

Lambdas



1.

¿Qué es una
función Lambda?



Es una forma simple de definir una función anónima. Es posible usar una función como parámetro de otra.



2.

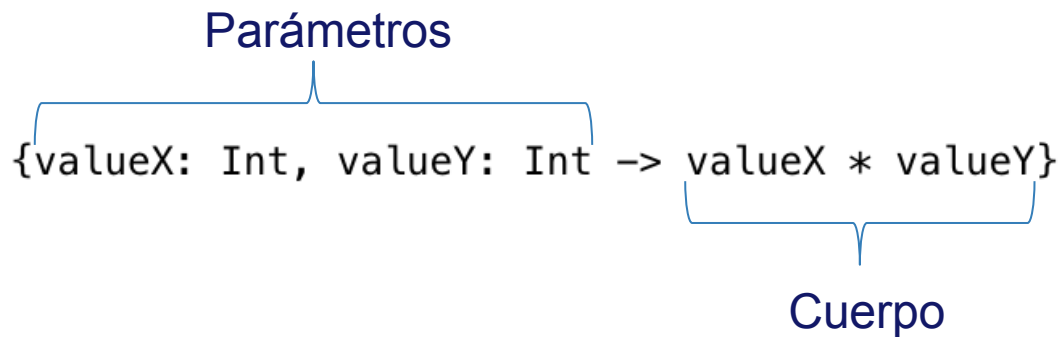
Sintáxis

- ▶ Siempre va encapsulado entre llaves
- ▶ La lista de parámetros no necesita ir entre paréntesis
- ▶ La flecha separa la lista de parámetros del cuerpo de la lambda

Parámetros

{valueX: Int, valueY: Int -> valueX * valueY}

Cuerpo



The diagram illustrates the components of a lambda expression. The text '{valueX: Int, valueY: Int -> valueX * valueY}' is shown. A blue bracket above the first part '{valueX: Int, valueY: Int}' is labeled 'Parámetros'. Another blue bracket below the second part 'valueX * valueY' is labeled 'Cuerpo'. The arrow '-' separates the parameters from the body.

3.

Inline y crossline

Inline y crossline

- ▶ Función ***Inline***. No consumen tantos recursos ya que en tiempo de compilación el compilador la sustituirá por el código y no creará clases anónimas.
- ▶ Función ***Crossline***. Se usa cuando una función será llamada desde otra lambda.



4.

Uso frecuente

Ejecutar tarea en segundo plano

```
private inline fun executeInBackground(crossinline function: () -> Unit) {  
    Thread({ function() }).start()  
}
```

Invocar a la función

```
executeInBackground { Log.e( tag: "Test", msg: "test" ) }
```

5.

Funciones de orden superior y lambdas (Higher- Order functions)



Se tratan de funciones que toman funciones como parámetros o retornan una función.

Control de excepciones





1.

Diferencias con Java

Diferencias

- ▶ No tiene checked exceptions
- ▶ *try* es una expresión, por lo que se puede usar para retornar valores
- ▶ *throw* también es una expresión por lo que se puede usar en las expresiones elvis



2.

Tipo *Nothing*

“

El tipo de la expresión *throw* es *Nothing*. Se usa para marcar una función que nunca retorna, por ejemplo que siempre acaba dando una excepción

Operador *when*





1.

Definición



Similar al *switch/case* pero mucho más potente y frecuente de ver en el código.

Tanto en el argumento como en los *case* se puede usar cualquier cosa. Un uso muy frecuente son los *smart cast*.



2.

Características

Características

- ▶ El caso por defecto se define con la estructura *e/se*
- ▶ Como es una expresión, puede retornar un resultado también
- ▶ Las condiciones pueden ser un conjunto de valores separados por comas

Características

- ▶ El argumento es automáticamente casteado en cada condición
- ▶ En las condiciones se pueden usar rangos
- ▶ Se puede usar para sustituir cadenas de *if / else*

Colecciones





1.

