
GALAXY
TRAINING

ANDROID DESDE 0

Sesión 3

Ing. Marco Estrella
Instructor en Tecnologías Java y Android
Github @jmarkstar

TEMAS A TRATAR

Rangos

Arrays y List

Generics

Tipos Función

RecyclerView

RecyclerView.ViewHolder

RecyclerView.Adapter

onClick() y onLongClick() en RecyclerView

onActivityForResult()

Enviar datos entre Activities

RANGOS

En kotlin se puede crear rangos de números de forma sencilla utilizando los siguientes operadores y funciones.

Operador '..'

Crea un rango cerrado, significa que incluye al valor inicial y final.

```
val unoATres = 1..3

for(n in unoATres){
    println(n)
}
```

Forma más sencilla de usar

```
for(n in 4..6){
    println(n)
}
```

Función rangeTo()

También crea un rango cerrado.

```
val unoAcuatro: IntRange = 1.rangeTo( other: 4)

for(n in unoAcuatro){
    println(n)
}
```

Forma más sencilla de usar

```
for(n in 5.rangeTo( other: 7)){
    println(n)
}
```

Función downTo()

Crea un rango desde un número mayor a un número menor..

```
val tresAuno: IntProgression = 3.downTo( to: 1)

for(n in tresAuno){
    println(n)
}

//Forma más sencilla de usar

for(n in 6 downTo 4){
    println(n)
}
```

Kotlin

RANGOS

Función step()

Crea un rango de números con saltos.

```
//Forma 1 de usar

for (i in 12.rangeTo( other: 20).step( step: 2))
    println(i)

//Forma 2 de usar

for (i in 0..10 step 2)
    println(i)
```

Función until()

Crea un rango de números pero no incluye al último.

```
//Forma 1 de usar
for(n in 8.until( to: 10)){
    println(n)
}

//Forma 2 de usar
for(n in 8 until 10){
    println(n)
}
```

Función reversed()

Invierte un rango.

```
val rango = 100..110

for (numero in rango.reversed()){
    println(numero)
}
```

Operador ‘in’

Verifica si un numero esta en un rango.

```
if(10 in 1..10){
    println("el número esta en el rango")
} else {
    println("el número no esta en el rango")
}
```

Kotlin

SOMOS PARTNER ORACLE

ARREGLOS

Nos permite almacenar un conjunto de datos del mismo tipo.

Podemos acceder a cada elemento mediante un **índice**.

Arreglo Uni-dimensional

Solo tiene de una dimensión

miArreglo[0]	miArreglo[1]	miArreglo[2]	miArreglo[3]	miArreglo[4]	miArreglo[5]
0	1	2	3	4	5

Arreglo Multi-dimensional

Tiene de dos dimensiones a más.

	miMatriz[0]	miMatriz[1]	miMatriz[2]
miMatriz[0]	5	8	1
miMatriz[1]	9	7	2
miMatriz[2]	10	15	25

SOMOS PARTNER ORACLE

Arreglo Uni-dimensional

Con la función **arrayOf ()**

No se agrega dato mediante un índice.

Sintaxis:

array : tipoArray = arrayOf(valor1,valor2)

```
val myArray : Array<Int> = arrayOf(1,2,3)
for (elemento : Int in myArray){
    print(" " + elemento)
}
println(" ; posicion 1 =" +myArray[1])
//Rpta: 1 2 3 ; posicion 1 = 2
```

Con el constructor **Array()**

Sintaxis:

- array : tipoArray =Array(tamaño){valor1,valor2}
  array[posición] = valor3

- array : tipoArray = Array(tamaño ,{valor1,valor2})

```
val array:Array<String> = Array( size: 3,{"España"})
    array[1]="Mexico"
    array[2]="Argentina"
for (elemento : String in array){
    print(" "+elemento)
}
print("\n")
print("El elemento de la posicion 1 = "+array[1])
//Rpta: España Mexico Argentina
// El elemento de la posición 1 = Mexico
```

