

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

Toasts i diàlegs



Continguts

Toasts i Diàlegs	3
Toasts	3
Diàlegs	4
DialogFragment	7
Diàlegs i callbacks	10
Personalitzant els diàlegs	14

Toasts i Diàlegs

Fins ara hem vist com afegir botons i quadres de text als nostres dissenys. Anem a veure ara com afegir alguns elements emergents com diàlegs i toasts.

Toasts

Els *toasts* serveixen per proporcionar informació a l'usuari de manera simple, sobre una operació en una xicoteta finestra emergent. Aquesta finestra només ocuparà l'espai requerit per mostrar el missatge, i no bloquejarà l'activitat actual. Estos avisos desapareixen automàticament al cap d'un temps determinat.

Per tal de mostrar un toast, farem ús del mètode estàtic `makeText()` de la classe `Toast`, que ens retornarà l'objecte de tipus *toast*. Aquest mètode requereix de tres paràmetres: el *Context* de l'aplicació, el missatge de text, i el temps durant el qual serà visible l'avís. Una vegada hem inicialitzat el *Toast*, el mostrem amb `show()`.

Per exemple, a l'aplicació en què fem clic al missatge d'Hola món, anem a completar el codi del mètode `setOnClickListener` del text `textHola`:

```
binding.textHola.setOnClickListener{
    // Canviem el text
    missatge="Bona vesprada"
    binding.textHola.text=missatge;

    // I mostrem un "toast" per informar del canvi

    // Inicialitzem el text a mostrar

    val text = "Has canviat el missatge a "+missatge

    // Afegim la durada (agafant una constant de la pròpia classe Toast)
    val duration = Toast.LENGTH_SHORT
    // Creem el toast
    val toast = Toast.makeText(applicationContext, text, duration)
    // Mostrem el Toast
    toast.show()
}
```

Fixeu-vos que estem utilitzant com a primer argument `applicationContext`, que no hem definit en cap lloc. Es tracta d'una propietat de la pròpia classe `Activity`, a la que realment estem accedint

a través del mètode accessor `getApplicationContext()`.



Context

La classe `Context` és una classe abstracta implementada pel propi sistema Android, i que dóna accés a recursos i classes de la pròpia aplicació, així com suport a les crides per iniciar activitats o enviar i rebre *intents*. Així doncs, distingim dos tipus de contextos: el context de l'aplicació i el de l'activitat.

Teniu més informació a l'article [Understanding Context In Android Application](#)

En principi, per tal de mostrar el *Toast* en l'activitat actual, podem utilitzar tant l'atribut `applicationContext` o directament utilitzar `this`.

També podem simplificar el codi anterior i encadenar el `show` amb la creació del toast, estalviant-nos la variable:

```
Toast.makeText(applicationContext, text, duration).show()
```

Si volem que el *toast* es mostre a la part superior, fem ús del mètode `setGravity(constant_Gravity: int, desplaçament_x: int, desplaçament_y: int)`. Per exemple:

```
toast.setGravity(Gravity.TOP, 0, 0)
```



Documentació oficial

Disposeu de més informació i personalització dels toasts en: <https://developer.android.com/guide/topics/ui/notifiers/toasts#kotlin>

Diàlegs

Els diàlegs són finestres menudes que serveixen per donar avisos a l'usuari, confirmar accions o afegir informació addicional. Els diàlegs no ocupen tota la pantalla, però fan que no siga possible la interacció amb la pantalla principal fins que no es realitzi determinada acció (finestres *modals*).

Els diàlegs s'implementen a la classe *Dialog*, de la que deriven els tres tipus de diàlegs que utilitzarem:

- *AlertDialog*, per mostrar un missatge d'avís, amb un títol, un, dos o tres botons, una llista d'elements seleccionables, o bé un disseny personalitzat.

- `DatePickerDialog/TimePickerDialog`, per mostrar un diàleg per triar una data o una hora.