Métodos de creación

Tipos	Inmutable	Mutable
List	listOf	mutableListOf, arrayListOf
Set	setOf	mutableSetOf, hashSetOf, linkedSetOf, sortedSetOf
Map	mapOf	mutableMapOf, hashMapOf, linkedMapOf, sortedMapOf



2.

Interoperabilidad con Java



Al pasar una colección de Kotlin a Java, ésta puede ser editada e incluso se pueden introducir nulos en la misma



3.

Arrays de tipos primitivos



Kotlin proporciona clases específicas para los arrays de tipo primitivo: `IntArray`, `ByteArray`, `CharArray`, `BooleanArray`, etc.



4.

Formas de crear un Array

Formas de crear Arrays

- ▶ Pasándole sólo el tamaño del mismo.

```
val ints = IntArray(10)
```

- ▶ Pasando directamente el valor de cada elemento.

```
val ints: IntArray = intArrayOf(1, 2, 3)
```

- ▶ Pasando el tamaño y una lambda para inicializar cada elemento

```
val ints = IntArray(10) {i -> i*2}
```

5.

Operaciones comunes con colecciones

Operaciones

- ▶ ***Filter***. Filtra los elementos de una lista incluso eliminando los elementos pero no puede editarlos.
- ▶ ***Map***. Igual que *filter* pero ésta si puede editar los elementos.
- ▶ ***All***. Comprueba si todos los elementos cumplen o no un predicado.
- ▶ ***Fold***. Acumula empezando con el valor inicial y aplicando la operación pasada mediante una lambda para cada elemento.

Operaciones

- ▶ ***Any***. Comprueba si algún elemento cumple el predicado pasado por parámetro.
- ▶ ***Count***. Retorna el número de elementos que cumple el predicado.
- ▶ ***Find***. Retorna el primer elemento que cumple el predicado o ***null*** si no hay ninguno.
- ▶ ***Max o min***. Retorna el valor máximo o mínimo de un listado (***null*** si la lista está vacía).

Operaciones

- ▶ ***Partition***. Retorna dos listas, una con los elementos que cumplen el predicado y otra con los que no.
- ▶ ***ElementAtOrNull***. Retorna el elemento o *null* si el índice está fuera de rango.
- ▶ ***Sort*, *sortBy*, *reverse***, etc. Métodos de ordenación de los elementos.

Función de extensión



1.

¿De qué se trata?

“

Es una función que agrega un nuevo comportamiento a una clase y lo mejor... ¡Hasta sin tener acceso al código fuente!



2.

Ventajas

Ventajas

- ▶ Añadimos funciones extras a clases ya existentes
- ▶ No es necesario pasar el objeto como argumento
- ▶ Actúa como si perteneciese a la clase, por lo que podemos usar sus métodos públicos
- ▶ Pueden añadir también propiedades



3.

Estructura

Dos elementos:

- ▶ ***Receiver Type***. Clase a la que se añade la función
- ▶ ***Receiver Object***. Valor al que llama la función

Ejemplo:

Receiver Type

Receiver
Object

```
fun TextView.setTextAndHideIfIsNeeded(text: String) {  
    this.text = text  
    if (TextUtils.isEmpty(text)) {  
        this.visibility = View.GONE  
    } else {  
        this.visibility = View.VISIBLE  
    }  
}
```



4.

Imports

“

Deben ser importadas estáticamente una por una o utilizando el *.

5.

**¿Se pueden usar
desde Java?**

“

Sí, simplemente tenemos que usarla como si fuese un método estático y pasarle como argumento el objeto que la llama.

Declaración de clases, funciones y constructores





1.

Classes

“

Para definir una clases usamos la palabra clave *class*.

Si una clase implementa una interfaz usaremos “:”



2.

Herencia



Por defecto todas las clases son “finales” por ello, si queremos extender de ella ésta debe ser declarada como *open* o *abstract*.

