

Apunts

[Descarregar aquests apunts](#)

Tema 3 . Components d'una aplicació Android

Índex

1. [Introducció](#)
2. [Context d'una aplicació](#)
3. [Components d'una aplicació](#)
 1. [Activity](#)
 2. [Services](#)
 3. [View](#)
 4. [Fragment](#)
 5. [Widget](#)
 6. [Content provider](#)
 7. [Intents](#)
 8. [Broadcast receiver](#)
4. [Cicle de vida d'una Activity](#)
5. [Comunicació de Activities a través de Intents](#)
 1. [Requerir permisos a l'usuari](#)
6. [Intent-filter](#)
7. [Treballant amb Parcelables](#)

Introducció

En aquest tema estudiarem els **components d'una aplicació Android** i les seues respectives funcions, a més de la comunicació entre *'activities'*.

Per a qualsevol aplicació Android, en general, serà necessari construir una finestra en la qual s'aniran afegint components, cada component és un punt d'entrada pel qual el sistema o un usuari interactua amb la teua aplicació. Alguns components depenen d'uns altres.

El primer que farem serà comprendre que és el context d'una aplicació i com permet relacionar els components.

Després veurem el concepte de **Activitats, Intents, Serveis, BroadCast Receivers i Content Providers**.

Context d'una aplicació

El context d'una aplicació és una 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.

També permet accedir als recursos de l'aplicació i coordinar el funcionament dels blocs de l'aplicació.

El context representa tota la meta-informació sobre les relacions que té l'aplicació amb altres aplicacions o el sistema i podem implementar-lo a través de la **classe abstracta Context**.

En Android ens trobem amb els següents tipus de contextos:


- **Aplicació**. Aquest context engloba a tots els altres, i cobreix tot el cicle de vida de l'aplicació des que l'arranquem fins que mor. Pel que cada aplicació té **un únic context d'aplicació**. El Context de l'aplicació viu fins que s'acaba per complet l'aplicació (fins que mor, no fins que es pausa)
Es pot accedir des d'una **Activity** o un **Service** amb `application` o des de qualsevol que herete de Context amb `applicationContext`.
- **Activity o Service**. Com hem dit, un Context viu tant com l'element al qual pertany, per la qual cosa depèn del cicle de vida. Així, 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.

La majoria de les aplicacions permeten l'execució de diverses accions a través de l'existència d'una o més pantalles.

 **Example:** Una aplicació de missatges de text podrà tindre la llista de contactes que es mostra en una finestra, mitjançant el desplegament d'una segona finestra l'usuari pot escriure el missatge al contacte triat, i en una altra tercera pot repassar el seu historial de missatges enviats o rebuts.

Cadascuna d'aquestes finestres pot estar representada a través d'un component Activity, de manera que navegar d'una finestra a una altra implica llançar una activitat o dormir una altra.

Services

Un servei és una entitat que **executa instruccions en segon pla** sense que l'usuari el note en la interfície. Són molt utilitzats per a realitzar accions de llarga duració mentre les activitats mostren un altre tipus d'informació. Per exemple guardar la informació en la base de dades, escoltar música mentre s'executa l'aplicació, administrar connexions de xarxa, etc.

Els serveis també tenen un cicle de vida com les activitats, però aquest és més curt. Només es compren dels estats de Creació, Execució i Destrucció.

View

Les vistes (view) són els components bàsics amb els quals es construeix la interfície gràfica de l'aplicació, anàleg per exemple als controls de Java o .NET. D'inici, Android posa a la nostra disposició una gran quantitat de controls bàsics, com a quadres de text, botons, llistes desplegables o imatges, encara que també existeix la possibilitat d'estendre la funcionalitat d'aquests controls bàsics o crear els nostres propis controls personalitzats.

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 per a formar la interfície completa de l'aplicació, i addicionalment aquests fragments es podrien reutilitzar en diferents activitats o parts de l'aplicació.

Widget

Els widgets són elements visuals, normalment interactius, que poden mostrar-se en la pantalla principal del dispositiu Android i rebre actualitzacions periòdiques.

Permeten mostrar informació de l'aplicació a l'usuari directament sobre la pantalla principal.