Con **Colecciones**

```
val myArrayInt : IntArray = intArrayOf(1,2,3)
for (elemento : Int in myArrayInt){
    println(elemento)
}
```

```
val myArrayLong : LongArray = longArrayOf(10L,20L,60L)
for (elemento : Long in myArrayLong){
    println(elemento)
}
```

```
val myArrayDouble : DoubleArray = doubleArrayOf(1.5,2.8,3.3)
for (elemento : Double in myArrayDouble){
    println(elemento)
}
```

```
val myArrayFloat : FloatArray = floatArrayOf(1F,2F,3F)
for (elemento : Float in myArrayFloat){
    println(elemento)
}
```

SOMOS PARTNER ORACLE

Arreglo de dos dimensiones

Con el constructor **Array()** .

```
var num=10
val dosD : Array<IntArray> =Array( size: 4,{IntArray( size: 3)})
```

```
for (i : Int in 0..dosD.size-1){
    val columna =IntArray( size: 3)
    for (j : Int in 0..columna.size-1){
        columna[j]=num++
    }
    dosD[i]=columna
}
```

```
for (columnaArray : IntArray in dosD){
    for (j : Int in columnaArray){
        print(j)
        print(" ")
    }
    println("")
}
```

```
Rpta: 10 11 12
      13 14 15
      16 17 18
      19 20 21
```

Con la función **arrayOf() .**

```
val dosArray:Array<IntArray> = arrayOf(intArrayOf(1,2,3,4),intArrayOf(5,6,7,8))
```

```
for (i:IntArray in dosArray){
    for (j:Int in i){
        print(j)
        print(" ")
    }
    println("")
}
```

KOTLIN(TIPOS GENÉRICOS)

En este ejemplo de generics se ha creado una clase genérica llamada **Tienda**, este tiene un método llamado **cambiarItem()** que permite cambiar el item del tipo T.

El ejemplo se ha desarrollado con 2 clases(Ropa y Fruta).

Luego T es reemplazado por Ropa y Fruta.

De esta forma podemos tener una tienda de ropa y otra de Frutas.

```
class Tienda<T>(private var item: T){

    fun cambiarItem(item: T){
        this.item = item
    }

    fun mostrarItem(){
        println("El item es: $item")
    }
}

data class Ropa(var nombre: String, var marca: String, var stock: Int)

data class Fruta(var nombre: String, var precioKilo: Double)

fun main(args: Array<String>){

    val ropa1 = Ropa( nombre: "Polo", marca: "Nike", stock: 1000)

    val tiendaRopa = Tienda<Ropa>(ropa1)
    tiendaRopa.mostrarItem()

    val ropa2 = Ropa( nombre: "Pantalon", marca: "Adidas", stock: 1000)

    tiendaRopa.cambiarItem(ropa2)
    tiendaRopa.mostrarItem()

    println("-----------------------------------------------")

    val frutal = Fruta( nombre: "Papaya", precioKilo: 2.20)
    val tiendaFrutas = Tienda<Fruta>(frutal)
    tiendaFrutas.mostrarItem()
}
```

KOTLIN(TIPO FUNCIÓN)

Koltin soporta funciones como un tipo de dato. Significa que podemos guardar una función en una variable.

Funciones Normales

```
fun funcion(){
    println("Mi primera funcion")
}

fun funcionConParametros(x: Int, y: Int){
    println("Resultado: ${x+y}")
}

fun funcionConParamyReturn(x: Int, y: Int): Int{
    return x * y
}
fun main(args: Array<String>){

    funcion()

    funcionConParametros( x: 10,  y: 20)

    println(funcionConParamyReturn( x: 100,  y: 10))
}
```

Variables de tipo función

```
val suma: (Int, Int) -> Int = { x, y -> x + y }
val resta: (Int, Int) -> Int = { x, y -> x - y }
val division: (Int, Int) -> Int = { x, y -> x / y }
val multiplicacion: (Int, Int) -> Int = { x, y -> x * y }

fun aplicarOperacion(x: Int,
            y: Int,
            operacion: (Int, Int) -> Int): Int = operacion(x, y)

fun main(args: Array<String>){

    println("resultado suma: ${aplicarOperacion( x: 10,  y: 10, suma)}")
    println("resultado resta: ${aplicarOperacion( x: 12,  y: 10, resta)}")
    println("resultado division: ${aplicarOperacion( x: 100,  y: 10, division)}")
    println("resultado multiplicacion: ${aplicarOperacion( x: 10,  y: 10, multiplicacion)}")
}
```

INTRODUCCIÓN A ADAPTERS

Adapter

Un objeto adapter actúa como puente entre un AdapterView y los datos a mostrar.