Todas las clases extienden por defecto de *Any*



3.

Constructores

Lo básico

- ▶ Todas las clases tienen un constructor predeterminado por defecto
- ▶ Los tipos se escriben justo después del nombre
- ▶ No es necesario corchetes si la clase está vacía
- ▶ Declarar una estructura *init* si quieres implementar algo en el cuerpo de un constructor



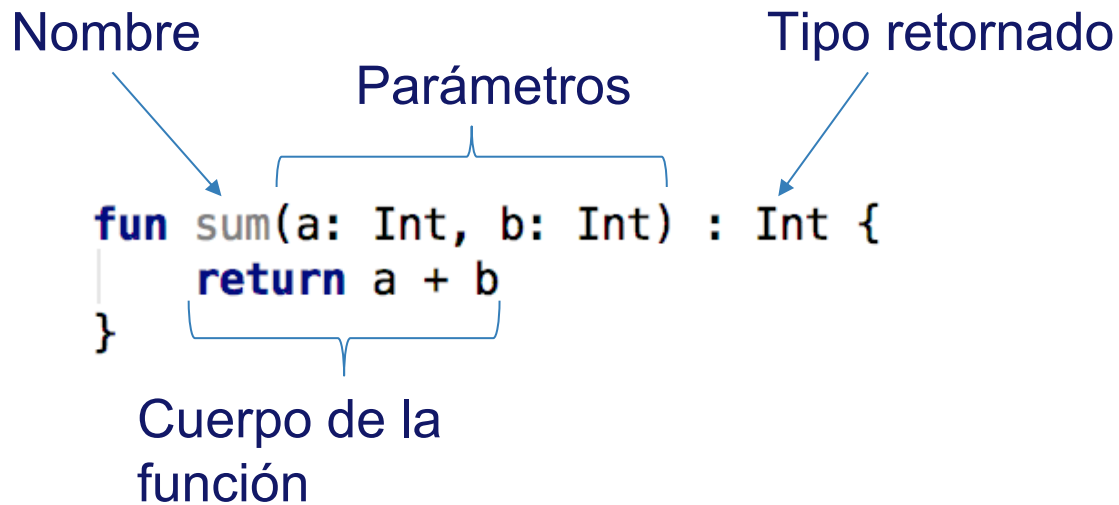
4.

Funciones

Lo básico

- ▶ Primero el nombre seguido de dos puntos y luego el tipo
- ▶ Podemos definir un valor por defecto a un parámetro para hacerlo opcional
- ▶ Al invocarlo, podemos usar el nombre del argumento que precede al valor
- ▶ Después del cierre de los parámetros iría el valor retornado precedido de dos puntos

Estructura





Introducción

1.

¿Qué es Kotlin?



Lenguaje de programación
estáticamente tipado que corre sobre
la Máquina Virtual de Java. Diseñado
para interoperar con código Java
aunque su sintaxis no sea
compatible.

A thick, light blue diagonal line runs from the top right corner towards the bottom left, separating the white background on the left from the solid blue background on the right.

2.

**¿Es compatible
con librerías ya
existentes?**

“

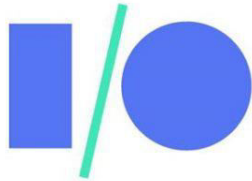
En resumidas cuentas, sí. Funciona con todas las bibliotecas y frameworks existentes en Java y se ejecuta con el mismo nivel de rendimiento.

3.

**¿Se trata de una
moda pasajera?**

“

No, la comunidad crece a una velocidad vertiginosa y ya se usa en muchos proyectos en producción.



Google I/O 2017



4.

¿Qué me aporta
Kotlin?



Simplicidad. Reduce en gran medida la cantidad de código por lo que:

A menor cantidad de código menor probabilidad de errores.



5.

