Software para Dispositivos Móviles

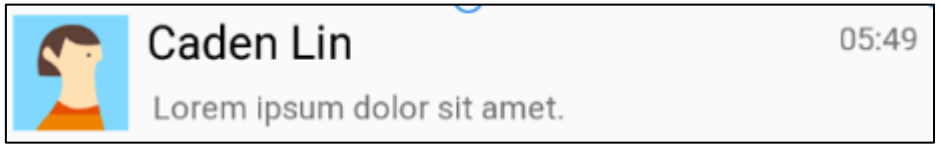
Miguel Sánchez Santillán
sanchezsmiguel@uniovi.es

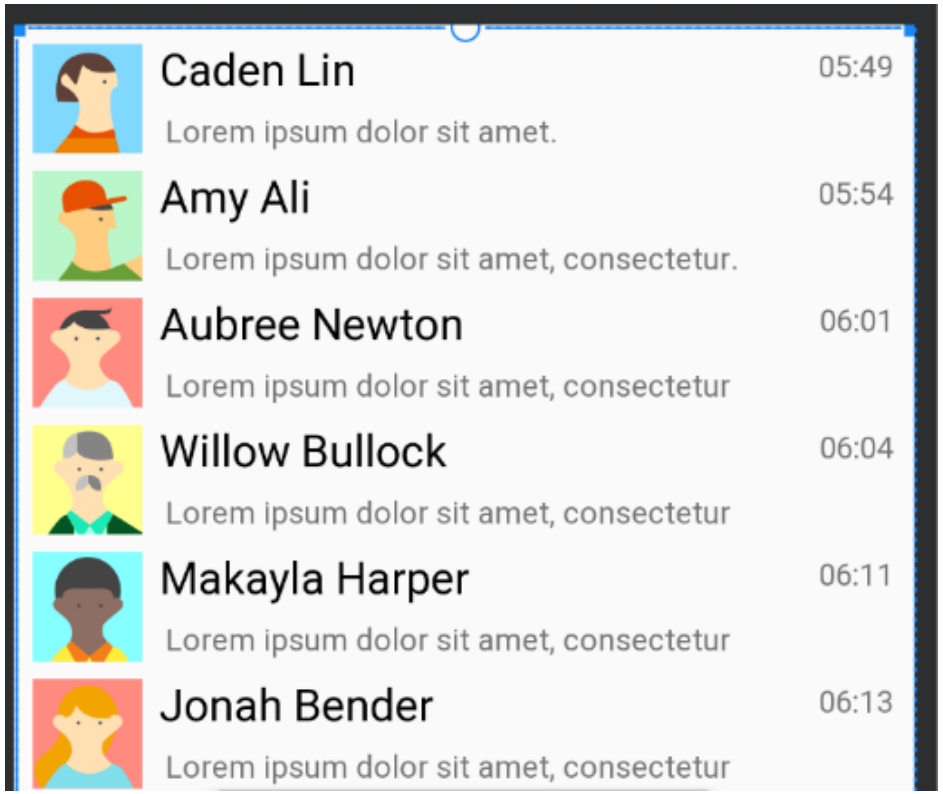
RecyclerView – Listas dinámicas

- Es un **ViewGroup** muestra colecciones de forma optimizada
- **Recicla** / Reutiliza los elementos visibles en pantalla
- Emplea **ViewHolder para representar** los elementos en pantalla
- Se basa en un **Adapter** para adecuarse al **tipo** de la colección
  - Relaciona datos de la colección con las vistas

RecyclerView – Representación gráfica

- Una colección de **contactos**
- Se muestran **6 en pantalla**
  - Pero hay otros 100, por ejemplo.
- **Vista** plantilla y la **adaptamos** a los datos





RecyclerView – Ejemplo básico con Strings

- Crea un proyecto vacío (empty **views** activity).

- Añadir en la vista un RecyclerView que ocupe el 100% del espacio
  - Asígnale una id

- En el MainActivity, crea un **atributo** para el Recycler
  - En el **onCreate**, **inicializa** el atributo

RecyclerView – LayoutManager

- **LinearLayoutManager**: Una dimensión
  - Este es el que usaremos hoy

    atributoRecycler.layoutManager = LinearLayoutManager(this)

- **GridLayoutManager**: Dos dimensiones
  - Mismos anchos y altos para una misma fila/columna

- **StaggeredGridLayoutManager**: Grid, pero al estilo *masonry*
  - Variabilidad de alto/ancho para una misma fila/columna

RecyclerView – Adapter

- Crea una clase TextosAdapter:

```
class TextosAdapter(private val listaTextos: List<String>) : RecyclerView.Adapter<TextosAdapter.ViewHolder>(){
```