Content provider

Amb el component **Content Provider**, 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**. Una classe que implemente el component Content Provider contindrà una sèrie de mètodes que permet emmagatzemar, recuperar, actualitzar i compartir les dades d'una aplicació.

Per defecte, el API d'Android porta amb si Content Providers predefinits per a intercanviar informació d'àudio, vídeo, imatges, i informació personal. Però si en algun moment desitges intercanviar informació amb 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. Els intents faciliten la comunicació entre components de diverses formes, des de la comunicació entre activitats, entre fragments, serveis, etc. En un punt posterior d'aquest tema s'explicaran més extensament.

Broadcast receiver

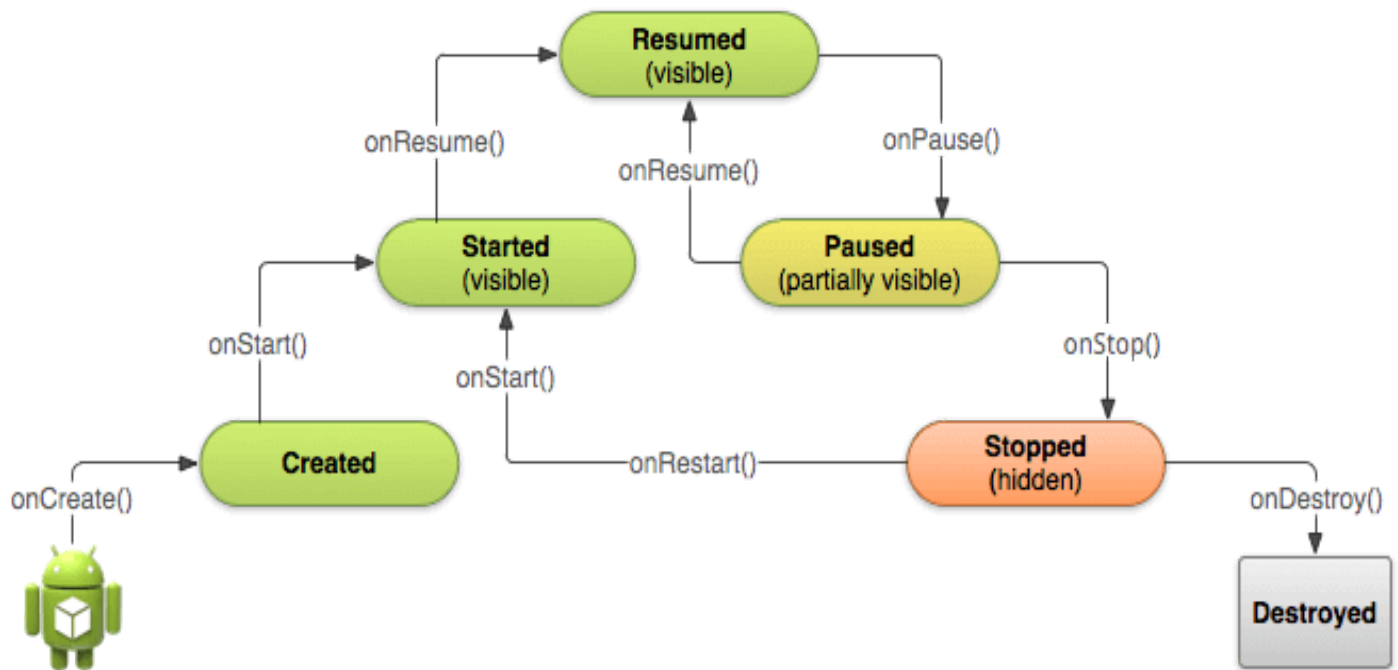
Finalment, l'últim component clau són els **Broadcast Receives**, o receptors d'emissió, que **reaccionen a intents específics podent al seu torn executar una acció** o iniciant una activity o poden retornar un altre intent al sistema perquè seguisca l'execució. Realment aquest tipus de components són capaços de processar els missatges del sistema operatiu.

Cicle de vida d'una Activity

Com hem comentat en el punt anterior. La activity és un dels components essencials d'una aplicació Android, a més que és el component que té associada la interfície d'usuari. Una mateixa aplicació pot tindre una o diverses activitats i Android permet controlar per complet el [cicle de vida](#) de cadascuna d'aquestes.

