Parcelables

Si volem enviar dades en forma de objecte des de una app a un altra mitjançant un Intent Implicit. Definirem una classe POJO, però està haurà de derivar de la **Interfície Parcelable**. Com és de suposar això ens obligarà a implementar una sèrie de mètodes i propietats. Això ens permetrà passar un objecte **contacte** mitjançant un **putExtra** a un intent.

Veiem com definir una classe amb dades de un **contacte** per a enviar-los d'una activity a una altra, **afegit-li el comportament** de **Parcelable**.

```
class Contacto() : Parcelable {
    var nombre: String?
    var telefono: String?
    var email: String?
    var foto: Bitmap?

    init {
        this.nombre = ""
        this.telefono = ""
        this.email = ""
        this.foto = null
    }

    //Constructor generado automáticamente
    constructor(parcel: Parcel) : this() {
        nombre = parcel.readString()
        telefono = parcel.readString()
        email = parcel.readString()
        foto = parcel.readParcelable(Bitmap::class.java.classLoader)
    }

    constructor(
        nombre: String?, telefono: String?,
        email: String?, foto: Bitmap?
    ) : this() {
        this.nombre = nombre
        this.telefono = telefono
        this.email = email
        this.foto = foto
    }

    //funciones generadas automáticamente
    override fun describeContents(): Int = 0

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(nombre)
        parcel.writeString(telefono)
        parcel.writeString(email)
        // Fixat que si tenim objectes complexes a mode de composició o agregació
        // també han de ser 'parceables' com si estiguérem serialitzant.
        parcel.writeParcelable(foto, flags)
    }

    // Clase creada automáticamente.
    // Un companion object és la forma que té Kotlin
    // de definir membres statics dins d'una classe.
    // Tots el elements d'aquest object seran static
    companion object CREATOR : Parcelable.Creator<Contacto> {
        override fun createFromParcel(parcel: Parcel): Contacto {
            return Contacto(parcel)
        }

        override fun newArray(size: Int): Array<Contacto?> {
            return arrayOfNulls(size)
        }
    }
}
```

Des de el MainActivity es podrien passar les dades amb **putExtra** i obtindre'ls al destí mitjançant el mètode **intent.getParcelableExtra("TAG")**

```

1 class MainActivity: AppCompatActivity() {
2     override fun onCreate(savedInstanceState: Bundle?) {
3         // Recuperem les dades dels diferents components per a omplir el
4         // objecte parcel.
5         button.setOnClickListener {
6             val c = Contacto(
7                 nombre.text.toString(),
8                 tlf.text.toString(),
9                 correo.text.toString(),
10                imagen)
11             val intent = Intent(applicationContext, ActivityDesti::class.java)
12                 .putExtra("CONTACTO", c)
13             startActivity(intent)}
14     }
15
16     class ActivityDesti: AppCompatActivity() {
17         ...
18         val c = intent.getParcelableExtra<Contacto>("CONTACTO");
19         ...
20     }

```