Veiem un xicotet exemple de com mostrar una alerta senzilla, per al que utilitzarem una instància de la classe interna `Builder` de l'`AlertDialog`:

```
fun showDialogAlertSimple() {
    var elMeuDialeg=AlertDialog.Builder(this) //també pot ser
        ↳ this@MainActivity
        // Afegim el títol amb
    elMeuDialeg.setTitle("Aquest és el títol del diàleg")
        // Afegim el missatge
    elMeuDialeg.setMessage("Aci va el text del missatge")
        // Diem si es pot tancar el diàleg fent clic a la part
        // opaca de darrere el diàleg (Cancelable=true)
        // o que s'haja de tancar fent clic en OK
        // o Cancel (Cancelable=false)
    elMeuDialeg.setCancelable(false)
        // Afegim el botó "Ok"
    elMeuDialeg.setPositiveButton(android.R.string.ok,
        DialogInterface.OnClickListener { dialog, which ->
            // Callback en forma de lambda per a quan es prem Ok
            // Per exemple, mostrar un toast
            val duration = Toast.LENGTH_SHORT
            Toast.makeText(applicationContext, "Click en Ok",
        ↳ duration).show()
            })
        // Afegim el botó "Cancel"
    elMeuDialeg.setNegativeButton(android.R.string.cancel,
        DialogInterface.OnClickListener { dialog, which ->
            // Callback en forma de lambda per a quan es prem Cancel·lar
            // Per exemple, mostrar un toast
            val duration = Toast.LENGTH_SHORT
            Toast.makeText(applicationContext, "Click en Cancel·lar",
        ↳ duration).show()
            })
    elMeuDialeg.show()
}
```

També podem expressar la construcció anterior fent ús d'una única línia:

```
AlertDialog.Builder(this)
    .setTitle("Aquest és el títol del diàleg")
    .setMessage("Aci va el text del missatge")
    .setCancelable(false)
    .setPositiveButton(android.R.string.ok,
        DialogInterface.OnClickListener { dialog, which ->
            val duration = Toast.LENGTH_SHORT
            Toast.makeText(applicationContext, "Click en Ok",
                duration).show()
        })
    .setNegativeButton(android.R.string.cancel,
        DialogInterface.OnClickListener { dialog, which ->
            val duration = Toast.LENGTH_SHORT
            Toast.makeText(applicationContext, "Click en Cancel·lar",
                duration).show()
        })
    .show()
}
```

Aquesta funció, ens demanarà afegir les dependències a `android.content.DialogInterface`? i `androidx.appcompat.app.AlertDialog`?

Veiem per parts com generem el diàleg:

- `AlertDialog.Builder(this)`: Estructura d'una classe anidada a l'`AlertDialog` que ens instancia un constructor (builder) per a un `AlertDialog`, seguint el patró de disseny builder.
- `.setTitle()` i `.setMessage()` estableixen el títol i el missatge del diàleg.
- `.setCancelable(true/false)` indica si es pot cancel·lar el diàleg fent clic a fora d'ell.

Per tal d'afegir els botons al diàleg fem ús dels mètodes `setPositiveButton`, `setNegativeButton`, i `setNeutralbutton`.

- `.setPositiveButton(Etiqueta, DialogInterface.OnClickListener { callback })`: Estableix el comportament i l'etiqueta del botó d'OK. El primer paràmetre que rep és l'etiqueta del botó, que pot ser qualsevol etiqueta, tot i que en aquest cas, fa ús de l'etiqueta dels recursos del sistema `android.R.OK`. Com a segon paràmetre, rep un *listener*, que no és més que el mètode `OnClickListener` de la interfície `DialogInterface`, que detecta el clic en el botó i invoca la funció que li passem com a expressió lambda, proporcionant-li el diàleg que ha rebut l'event (`dialog`) i l'element sobre el que s'ha fet clic (`which`).
- `.setNegativeButton(Etiqueta, DialogInterface.OnClickListener {`

`callback }`): Estableix el comportament i l'etiqueta del botó de cancel·lar. El seu funcionament és com el de `setPositiveButton`.



Amb aquest mètode hem creat una forma relativament senzilla de crear un diàleg d'alerta. Ara bé, en el moment en què es realitza un redibuixat de la pantalla, per exemple, quan girem aquesta, el diàleg desapareix, pel que haurem de buscar una manera més pràctica de mostrar els diàlegs.

DialogFragment

Tot i que de l'anterior forma hem creat un diàleg, la forma habitual de fer-ho és mitjançant un contenidor *DialogFragment*, amb el que se'ns proporcionaran els controls necessaris per crear i gestionar els diàlegs, grantint el control dels esdeveniments del cicle de vida. D'aquesta manera, el diàleg es manté quan l'usuari prem el botó de tornar enrere o gira la pantalla.