Els estats pels quals pot passar una Activity són els següents: **Creació, Execució, Represa, Pausa, Parada i Destrucció**.

A la relació entre ells se'n diu **Cicle de vida d'una Activitat**.



El cicle de vida d'una Activity ens descriu els estats i les transicions entre estats que una determinada Activity, com hem comentat són:

- **Creació:** una activitat s'ha creat quan la seua estructura es troba en memòria, però aquesta no és visible fins i tot. 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:** 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 interactue amb l'usuari.
- **Pausa:** 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:** 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:** 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.

Encara que la teua aplicació pot tindre diverses activitats en la seua estructura, s'ha de definir una activitat principal. Per a fer-ho s'ha d'especificar en el teu arxiu Android Manifest un component

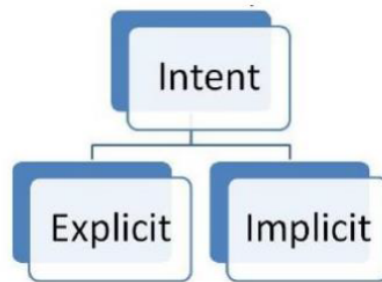
`<activity>` , amb un component fill `<intent-filter>` . Dins d'aquest component declarares dos nous components, `<action>` i `<category>` . Al primer li assignares l'element enumerat `MAIN` i al segon l'element enumerat `LAUNCHER` .

✍ **Crea un projecte nou i còpia el codi de `CicloVidaActivity` en la classe principal. Prova l'eixida que mostra Logcat en Debug i filtrat per "SALIDA". Fixa't perquè mètodes passa l'App segons els diferents canvis d'estat.**

Comunicació de Activities a través de Intents

Un dels components que hem comentat que formen part d'una aplicació android són els [intents](#). Com vam dir, són els que permeten la comunicació amb la resta de components, també es poden definir com la voluntat de realitzar una acció o una tasca determinada, aquestes tasques poden ser una trucada de telèfon o visualitzar una pàgina web entre altres.

Android suporta dos tipus d'Intents **explícits i implícits**.



Són explícits quan executem un component en específic. Són implícits quan s'entrega una referència genèrica sobre alguna condició, i aquelles entitats que complisquen aqueix requisit seran presentades com a candidates per a la tasca.

Intent explícit

S'especifica exactament el component a llançar. Se sol utilitzar quan es vol executar els components interns d'una aplicació.

🎓 **Exemple:** Un exemple de **Intent explicite** es dona quan invoquem una activitat amb una mena de classe específica. Per exemple, suposa que dins de la nostra aplicació tenim dues activitats anomenades **A** i **B** . Si decidim iniciar l'activitat **B** des de l'activitat **A** usaríem un **Intent** explícit pel fet que **B** és un tipus definit.



En l'inici d'una nova activitat es poden donar diverses circumstàncies:

- Inici d'una activitat sense arguments ni devolució de resultats:

```
var intento = Intent(this, Activity2::class.java)
//var intento =Intent(applicationContext, Activity2::class.java)
startActivity(intento)
```

- Inici d'una activitat amb arguments i sense devolució de resultats. Podem passar informació a les activitats afegint dades a l'objecte intent amb el mètode putExtra:

```
var intento = Intent(applicationContext, Activity2::class.java)
intento.putExtra("DATO", "Este es el valor que se manda desde Main Activity")
startActivity(intento)
```

En la classe invocada obtindrem les dades rebudes de la següent manera:

```
var intento = intent
findViewById<TextView>(R.id.textoActivity2).setText(intento.getStringExtra("DATO"))
```