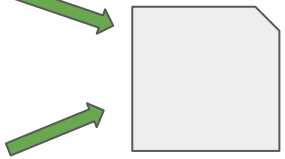
El adapter provee acceso a los elementos de los datos.

El adapter es responsable de hacer un View por cada ítem de la lista de datos.

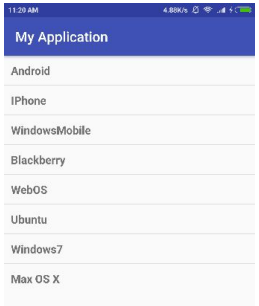
Modelo View para cada ítem

```
private val listaSO = arrayOf("Android", "IPhone",
                              "WindowsMobile", "Blackberry",
                              "WebOS", "Ubuntu", "Windows7",
                              "Max OS X")
```

Lista de objetos(datos)



Adapter



AdapterView

AdapterView

Un AdapterView es un View cuyo hijos están determinados por un Adapter.

Algunos AdapterView

- ~~ListView~~
- ~~GridView~~
- Spinner
- ~~Gallery~~
- RecyclerView

RECYCLERVIEW

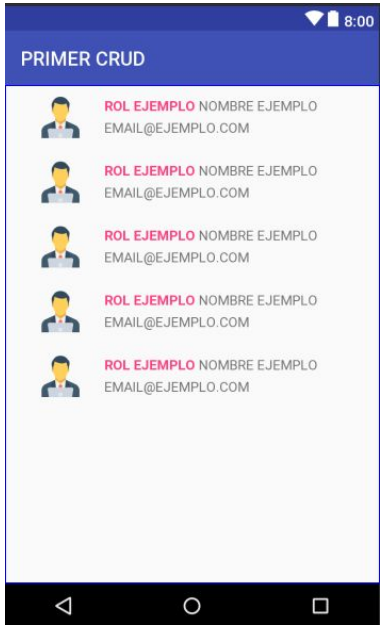
Es un View de tipo Lista.

Es un ViewGroup más avanzado y flexible que ListView.

No es nativo, tenemos que importar una librería.

Sus componentes principales son:

- Layout item
- ViewHolder
- Adapter



Pasos para implementar un RecyclerView

1- Importar la librería.

2- Agregar RecyclerView en el Layout que deseamos la lista.

3. Crear un Layout Item para la lista.

4. Crear un ViewHolder

5. Crear un Adapter

6. Asignar el Adapter al RecyclerView

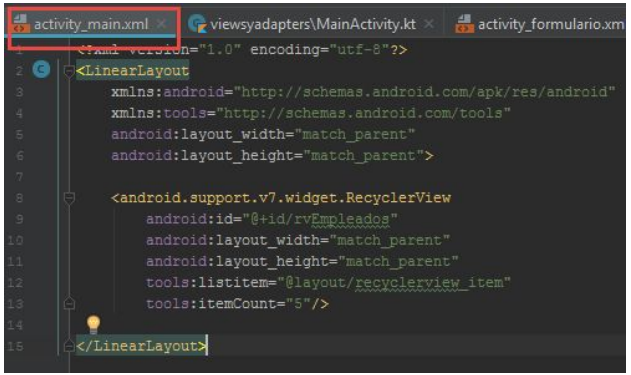
RECYCLERVIEW

SOMOS PARTNER ORACLE

Paso 1: Importar la librería

```
dependencies {
    implementation 'com.android.support:recyclerview-v7:27.1.1'
}
```