Per a això crearem el nostre diàleg com una nova classe:

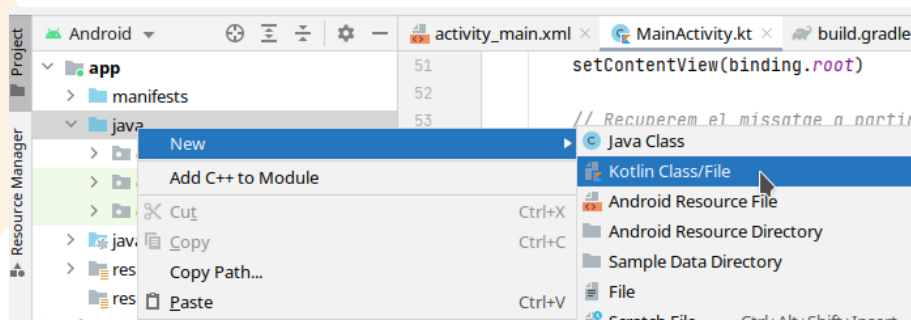


Figura 1: Creació d'una nova classe en Kotlin

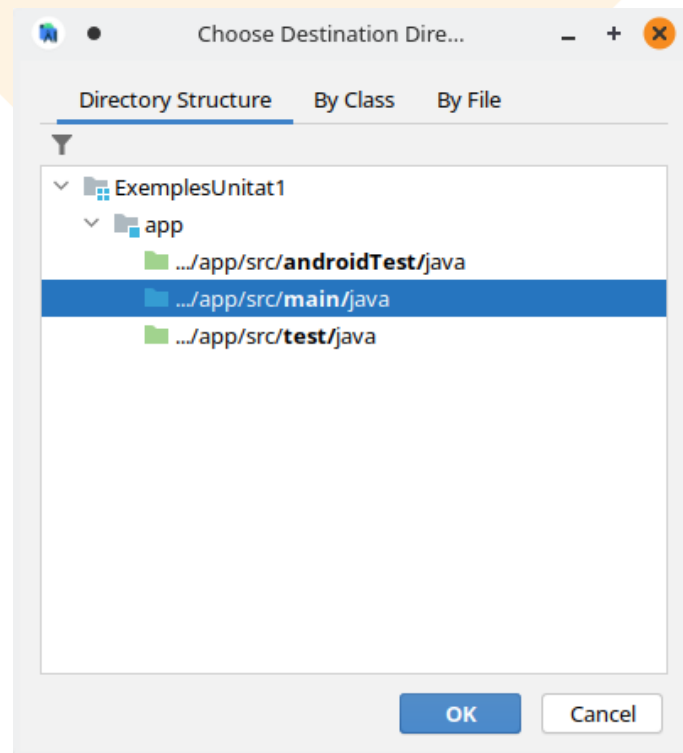


Figura 2: Creació d'una nova classe en Kotlin

Que serà subclasse de `DialogFragment`, a la que sobreescrivirem el mètode `onCreateDialog`:

```
class SimpleAlertDialog : DialogFragment() {
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return activity?.let {
            val title = "Aquest és el títol del diàleg"
            val content = "Aci va el text del missatge"
            val builder: AlertDialog.Builder =
                ↪ AlertDialog.Builder(requireActivity())
            builder.setTitle(title).setMessage(content)
                .setPositiveButton(android.R.string.ok) { _, _ ->
                    // Callback per al "Ok"
                    val duration = Toast.LENGTH_SHORT
                    Toast.makeText(requireActivity().applicationContext,
                    ↪ "Click en Cancel", duration).show()
                }
                .setNegativeButton(android.R.string.cancel) { _, _ ->
                    // Callback pe al Cancel
                }
        }
    }
}
```

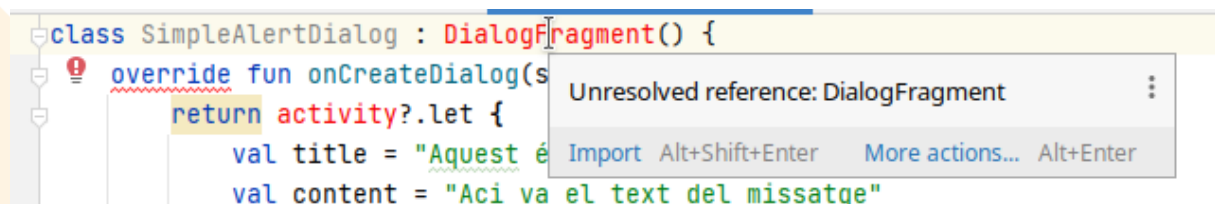


```

        val duration = Toast.LENGTH_SHORT
        Toast.makeText(requireActivity().applicationContext,
            ↪ "Click en Cancel", duration).show()
    }
    return builder.create() // Retorna un AlertDialog, tal i com
    ↪ l'hem definit al Builder
} ?: throw IllegalStateException("El fragement no pot ser nul")
}
}