- Inici d'una activitat amb arguments i devolució de resultats. Quan s'inicia una activitat per a obtenir un resultat, és possible (i quasi segur en el cas de les operacions que consumeixen molta memòria, com l'ús de la càmera) que es destruisca el teu procés i la teua activitat a causa de la poca memòria. Pel que s'ha de separar la devolució de crida de la resta de codi, per a això la llibreria d'Android ens proporciona diferents components:
 - **registerForActivityResult()** . **Línia 6** És l'element principal i que s'usa per a registrar la devolució de resultats. Té diverses sobrecarregues, al principi usarem la que pren un objecte **ActivityResultContract** . Retorna un element **ActivityResultLauncher** que s'usarà per a iniciar l'altra activitat.
 - **ActivityResultContract** . **Línia 6** i paràmetre de l'element anterior. Un contracte que especifica que una activitat es pot cridar amb una entrada de tipus I i produir una eixida de tipus O. La API proporciona **contractes predeterminats** per a accions de intent bàsiques, com prendre una foto, sol·licitar permisos, etc. També es poden crear contractes personalitzats.
 - **ActivityResultCallback** . **Línia 12** s'usa aquest objecte, creat en la línia 6. És una devolució de crida amb seguretat de tipus que es dirà quan el resultat d'una activitat

estiga disponible. És una interfície de mètode únic amb un mètode onActivityResult() que pren un objecte de la mena d'eixida definit en **ActivityResultContract**

- **ActivityResultLauncher** . **Línia 12** mana el intent que volem usar. És un llançador per a una crida preparada prèviament per a iniciar el procés d'execució d'un **ActivityResultContract** .

```
1 class MainActivity: AppCompatActivity() {
2     override fun onCreate(savedInstanceState: Bundle?) {
3         super.onCreate(savedInstanceState)
4         setContentView(R.layout.activity_main)
5         val openPostActivity =
6             registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
7                 result -> if (result.resultCode == Activity.RESULT_OK) {
8                     Toast.makeText(applicationContext, result.data?.
9                         getStringExtra("DATORETURN"), Toast.LENGTH_SHORT).show()
10                }
11            }
12        findViewById<TextView>(R.id.textoActivity1).setOnClickListener(View.OnClickListener {
13            openPostActivity.launch(
14                Intent(applicationContext, Activity2::class.java).apply
15                {putExtra("DATO", "Este es el valor que se manda desde Main Activity")})
16        })
17    }
```

🔥 **Nota:** De **Línia 6 a Línia 10**, es registra l'activitat per a esperar resultats d'altres '*activities*' o '*fragments*'. S'ha de registrar l'activitat per a possibles resultats en el mètode **onCreate()** . Si el resultat és OK realitzem el codi necessari, en aquest cas s'ha extret la informació del intent rebut **result.data** . De **Línia 12 a 16** es llança el intent de l'activitat que es vol iniciar, però amb l'objecte **ActivityResultCallback** creat en el bloc anterior.

En la classe invocada retornarem les dades de la següent manera:

```
1 textView.setOnClickListener {
2     val intentoResultado = Intent()
3     intentoResultado.putExtra("DATORETURN", "Este es el dato")
4     setResult(Activity.RESULT_OK, intentoResultado)
5     finish()
6 }
```

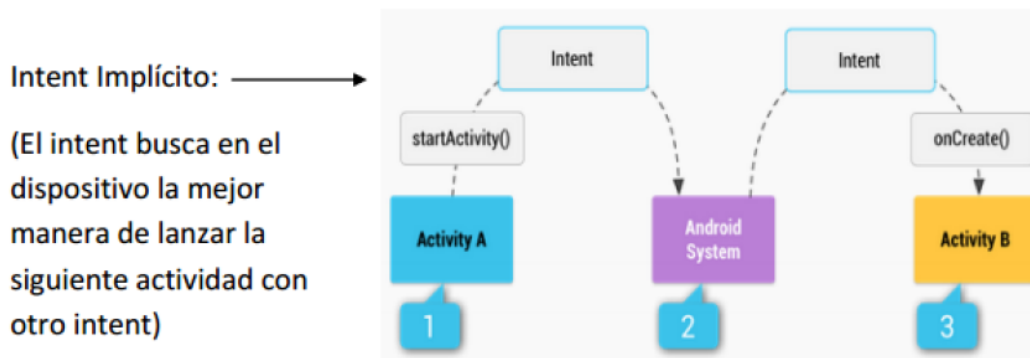
🔥 **Nota:** en la **Línia 4** preparem el resultat de l'activitat indicant que ha sigut correcte i afegint un intent amb informació si el creiem necessari. **Línia 5** tanquem la activity.