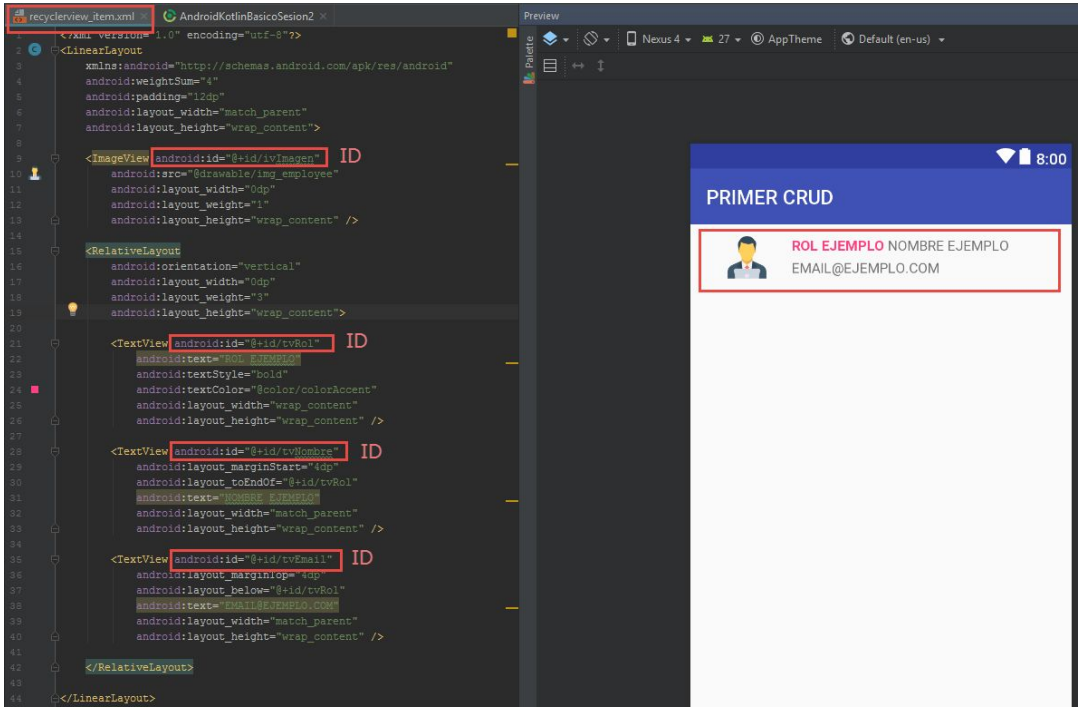
Paso 2: Agregar RecyclerView



Paso 3

Crear un Layout para los ítems del recyclerView.

Los componentes que serán dinámicos deberán tener un ID.



RECYCLERVIEW.VIEWHOLDER

SOMOS PARTNER ORACLE

ViewHolder

Describe un Item Layout.

Sirve para colocar en cache los componentes View del item Layout.

RecyclerView.ViewHolder

Es una clase abstracta que tenemos que extender cuando queremos crear nuestro propio ViewHolder para un RecyclerView.Adapter.

Usualmente es creado como una clase anidada dentro de la clase Adapter.

```
import android.support.v7.widget.RecyclerView
import android.view.View
import kotlinx.android.synthetic.main.recyclerview_item.view.*

class EmpleadoAdapter {

    class EmpleadoViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val ivFoto = itemView.ivImagen
        val tvNombre = itemView.tvNombre
        val tvRol = itemView.tvRol
        val tvEmail = itemView.tvEmail
    }
}
```

El ViewHolder debe contener todos los componentes View que van a ser dinámicos.

RECYCLERVIEW.ADAPTER

Cuando heredamos de la clase RecyclerView.Adapter estamos obligados a implementar 3 métodos.