```

Com veurem, la classe `DialogFragment` no la reconeix directament, però si passem el ratolí per damunt ens suggerirà importar-la:



```

class SimpleAlertDialog : DialogFragment() {
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return activity?.let {
            val title = "Aquest é
            val content = "Ací va el text del missatge"
        } ?: throw IllegalStateException("El fragment no està associat a cap activitat")
    }
}

```

Unresolved reference: DialogFragment

Import Alt+Shift+Enter More actions... Alt+Enter

Figura 3: Importació de la classe `DialogFragment`

De la mateixa manera, ens passa amb diverses classes, tals com `Bundle`, `AlertDialog` o `DialogFragment`.

Parem-nos a mirar un poc l'estructura per entendre-la millor:

```

class SimpleAlertDialog : DialogFragment() {
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return activity?.let {
            ...
            return builder.create()
        } ?: throw IllegalStateException("El fragment no està associat a cap
            ↪ activitat")
    }
}

```

Com veiem, l'estructura general consta d'una única ordre `return` del resultat de la funió d'àmbit `let`. Fixem-nos en la construcció:

- En primer lloc, obté l'activitat a la què està associada el fragment (amb `activity/getActivity()`).

- Si aquesta activitat no és nul·la, fa ús de la funció d'àmbit `let`, per crear dins aquesta activitat el diàleg.
- En cas que el fragment no estigui associat a cap activitat, es llançarà una excepció del tipus `IllegalStateException` (estat il·legal), fent ús per a això de l'operador Elvis (`? :`).

Expressant aquest codi sense fer ús de funcions d'àmbit, més a l'estil de Java, seria:

```
Dialog onCreateDialog(..) {  
    if (getActivity() != null) {  
        ...  
        return builder.create()  
    } else throw IllegalStateException("El fragment no està associat a cap  
        ↳ activitat")  
}
```

Per altra banda, adoneu-vos que ara, com que estem fora de l'activitat, no tenim accés directe al `Context`, i que hem d'obtenir aquest a partir de l'activitat a la què el fragment està associada:

```
Toast.makeText(requireActivity().applicationContext, "Click en ...",  
    ↳ duration).show()
```

Aquesta estructura és bastant comú en Android, ja que moltes vegades tenim elements que mai sabem si són nuls o no, i tant els contextos com els fragments poden ser-ho.

Per fer ús d'aquest diàleg que hem creat, fariem:

- Si s'invoca des d'una activitat:

```
val elMeuDialogModal = SimpleAlertDialog()  
elMeuDialogModal.show(supportFragmentManager, "confirmDialog")
```

- Si s'invoca des d'altre fragment: kotlin `val elMeuDialogModal = SimpleAlertDialog() elMeuDialogModal.show(requireActivity().supportFragmentManager, "confirmDialog")`

Diàlegs i callbacks

Ara bé, si volguérem, per exemple modificar el text del `TextView` només si fem click en OK, a través de la variable `missatge`, ens trobaríem amb el problema que no tenim accés a aquesta variable des del diàleg. En estos casos, voldrem enviar l'event a l'activitat o fragment que va obrir el diàleg.

Per tal de propagar un esdeveniment del diàleg a l'activitat, hem de fer ús de *Callbacks*, que caldrà especificar en la pròpia activitat. Per tal de fer això, el que fem és definir una interfície interna en el diàleg, de manera que l'activitat que vulga utilitzar-lo, haurà d'implementar aquesta interfície. Dins del mètode de gestió de cada clic, el que farem serà invocar els diferents mètodes de la interfície.

Així doncs, en primer lloc afegim la interfície amb els *callbacks* al `AlertDialog`, i després utilitzem els seus mètodes en la gestió d'esdeveniments:

```
class SimpleAlertDialog : AlertDialog() {  
    ...  
    interface OnContinueCancelClickListener {  
        fun onPositiveClick()  
        fun onCancelClick()  
    }  
    ...  
    builder.setTitle(title).setMessage(content)  
        .setPositiveButton(android.R.string.ok) { _, _ ->  
            val listener = activity as OnContinueCancelClickListener?  
            listener!!.onPositiveClick()  
        }  
        .setNegativeButton(android.R.string.cancel) { _, _ ->  
            val listener = activity as OnContinueCancelClickListener?  
            listener!!.onCancelClick()  
        }  
    ...  
}
```



Observeu que cal fer un càsting per tractar l'activitat a la que el diàleg està associada com a una classe que implementa la interfície `OnContinueCancelClickListener`:

```
val listener = activity as  
    OnContinueCancelClickListener?
```

Aquest càsting (amb `as`) es coneix com *Unsafe Cast Operator*, o operador de càsting no segur, el qual ens llançarà una excepció quan aquest càsting és impossible.