Características básicas

Características básicas

- ▶ Conciso
- ▶ Seguro (¡adiós a los NPE!)
- ▶ Pragmático
- ▶ No penaliza el rendimiento
- ▶ Interoperabilidad con Java
- ▶ Programación funcional



6.

Referencias



Hasta llegar a la parte de Android, escribiremos y ejecutaremos nuestro código en <http://try.kotlinlang.org>

Libros:

- ▶ Kotlin in Action
- ▶ Kotlin for Android Developers

Kotlin para principiantes

¿Qué es Kotlin y por qué deberías aprender a utilizarlo?

¿QUÉ ES KOTLIN?

¿QUIÉN ERES, **KOTLIN**?

- Es un lenguaje de programación fuertemente tipado desarrollado por JetBrains.
- Ha sido influenciado por lenguajes como Groovy, Scala o C#.
- Permite generar código para la JVM (Java Virtual Machine), JavaScript y en las últimas versiones también ejecutables nativos.
- Tiene muchas menos características que Scala pero es mucho más familiar y tiene un mejor tiempo de compilación que este.



ORÍGENES

UN POCO DE HISTORIA...

- Kotlin fue creado en 2010 por JetBrains, la empresa detrás de IntelliJ IDEA, uno de los mejores IDE's de desarrollo para Java.
- Todos los IDE's de JetBrains están escritos en Java y su mayor problema era que su base de datos era muy grande porque Java es un lenguaje genérico y poco conciso y pensaron que características de lenguajes modernos podrían ayudarles.
- Su mejor opción era Scala pero lo descartaron por cuestiones de eficiencia y por ser demasiado potente para la solución que buscaban.

UN POCO DE HISTORIA...

- Una vez descartado Java, no querían salir del mundo JVM y decidieron que la mejor opción fue su propia versión mejorada de Java: Kotlin.
- Inicialmente fue creado para aplicaciones de escritorio (que es el mercado de JetBrains) pero ahora mismo es el multiplataforma además del lenguaje oficial de Android.



JAVA **VS.** KOTLIN

TODO ES MÁS FÁCIL CON KOTLIN

Java

```
final int x = // value;
final String xResult;

switch (x) {
    case 0:
    case 11:
        xResult = "0 or 11";
        break;
    case 1:
    case 2:
        //...
    case 10:
        xResult = "from 1 to 10";
        break;
    default:
        if (x < 12 && x > 14) {
            xResult = "not from 12 to 14";
            break;
        }

        if (isOdd(x)) {
            xResult = "is odd";
            break;
        }

        xResult = "otherwise"
}

final int y = //value;
final String yResult;

if(isNegative(y)) {
    yResult = "is zero";
} else if(isOdd(y)) {
    yResult = "is odd";
} else {
    yResult = "otherwise";
}
```

Kotlin

```
val x = //value
val xResult = when (x) {
    0, 11 -> "0 or 11"
    in 1..10 -> "from 1 to 10"
    !in 12..14 -> "not true from 12 to 14"
    else -> if (isOdd(x)) { "is odd" } else { "otherwise" }
}

val y = //value
val yResult = when {
    isNegative(y) -> "is negative"
    isZero(y) -> "is zero"
    isOdd(y) -> "is odd"
    else -> "otherwise"
}
```

Hello world

Java

```
public static void main(final String[] args) {
    System.out.println("Hello world!")
}
```

Kotlin

```
fun main() {
    println("Hello world!")
}
```

Variables I

Java

```
final int x;
final int y = 1;
```

Kotlin

```
val x: Int
val y = 1
```

Variables II

Java

```
int w;
int z = 2;
z = 3;
w = 1;
```

Kotlin

```
var w: Int
var z = 2
z = 3
w = 1
```


ADIÓS NULLPOINTEREXCEPTIONS

```
String value = null;

public void doSomeStuff() {
    int size = value.length();
    // Null pointer exception
}
```

```
var value: String? = null

fun doSomeStuff() {
    val size :Int? = value?.length
    // Anti-NPE
}
```

EMPRESAS QUE USAN KOTLIN



Android Studio y Kotlin



1.

