Programació multimèdia
i dispositius mòbils

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

Toasts i diàlegs

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

Continguts

Toasts i Diàlegs

Fins ara hem vist com afegitr botons i quadres de text als nostres dissenys. Anem a vore 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 blocarà 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 realitze 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 misstge d'avís, amb un títol, 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 el mètode `Builder` de l'`AlertDialog`:

```
fun showDialogAlertSimple() {
    AlertDialog.Builder(this)
        // Afegim el títol amb
        .setTitle("Aquest és el títol del diàleg")
        // Afegim el missatge
        .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)
        .setCancelable(false)
        // Afegim el botó "Ok"
        .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"
        .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()
            })
        .show()
}
```

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

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, de manera que, per exemple, el diàleg no es tanque de forma predeterminada en rotar la vista.

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

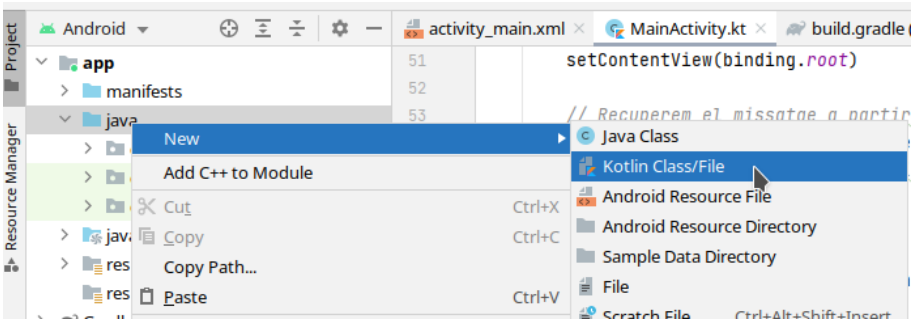

Figura 1: Creació d'una nova classe en Kotlin

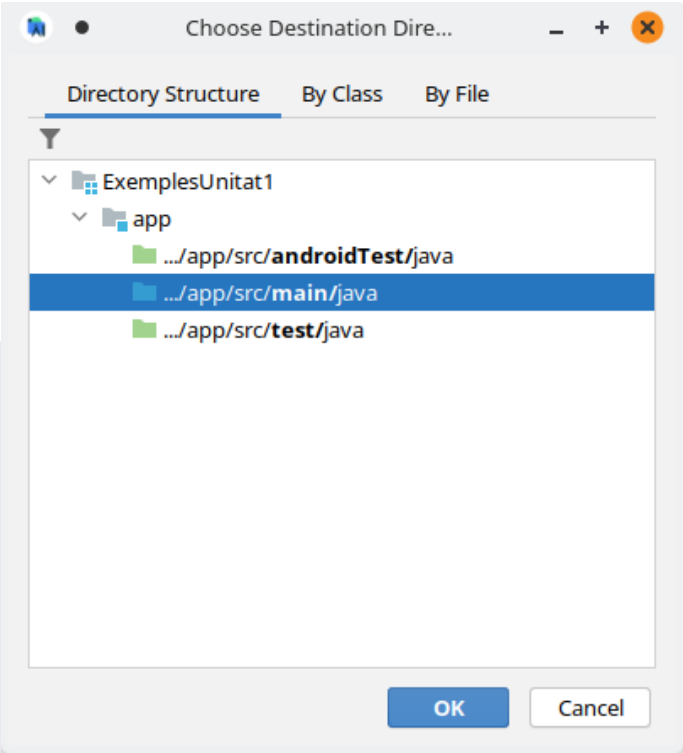

Figura 2: Creació d'una nova classe en Kotlin

Que serà subclasse de `DialogFragment`, a la que sobreescriurem 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()
        } ?: throw IllegalStateException("El fragement no pot ser nul")
    }

}
```

————————> Comentar millor val builder: AlertDialog.Builder = AlertDialog.Builder(requireActivity())

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

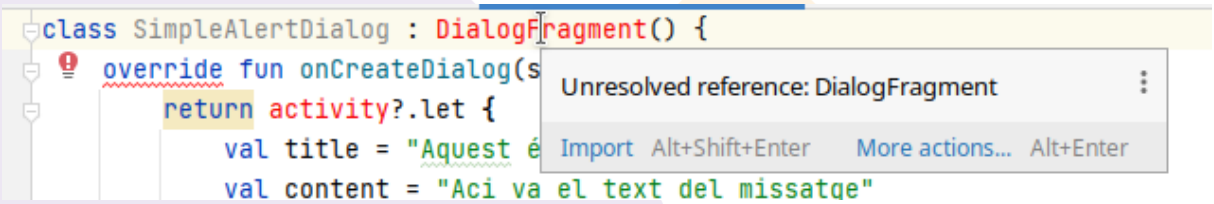

Figura 3: Importació de la classe DialogFragment

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 pot ser nul")
    }
}
```