Documentació: <https://kotlinlang.org/docs/typecasts.html#unsafe-cast-operator>

I a l'activitat implementem la interfície junt amb els seus mètodes:

```
class MainActivity : AppCompatActivity(), OnContinueCancelClickListener {  
    ...  
    override fun onPositiveClick() {  
        ...  
    }  
  
    override fun onCancelClick() {  
        ...  
    }  
}
```

Veiem amb açò com quedaria la implementació completa de les dues classes:

```
class SimpleAlertDialog : DialogFragment() {  
  
    // Definim la interfície interna amb els callbacks  
    // la classe que utilitze aquest diàleg, haurà  
    // d'implementar-los  
    interface OnContinueCancelClickListener {  
        fun onPositiveClick()  
        fun onCancelClick()  
    }  
  
    // Sobreescrivim el mètode onCreateDialog  
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {  
        return activity?.let {  
            val title = "Confirmació"  
            val content = "Voleu canviar el missatge?"  
            // Creem l'objecte Builder per construir el diàleg  
            val builder: AlertDialog.Builder =  
                ↪ AlertDialog.Builder(requireActivity())  
            // Afegim el títol i el contingut, així com  
            // els callbacks per als dos botons.  
            builder.setTitle(title).setMessage(content)  
                .setPositiveButton(android.R.string.ok) { _, _ ->  
                    // Callback per al "Ok"  
                    // Definim el listener com a resultat del càsting de  
                    // l'activitat com una classe que implementarà  
                    // la interfície OnContinueCancelClickListener  
                    val listener = activity as OnContinueCancelClickListener?  
                    listener!!.onPositiveClick()  
                }  
        }  
    }  
}
```

```
        .setNegativeButton(android.R.string.cancel) { _, _ ->
            // Callback pe al "Cancel", igual que amb el "Ok"
            val listener = activity as OnContinueCancelClickListener?
            listener!!.onCancelClick()
        }
        return builder.create()
    } ?: throw IllegalStateException("El fragment no està associat a cap
        ↳ activitat")
}

// La nostra activitat implementa la interfície
// SimpleAlertDialog.OnContinueCancelClickListener
class MainActivity : AppCompatActivity(),
    SimpleAlertDialog.OnContinueCancelClickListener {

    var Missatge:String="";

    // I ara sobreescrivim els mètodes que hem definit
    // a la interfície que implementem.

    override fun onPositiveClick() {

        // Modifiquem el missatge
        Missatge="Hola què tal?";

        // I el bolquem a la interfície (cal fer de nou el binding)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.textHola.text=Missatge

        // I mostrem un toast
        val duration = Toast.LENGTH_SHORT
        Toast.makeText(applicationContext, "Text canviat a "+Missatge,
        ↳ duration).show()
    }

    override fun onCancelClick() {
        val duration = Toast.LENGTH_SHORT
        Toast.makeText(applicationContext, "Acció cancel·lada",
        ↳ duration).show()
    }
}
```

```

    }
    ...
}

```

Personalitzant els diàlegs

Els diàlegs ofereixen moltes possibilitats de personalització i configuració. A la documentació oficial podeu trobar descrites totes aquestes possibilitats. Ací anem a veure una xicoteta selecció d'aquestes.

- **Afegir una icona al diàleg:** Només cal que afegim la imatge als recursos, concretament a la carpeta `res/drawable` i quan creem el diàleg, fem ús del mètode `setIcon(R.drawable.nom_fitxer_sense_extensio)`