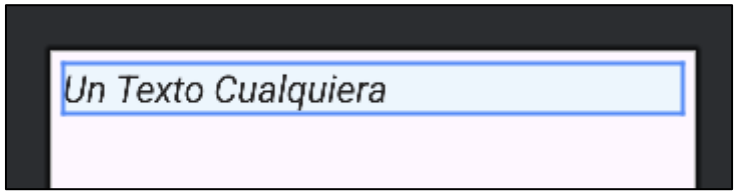
- Se indica que:
  - **Recibe y almacena** como atributo una lista de strings (la **colección**)
  - **Hereda de la clase abstracta** RecyclerView.Adapter genérica

- Dos errores:
  - Tres métodos no redefinidos (clase abstracta).
  - TextosAdapter.ViewHolder ← La clase ViewHolder no está definida

RecyclerView – Vista del elemento

- Crea un XML que represente la vista de un elemento del Recycler
  - Clic derecho en carpeta layout → New… → Layout Resource File

- Por ahora nada más **tendrá un TextView**
  - Un ejemplo:



RecyclerView – Clase ViewHolder

- Anida dentro del adapter la clase **ViewHolder**

- Representa la vista de un elemento
  - **Completa el método bind**

- **Fíjate** cómo se añade el **listener**

```
class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
    private var tvTexto : TextView = view.findViewById(R.id.tvTexto)
    private var texto : String? = null

    init {
        view.setOnClickListener {
            Log.d( tag: "PR3",  msg: "Clic en ${texto}")
        }
    }

    //Recibe un elemento de la colección
    //Gestiona toda la representación VISUAL del mismo
    fun bind(texto: String) {
        //Complétalo
    }
}
```

RecyclerView – Adapter y los tres métodos

- **getItemCount**: Devuelve el tamaño de la colección

- **onBindViewHolder**: Enlaza el elemento de la posición con el viewHolder.
  - Aquí se invoca al método bind

-
```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
    val layoutElemento = R.layout.texto_recycler
    //Se establece el layout de elementos visibles recyclerview
    val view = LayoutInflater.from(parent.context).inflate(layoutElemento, parent, attachToRoot: false)
    //Con esa información se genera un objeto ViewHolder
    return ViewHolder(view)
}
```

RecyclerView – Asignando el adapter

- Creamos una lista de 30 elementos y asignamos el adapter

```
recyclerTextos = findViewById(R.id.recyclerTextos)
recyclerTextos.layoutManager = LinearLayoutManager( context: this)
val listaTextos = List( size: 30) {i -> "Texto cualquiera $i"}
recyclerTextos.adapter = TextosAdapter(listaTextos)
```

- **Comprueba que todo funciona correctamente**

Ejercicio – Aplicación de Contactos

- Crea la clase **Contacto** :
  - **id** → Es un int para identificar al contacto
  - **nombre** → Es una cadena de texto
  - **telefono** → cadena de texto

- En un Intent sabemos **pasar** campos, pero no **objetos**…
  - Para eso existe la **interfaz Parcelable**
  - https://developer.android.com/reference/android/os/Parcelable

- En lugar de pasar 3 campos, pasamos el objeto directamente

Ejercicio – Parcelable automático con plugin

- El proceso se automatiza mediante la anotación @Parcelize
  - Funciona con objetos más o menos simples
  - https://developer.android.com/kotlin/parcelize

- En el fichero build.gradle.kts, añade el plugin y sincroniza

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    id("kotlin-parcelize") //nuevo
}
```

- La clase quedaría así:

```
@Parcelize
data class Contacto(val id: Int, val nombre: String, val telefono: String): Parcelable
```

Ejercicio – Lista de contactos

- En **MainActivity** genera una lista de contactos
  - Guárdala como atributo e inicialízala en el onCreate
- Puedes basarte en el código de la lista de textos
- Una cantidad suficiente para que exista scroll en el móvil

Ejercicio – MainActivity

- **MainActivity**: Mostrará el recyclerView de los contactos

- El diseño del contacto será algo similar a:
  - Ala izquierda una imagen fija. **Pregúntame por Coil** y los permisos de red
  - Justo después, el nombre del contacto

- Si haces clic en un contacto, se lanza la **DetallesActivity**

- Abajo a la derecha un Floating Action Button.Lanzará **NuevoActivity**

Ejercicio - DetallesActivity

- Muestra todos los detalles del contacto

- Incluye el botón para salir de la Activity

Ejercicio – NuevoActivity

- Permite añadir un nuevo contacto a la aplicación

- Presentará un formulario para los campos:
  - id → Numérico
  - Nombre → No vacío
  - Teléfono → No vacío

- Botón de Crear (RESULT_OK) y de Cancelar (RESULT_CANCEL)

Ejercicio – Insertando el contacto

- Una vez se reciba el contacto de NuevoActivity, no solamente sirve con añadirlo a la lista de contactos

- De alguna forma hay que **indicar al adapter que ha habido un cambio**

- La forma más rápida (y menos eficiente) es invocar al **método notifyDataSetChanged()** del **adapter**