Aquest codi introdueix una nova funcionalitat de Kotlin que no havíem vist, l'operador `?.let`, i l'operador *Elvis* (`?:`) que ja coneixíem. El primer, defineix un bloc *let* que s'executarà sempre que l'`activity` no siga nul·la. Per la seua banda, l'operador *Elvis* ens servia per especificar un valor alternatiu quan el resultat del bloc siga nul. El codi anterior, en pseudocodi podria ser:

```
funció onCreateDialog(..):Dialog{
    if (activity != null) {
        ...
        return builder.create()
    } else throw IllegalStateException("L'activitat no pot ser nulla")
}
```

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:

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

Aquesta estructura és bastant comú en Android, ja que moltes vegades tenim elements qua mai sabrem 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 elMeuDialegModal = SimpleAlertDialog()
elMeuDialegModal.show(supportFragmentManager, "confirmDialog")
```

- Si s'invoca des d'altre fragment: `kotlin val elMeuDialegModal = SimpleAlertDialog() elMeuDialegModal.show(requireActivity().supportFragmentManage "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.

// WIP > Canviant comptador per missaatge

Per tal de propagar un esdeveniment del diàleg a l'activitat, hem de fer ús de *Callbacks* per al retorn de l'esdeveniment, definint-los amb una interfície, i interceptar-los a l'*Activity* amb els corresponents *Listeners*.

Afegim la interfície amb els esdeveniments al `DialogFragment`:

```
class SimpleAlertDialog : DialogFragment() {
...
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()
    }
    ...
}
```

I a l'activitat afegim el Listener:

```
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 = "Comptador"
            val content = "Voleu incrementar el comptador?"
            // 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 una activitat 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("L'activitat no pot ser nulla")
    }

}

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

    var comptador: Int = 0

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

    override fun onPositiveClick() {
        val duration = Toast.LENGTH_SHORT
        comptador++ // Incrementem el comptador
        TextPrincipal.text = "Comptador: " + comptador
        Toast.makeText(applicationContext, "Comptador incremantat a
↪   "+comptador, duration).show()
    }

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

    // Definim el mètode onCreate de l'activitat
    override fun onCreate(savedInstanceState: Bundle?) {
        Log.d(tag, "Entre al mètode onCreate")
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        boto1.setOnClickListener { view: View ->
```

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

    }
...

}
```

Com veiem, es tracta pràcticament del mateix mecanisme que utilitzàvem per implementar els clicks sobre els botons amb la interfície `View.OnClickListener`.

Personalitzant els diàlegs

Els diàlegs ofereixen moltes possibilitats de personalització i configuració. A la documentació oficial podeu trobar descrites totes aquestes possibilitats. Aci 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)`

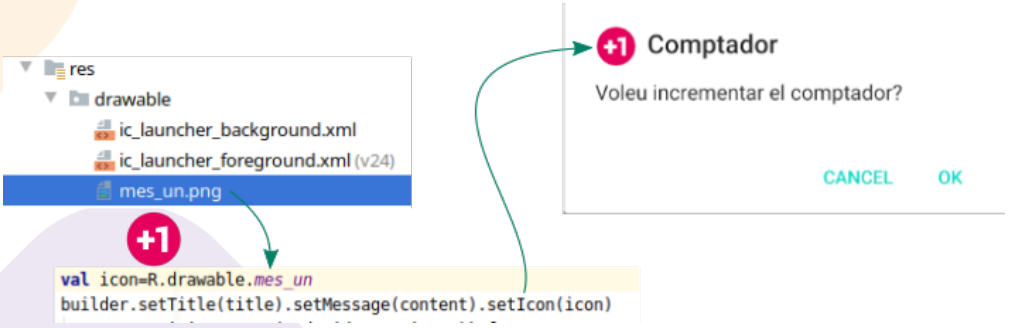

Figura 4: Afegint imatges als diàlegs

- **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()
    }
    ...
}
```

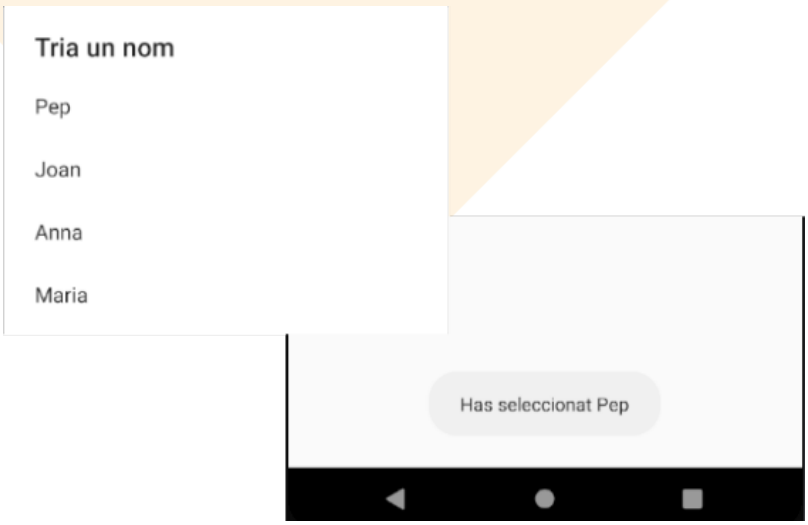

Figura 5: Exemple de llista en un dàleg

> Documentació oficial sobre Diàlegs
>
> https://developer.android.com/guide/topics/ui/dialogs?hl=es#kotlin