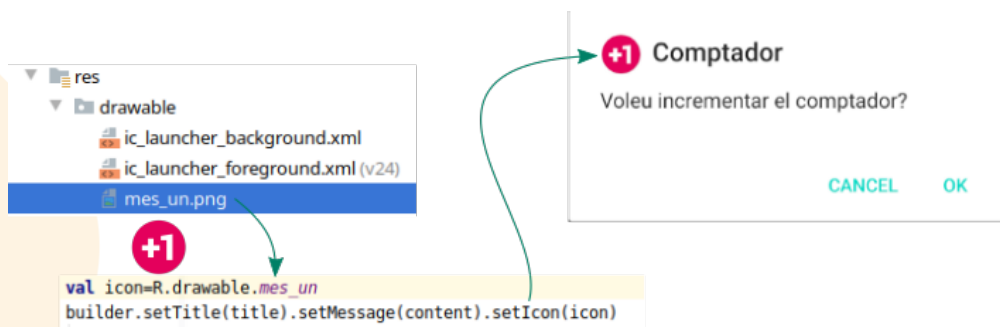


Figura 4: Afegint imatges als diàlegs

- **Afegir una llista d'opcions al diàleg:** Fem ús del mètode del *Builder* `setItems`, al que li passem un conjunt de valors. En el mètode de callback, el segon argument indicarà l'índex que s'ha seleccionat de la llista, i podrem utilitzar-lo al callback de la interfície:

```

class SimpleAlertDialog : DialogFragment() {

    interface OnContinueCancelClickListener {
        ...
        // Afegim un nou callback per a quan
        // es trie un element de la llista
        // Aquest rebrà un string del propi
        // callback del diàleg.
        fun onSelectSalutacio(salutacio:String)
    }

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {

```

```
return activity?.let {
    val title = "Tria una salutació"
    val builder: AlertDialog.Builder =
        ↪ AlertDialog.Builder(requireActivity())
    // Definim una llista mutable
    val salutacions: Array<String> = arrayOf<String>("Bon dia",
        ↪ "Bona vesprada", "Bona nit")
    builder.setTitle(title)
    builder.setItems(salutacions, { _, index_seleccionat ->
        val listener = activity as OnContinueCancelClickListener?
        // index_seleccionat conté la posició en el vector
        // de l'element seleccionat, és el que li passarem
        // al callback

        ↪ listener!!.onSelectSalutacio(salutacions[index_seleccionat])
            })
        ...
    }
}

class MainActivity : AppCompatActivity(),
    SimpleAlertDialog.OnContinueCancelClickListener {
    ...

    // Sobreescrivim el mètode onSelectNom de la interfície
    override fun onSelectSalutacio(salutacio: String) {
        // I mostrem al toast la salutacio
        val duration = Toast.LENGTH_SHORT
        Toast.makeText(applicationContext, "Has seleccionat
        ↪ "+salutacio, duration).show()
    }
    ...
}
```

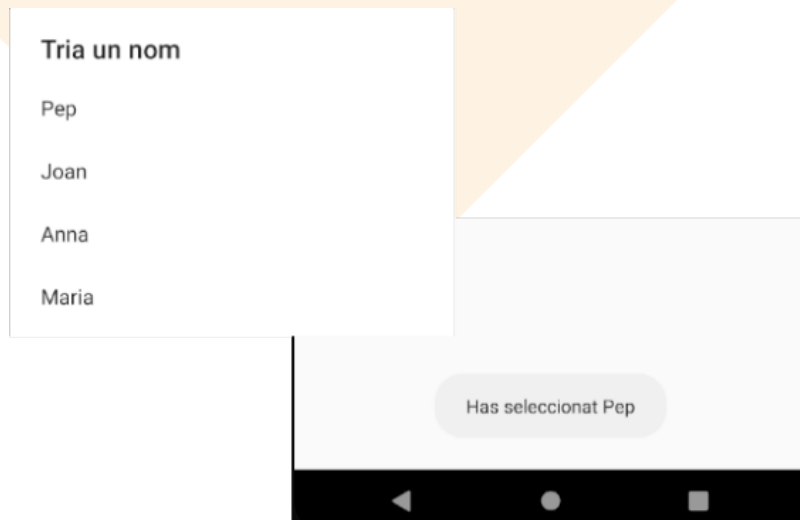


Figura 5: Exemple de llista en un dàleg

:::tip

Documentació oficial sobre Diàlegs

- <https://developer.android.com/guide/topics/ui/dialogs?hl=es#kotlin>
- <https://developer.android.com/reference/kotlin/android/app/AlertDialog.Builder>
- Cuadros de diálogo: <https://developer.android.com/guide/topics/ui/dialogs?hl=es-419>
DialogFragment: <https://developer.android.com/reference/androidx/fragment/app/DialogFragment?hl=es-419> Using DialogFragment: <https://guides.codepath.com/android/using-dialogfragment> >