✍ **Ejercicio Resuelto2Activitys**

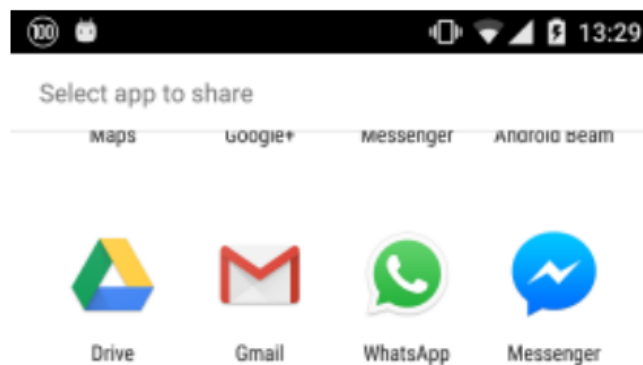
✍ **Ejercicio Propuesto3ActivitysRetornoDatos**

Intent Implícit

Els Intent implícits són 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, i només s'especifica la funcionalitat requerida a través d'una acció (veure, fer cerca, fer foto, marcar número...) i una dada (URL, número a marcar...) En aquest cas el sistema determina el component per a la seua resolució, desencadenant un "intent" sense conèixer exactament quina és l'aplicació o component que ho rebrà, si no pot establir exactament el component a utilitzar perquè hi ha més d'un, mostra una finestra amb les opcions a seleccionar.



Un exemple de **Intent implícit**: es quan desitges compartir un lloc web en alguna xarxa social. Per a això Android ens ofereix un dialogue llistant les app socials instal·lades en el dispositiu en aqueix moment, perquè triem la del nostre interès. No especifiquem que aplicació volíem, simplement es van mostrar totes les aplicacions que podrien respondre a aqueixa mena d'acció.



Bàsicament els passos per a llançar una activitat amb un **intent implícit** serien els següents:

- Declarem el intent amb l'acció apropiada (ACTION_VIEW, ACTION_WEB_SEARCH, etc).
- Adjuntem informació addicional necessària depenent de l'acció del intent.
- Llancem el intent deixant que el sistema trobe l'activitat adequada).

Per a iniciar l'activitat utilitzarem igualment startActivity. Alguns exemples de [Intents Comuns](#):

- Per a mostrar una web dins d'un navegador:

```
val intento = Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.es"));
startActivity(intento);
```

Requereix els permisos: `<uses-permission android:name="android.permission.INTERNET"/>`

- Per a utilitzar la cerca de google;

```
val intento = Intent(Intent.ACTION_WEB_SEARCH);
intento.putExtra(SearchManager.QUERY, "I.E.S. Doctor Balmis");
startActivity(intento);
```

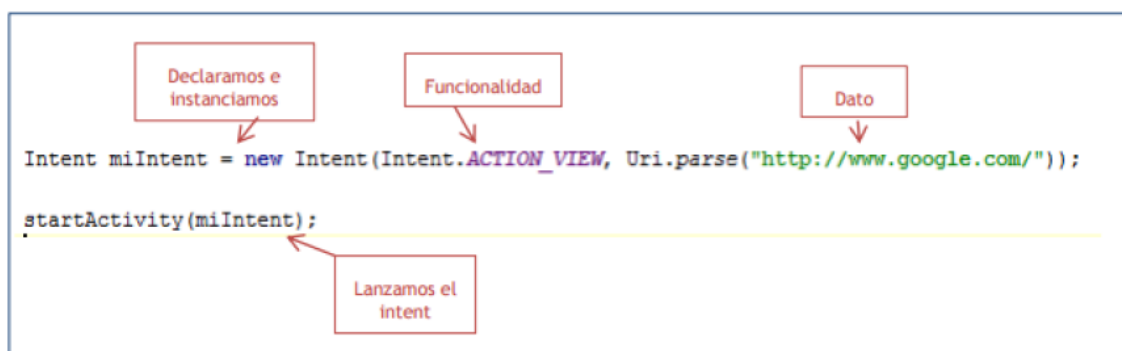
Requereix els permisos `<uses-permission android:name="android.permission.INTERNET"/>`

- Per a mostrar els contactes.

```
val intento = Intent(Intent.ACTION_VIEW, Uri.parse("content://contacts/people/"));
startActivity(intento);
```

Requereix els permisos:

`<uses-permission android:name="android.permission.READ_CONTACTS"/>`



És possible que a vegades no es pugui accedir a una aplicació pel fet que no està instal·lada en el dispositiu, a restriccions de perfil o a la configuració d'un administrador. En aquests casos, no es produeix la crida i es bloqueja l'app. Per a verificar que una activitat rebria la intent, és molt important usar abans `resolveActivity()` en l'objecte Intent. Si el resultat no és nul, hi ha almenys una aplicació que pot administrar la intent i és segur cridar a `startActivity()`.

```
val intento = Intent(Intent.ACTION_VIEW, Uri.parse("content://contacts/people/"));

if (intento.resolveActivity(packageManager) != null) startActivity(intento);
```

Ejercicio ResueltoIntentImplicito

Ejercicio PropuestoIntentImplicito (**Fer al Tema 5**)

Requerir permisos a l'usuari

Android està apostant cada vegada més fort per la seguretat, per la qual cosa en les versions més recents del sistema operatiu no n'hi ha prou amb afegir l'etiqueta `<uses-permission ...>` en el Manifest. Si es vol accedir a recursos aliens (ajenos) a l'aplicació en funcionament, s'haurà de realitzar una [sol·licitud de permisos](#) de forma explícita a l'usuari.

Això és pel fet que cada app s'executa en una zona de proves amb accés limitat, si necessita usar recursos o informació aliens a la seua pròpia zona de proves hauràs de seguir els següents passos:

1. En l'arxiu de manifest de l'app, declara els permisos que necessites.
2. Esperar que l'usuari invoque la tasca o acció de l'app que requereix accés a dades privades específiques. En aqueix moment, l'app pot sol·licitar el permís de temps d'execució necessari per a accedir a aqueixes dades. Per a això:
3. Verifica si l'usuari ja va atorgar el permís de temps d'execució que requereix l'app. En aquest cas, aquesta ja pot accedir als recursos necessaris.
4. En cas contrari, sol·licitar el permís en temps d'execució, en aqueix cas el sistema mostrarà un dialogue sol·licitant el permís i si és atorgat es passarà a l'accés dels recursos. En aquest pas es pot optar per mostrar un dialogue explicant perquè motiu són necessaris els recursos.
5. Si l'usuari rebutja la sol·licitud, l'app hauria de proporcionar una opció elegant a l'usuari.

Podem crear un codi semblant al següent, en el qual es demana permís per a accedir als contactes i a la memòria externa del dispositiu:

```
1  val RESPUESTA_PERMISOS = 111
2  @RequiresApi(Build.VERSION_CODES.Q)
3  fun solicitarPermisos()
4  {
5      if (checkSelfPermission(READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_DENIED
6          || checkSelfPermission(READ_CONTACTS) == PackageManager.PERMISSION_DENIED)
7      {
8          if (shouldShowRequestPermissionRationale(READ_EXTERNAL_STORAGE))
9              //Si se decide explicar los motivos de los permisos con similar a un dialogo
10         if (shouldShowRequestPermissionRationale(READ_CONTACTS))
11             //Si se decide explicar los motivos de los permisos con similar a un dialogo
12             //Con requestPermissions se pide al usuario que permita los permisos,
13             //en este caso dos
14             requestPermissions(arrayOf(READ_EXTERNAL_STORAGE, _CONTACTS), RESPUESTA_PERMISOS)
15         }
16         else lanzarFuncionalidadQueRequierePermisos()
17     }
```

```

1  override fun onRequestPermissionsResult(
2      requestCode: Int, //código de identificación del resultado
3      permissions: Array<out String>, //array con los nombres de los permisos
4      grantResults: IntArray //array de 0 y -1 (permitido, no permitido) en orden)
5  {
6      super.onRequestPermissionsResult(requestCode, permissions, grantResults)
7      when (requestCode) {
8          RESPUESTA_PERMISOS -> {
9              if (grantResults[0] == PackageManager.PERMISSION_GRANTED)
10                 Toast.makeText(
11                     applicationContext,
12                     permissions[0] + " Permiso concedido",
13                     Toast.LENGTH_SHORT
14                 ).show()
15                 if (grantResults[1] == PackageManager.PERMISSION_GRANTED)
16                     Toast.makeText(
17                         applicationContext,
18                         permissions[1] + " Permiso concedido",
19                         Toast.LENGTH_SHORT
20                     ).show()
21             }
22         }
23     }