**¿Es necesario
algún plugin?**



No, Kotlin está totalmente soportado en [Android Studio 3.0](#). Puedes usar todas sus herramientas, autocompletar, etc.

Además, es posible usar un “conversor” de Java a Kotlin.



2.

**¿Cómo hacer que
nuestro proyecto
lo soporte?**



Al crear un nuevo proyecto es necesario marcar el check que indica “Include Kotlin Support”.

También es posible crear clases en Kotlin en los proyectos existentes.



3.

Conversor

“

Simplemente tenemos que copiar código Java en nuestro fichero Kotlin. Seguidamente saldrá un diálogo que nos preguntará si queremos que nos convierta nuestro código.

Bind de vistas





1.

Propiedades de extensión



Las propiedades de extensión evitarán las llamadas repetitivas al método *findViewById* al crear las propiedades usando su id.



2.

¿Cómo las
usamos?

“

Accederemos a la vista a través de una propiedad de la clase nombrada con el mismo id que tenía en el *layout*.



3.

Ejemplo



Tenemos que importar todas
nuestras propiedades de extensión:

```
import kotlinx.android.synthetic.main.activity_main.*
```

Hacemos referencia al *TextView* a
través de su id:

```
label1.text = "HoLaMundo"
```

Definición de *listeners*





1.

Un solo método

“

Si la interfaz sólo tiene un método es mejor usar una lambda

```
class CarAdapter(val items: List<CarModel>, val listener: (CarModel) -> Unit)  
: RecyclerView.Adapter<CarAdapter.CarViewHolder>() {
```

```
    mainList.adapter = CarAdapter(carList, {  
        Toast.makeText(context, this@MainActivity,  
            text: "clickOnCarItem: " + it.brand, Toast.LENGTH_SHORT).show()  
    })
```



2.

**Más de un
método**

“

En este caso sería necesario crear un objeto de la interfaz

```
interface CustomListener {  
    fun clickOnCarItem(carModel: CarModel)  
    fun longClickOnCarItem(carModel: CarModel)  
}
```

```
class CustomCarAdapter(val items: List<CarModel>, val listener: CustomListener)  
    : RecyclerView.Adapter<CustomCarAdapter.CarViewHolder>() {
```

“

Establecer el *Adapter* al *RecyclerView*

```
mainList.adapter = CustomCarAdapter(carList, object : CustomCarAdapter.CustomListener {  
    override fun clickOnCarItem(carModel: CarModel) {  
        Toast.makeText(context: this@MainActivity,  
            text: "clickOnCarItem: " + carModel.brand, Toast.LENGTH_SHORT).show()  
    }  
  
    override fun longClickOnCarItem(carModel: CarModel) {  
        Toast.makeText(context: this@MainActivity,  
            text: "longClickOnCarItem: " + carModel.brand, Toast.LENGTH_SHORT).show()  
    }  
})
```

Android – Funciones de extensión





1.

Utilidades



View

```
fun View.isVisible() : Boolean {  
    return this.visibility == View.VISIBLE  
}
```

ImageView y Picasso

```
fun ImageView.loadUrl(url: String) {  
    Picasso.with(this.context).load(url).into(target: this)  
}
```

“

TextView

```
fun TextView.setTextAndHideViewIfIsNeeded(text: String) {  
    if (!TextUtils.isEmpty(text)) {  
        this.text = text  
        this.visibility = View.VISIBLE  
    } else {  
        this.text = ""  
        this.visibility = View.GONE  
    }  
}
```




RecyclerView

```
fun RecyclerView.setUp(layoutManager: RecyclerView.LayoutManager,  
                        adapter: RecyclerView.Adapter<RecyclerView.ViewHolder>) {  
    this.layoutManager = layoutManager  
    this.adapter = adapter  
}
```