OnCreateViewHolder(), aquí es donde inflamos el Layout Item en el ViewHolder.

getItemCount(), aquí retornamos la cantidad de items que tendra la lista.

onBindViewHolder(), este método se ejecuta dentro de un for, la cantidad de veces que tenga la lista de datos. Aquí es donde cargamos los valores de un ítem de dato en un Layout Item.

```
class EmpleadoAdapter: RecyclerView.Adapter<EmpleadoAdapter.EmpleadoViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): EmpleadoViewHolder {

    }

    override fun getItemCount(): Int {

    }

    override fun onBindViewHolder(holder: EmpleadoViewHolder, position: Int) {

    }

    class EmpleadoViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val ivFoto = itemView.ivImagen
        val tvNombre = itemView.tvNombre
        val tvRol = itemView.tvRol
        val tvEmail = itemView.tvEmail
    }
}
```

CARGAR EL ADAPTER EN EL RECYCLERVIEW

Creamos un método que retorne una lista de empleados.

Creamos el adapter y cargamos la lista en el adapter.

Finalmente cargamos el adapter en el RecyclerView.

```
class MainActivity : AppCompatActivity() {

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

        val empleados = obtenerEmpleados()

        val empleadoAdapter = EmpleadoAdapter(empleados)

        rvEmpleados.layoutManager = LinearLayoutManager( context: this)
        rvEmpleados.adapter = empleadoAdapter
    }

    private fun obtenerEmpleados(): ArrayList<Empleado>{
        val empleados = ArrayList<Empleado>()

        empleados.add(Empleado( rol: "JEFE", nombre: "JUAN PEREZ", email: "jperez@emp.com"))
        empleados.add(Empleado( rol: "GERENTE", nombre: "MARIA ROJAS", email: "mrojas@emp.com"))
        empleados.add(Empleado( rol: "ASISTENTE", nombre: "JORGE CHAVEZ", email: "jchavez@emp.com"))

        return empleados
    }
}
```

ONCLICK Y ONLONGCLICK

OnClickListener() es una interface que es invocado cuando un View es clicado

OnLongListener() es una interface que es invocado cuando un View ha sido clicado y retenido.

El objetivo es detectar estos eventos cuando el usuario hacer click o longclick a un item de la lista.

Para ello tenemos que pasar funciones al adaptador, ejecutar esas funciones cuando alguno de estos eventos sucedan y finalmente implementar esas funciones.

```
class ProductoAdapter(private val items: ArrayList<String>,
                      var onClick: ( (Int, String) -> Unit )? = null,
                      var onLongClick: ( (Int, String) -> Unit )? = null) : RecyclerView.Adapter<ProductoAdapter.ProductoViewHolder>() {
```

1

```
itemView.setOnClickListener { it: View!
    val onClickChecked = checkNotNull(onClick) { "OnClick no puede ser NULL" }
    onClickChecked.invoke(position, item)
}

itemView.setOnLongClickListener { it: View!
    val onLongClickChecked = requireNotNull(onLongClick)
    onLongClickChecked(position, item)
     ^setOnLongClickListener true
}
```

2

```
productoAdapterChecked.onClick = { position, item ->
    // codigo a ejecutar
}

productoAdapterChecked.onLongClick = { position, item ->
    // codigo a ejecutar
}
```

3

OnActivityForResult

Es usado cuando el activity a llamar retorna algún valor primer activity

Primer Activity

```
companion object {
    private const val REQUEST_CODE = 1000
}
```

1

```
val myIntent = Intent( packageContext: this, FormActivity::class.java)
startActivityForResult(myIntent, REQUEST_CODE)
```

2

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    if(requestCode == REQUEST_CODE){
        Log.v( tag: "main", msg: "REQUEST_CODE = $requestCode - resultCode = $resultCode")

        when(resultCode){
            100 -> {
               //codigo
            }
            200 -> {
                //codigo
            }
        }
    }
    super.onActivityResult(requestCode, resultCode, data)
}
```

3

Segundo Activity

Podemos retornar valores con un intent e invocando el método setResult()

```
if(accionChecked == Accion.REGISTRAR){
    Log.v( tag: "form", msg: "reg")

    val result = Intent()
    result.putExtra( name: "llave", value: "valor")
    setResult( resultCode: 100, result)

} else {
    Log.v( tag: "form", msg: "act")
    val result = Intent()
    result.putExtra( name: "llave", value: "valor")
    setResult( resultCode: 200, result)
}
```

Kotlin

Thank you!
Questions?