```

📌 **Nota: Línia 1** creem una constant identificativa de la petició de permisos. Des de la **Línia 3 fins a la 17** creem el mètode encarregat de llançar els diàlegs amb la petició dels dos permisos (en aquest cas). **Línia 5** es comprova que encara no s'han concedit, i en aqueix cas es llança la petició **Línia 16**. Les **línies 8 i 10**, servien per a afegir alguna forma d'explicació a l'usuari. Si el permís ha sigut concedit amb anterioritat es passarà a llançar la funcionalitat que l'usa, **Línia 16**.

El fil que controla la resposta de l'usuari, es recull mitjançant el mètode sobreescrit **onRequestPermissionsResult**, a aquest mètode li arriba la informació necessària per a poder saber al fet que petició de permisos es refereix (usant la constant creada anteriorment **Línia 7**), el array amb els noms dels permisos i el resultat de si han sigut concedit o no. Amb aquesta informació passarem a llançar la funcionalitat que els requereix o a donar altres opcions a l'usuari.

Intent-filter

Les activitats i serveis poden declarar el tipus d'accions que poden dur a terme i els tipus de dades que poden gestionar. D'aquesta manera s'informa el sistema de la mena d'intencions implícites que pot atendre aquest component. Els intent-filter es defineixen en l'arxiu de manifest de l'aplicació dins de la activity corresponent.

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 en els arxius de manifest de les aplicacions instal·lades en el dispositiu. Si existeix coincidència llança aquesta aplicació. Si la coincidència és múltiple (diverses aplicacions poden donar resposta a aqueixa petició), el sistema mostra un quadre de diàleg amb la finalitat que l'usuari decidisca. A un Intent podem associar-li una acció, unes dades i una categoria.

Les activitats poden declarar el tipus d'accions que poden dur a terme i els tipus de dades que poden gestionar.

- Les accions són cadenes de text estàndard que descriuen el que l'activitat pot fer. Per exemple `android.intent.action.VIEW`, és una acció que indica que l'activitat pot mostrar dades a l'usuari.
- La mateixa activitat pot declarar el tipus de dades del qual s'ocupa, per exemple `vnd.android.cursor.dir/person`, indica que l'activitat manipula les dades de l'agenda*.
- També poden declarar una categoria, que bàsicament indica si l'activitat serà llançada des del llançador d'aplicacions `android.intent.category.DEFAULT`, des del menú d'una altra aplicació o directament des d'una altra activitat.

Un possible exemple del `AndroidManifest.xml` que complisca les anteriors condicions:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="vnd.android.cursor.dir/person"/>
</intent-filter>
```

Una activity pot tindre diversos intent-filter definits. Igualment cadascun d'aqueixos intent-filter pot tindre diverses accions definides.

Manifest exemple de diversos intent-filter:

```
<activity class = ".NotesList" android:label = "@string/title_notes_list" >
  <intent-filter>
    <action android:name = "android.intent.action.MAIN" />
    <category android:name = "android.intent.category.LAUNCHER" />
  </intent-filter>
  <intent-filter>
    <action android:name = "android.intent.action.VIEW" />
    <action android:name = "android.intent.action.EDIT" />
    <action android:name = "android.intent.action.PICK" />
    <category android:name = "android.intent.category.DEFAULT" />
    <data android:mimeType = "vnd.android.cursor.dir/ vnd.google.note " />
  </intent-filter>
  <intent-filter>
    <action android:name = "android.intent.action.GET_CONTENT" />
    <category android:name = "android.intent.category.DEFAULT" />
    <data android:mimeType = "vnd.android.cursor.item/ vnd.google.note " />
  </intent-filter>
</activity>
```

📌 **Nota:** El primer intent-filter defineix que aquesta activity és punt d'entrada per a l'aplicació. La categoria especifica que ha d'aparèixer en el llançador d'aplicacions. El segon, permet a la activity visualitzar i editar el directori de dades (a través de les actions VIEW i EDIT), o recuperar una entrada particular i retornar-la (a través de la action PICK). En data s'especifica el tipus de dades que es manegen. Com està definit amb la Categoria DEFAULT, estem indicant que atendra els intents implícits que sol·liciten altres aplicacions.

Treballant amb Parcelables

Quan estem desenvolupant en Android i necessitem enviar dades a una altra Activity ho podem realitzar mitjançant un **Intent i Bundle** però què succeeix si desitgem enviar un objecte per a economitzar una anomenada a la API. Android ens brinda les opcions de **Serializable** (pròpia de Java) i **Parcelable**, no obstant això aquesta última és fins a 10 vegades més ràpida pel que ens enfocarem en ella.

Parcelable és una interfície i podem implementar-la manual o mitjançant un plugin que, en les últimes versions d'Android Studio, ve inclòs per defecte.

Clarament haurem de crear la classe POJO, com es fa sempre, però està haurà de derivar de la **Interfície Parcelable**. Com és de suposar això ens obligarà a implementar una sèrie de mètodes i una variable CREATOR que hereta de Creator i que tindrà tots els seus membres estàtics gràcies el modificador **companion**.

Vegem un exemple amb dos activitys, una en la qual es poden introduir les dades d'un contacte i que en prémer un botó envie les dades existents en la pantalla a una segona activitat que mostrarà les dades de manera senzilla.

Classe **Contacte** que hereta de la **Interfície Parcelable** :

```
class Contacto() : Parcelable {
    var nombre: String?
    var telefono: String?
    var email: String?
    var foto: Bitmap?
    //Constructor generado automáticamente
    constructor(parcel: Parcel) : this() {
        nombre = parcel.readString()
        telefono = parcel.readString()
        email = parcel.readString()
        foto = parcel.readParcelable(Bitmap::class.java.classLoader)
    }
    init{
        this.nombre=""
        this.telefono=""
        this.email=""
        this.foto=null
    }
    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 {
        return 0
    }
    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(nombre)
        parcel.writeString(telefono)
        parcel.writeString(email)
        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)
        }
    }
}
```

Classe **MainActivity** que inicia l'activitat segona manant l'objecte parcel·lat.

```
1 class MainActivity : AppCompatActivity()
2 {
3     override fun onCreate(savedInstanceState: Bundle?) {
4         super.onCreate(savedInstanceState)
5         setContentView(R.layout.activity_main)
6         val nombre = findViewById<TextView>(R.id.campo_nombre)
7         val tlf = findViewById<TextView>(R.id.campo_tlf)
8         val correo = findViewById<TextView>(R.id.campo_correo)
9         val img = findViewById<ImageButton>(R.id.placeImage)
10        val imagen = BitmapFactory.decodeResource(resources, R.drawable.noimagen)
11        val button: Button = findViewById<Button>(R.id.boton)
12        button.setOnClickListener {
13            val c = Contacto(nombre.text.toString(), tlf.text.toString(),
14                            correo.text.toString(), imagen)
15            val intent = Intent(applicationContext, Activity2::class.java)
16            intent.putExtra("CONTACTO", c)
17            startActivity(intent)}
18    }
```

📌 **Línia 15** es crea el intent per a l'activitat 2 . **Línia 16** com es pot veure, encara que l'element que afegim al intent és un objecte de tipus Contacte, es pot fer amb **putExtra** pel fet que el contacte hereta de Parcelable.

En la **Activity2** recuperarem l'objecte passat, amb la **línia 6**.

```
1 class Activity2: AppCompatActivity() {
2     override fun onCreate(savedInstanceState: Bundle?) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.activity2);
5         //recuperar datos y visualizar
6         val c = this.intent.getParcelableExtra<Contacto>("CONTACTO");
7         val nombre = findViewById<TextView>(R.id.txt_nombre);
8         val tlf = findViewById<TextView>(R.id.txt_tlf);
9         val correo = findViewById<TextView>(R.id.txt_correo);
10        val img = findViewById<ImageButton>(R.id.placeImage);
11        if (c != null) {
12            nombre.setText(c.nombre)
13            tlf.setText(c.telefono)
14            correo.setText(c.email)
15            img.setImageBitmap(c.foto)}}
16    }
```

👉 **Importante:** No olvides declarar en el Manifest las dos actividades.

✍️ **Ejercicio PropuestoParcelabe**