Activity

```
fun Activity.showToast(text: String, duration: Int = Toast.LENGTH_SHORT) {  
    Toast.makeText(context: this, text, duration).show()  
}
```

```
fun <T : View> Activity.bindView(@IdRes res: Int): Lazy<T> {  
    return lazy(LazyThreadSafetyMode.NONE) { findViewById<T>(res) }  
}
```

```
fun <T : View> Activity.bindViews(@IdRes resList: IntArray): List<Lazy<T>> {  
    return resList.map { bindView<T>(it) }.toList()  
}
```



2.

**¿Dónde las
declaramos?**



Crearemos un archivo con la palabra
Extension al final.

Ej: *ActivityExtension.kt*

ViewExtension.kt

3.

**¿Cómo las
usamos?**



El *import* necesario

```
import com.openwebinars.funcionesdeextension.extension.*
```

```
var titleLabel: Lazy<TextView> = bindView(R.id.main__title_label)
```

```
titleLabel.value.isVisible()
```

```
titleLabel.value.setTextAndHideViewIfIsNeeded("")
```

```
showToast( text: "Hola Mundo")
```

```
showToast( text: "Hola Mundo", duration: 500)
```



Android Lambdas

1.

setOnClickListener
r

Java

```
view.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        navigator.goToDetail();  
    }  
});
```

Kotlin

```
textView.setOnClickListener({ view -> navigateToDetail() })
```



```
textView.setOnClickListener({ navigateToDetail() })
```



```
textView.setOnClickListener() { navigateToDetail() }
```



```
textView.setOnClickListener { navigateToDetail() }
```



2.

**Salvando la
fragmentación**

Ejecutar función si es Nougat o superior

```
private inline fun executeIfIsNougatOrAbove(function: () -> Unit) {  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {  
        function()  
    }  
}
```

```
executeIfIsNougatOrAbove { Log.e( tag: "Test", msg: "test") }
```

Ejecutar una función u otra

```
private inline fun processNumber(number: Int, executeIfIsPair: () -> Unit,  
    executeIfNot: () -> Unit) {  
    if (number % 2 == 0) executeIfIsPair() else executeIfNot()  
}
```

```
processNumber( number: 20, executeIfIsPair = {  
    Log.d( tag: "Test", msg: "Es par")  
}, executeIfNot = {  
    Log.d( tag: "Test", msg: "No es par")  
})
```

RecyclerView y Adapter





1.

Definición del *Adapter*

“

El constructor recibirá una lista de objetos y una Lambda

```
class CarAdapter(val items: List<CarModel>, val listener: (CarModel) -> Unit)  
: RecyclerView.Adapter<CarAdapter.CarViewHolder>() {
```




2.

Implementar métodos

Métodos

- ▶ ***onCreateViewHolder***. Retornará el *ViewHolder* de nuestro *adapter* por lo crearemos usando su constructor.
- ▶ ***onBindViewHolder***. Configurarán la vista para cada elemento. En este método configuraremos el click del elemento para que ejecute la lambda.
- ▶ ***getItemCount***. Retornará el tamaño de los elementos que contiene.



3.

**Creamos el
ViewHolder**

“

Con menos código es imposible...

```
class CarViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  
    val carIdLabel = itemView.rowCarId  
    val carNameLabel = itemView.rowCarName  
}
```



4.

**Creamos el
listado**

“

Usamos la operación *mapTo*
pasándole la función de
transformación:

```
val carList = mutableListOf<CarModel>()  
(1..10).mapTo(carList) { CarModel(""+ it, String.format("Coche %s", it)) }
```



5.

**Configuramos el
*RecyclerView***

“

Creamos el *LayoutManager* y el *Adapter*.

```
mainRecycler.layoutManager = LinearLayoutManager(this)  
mainRecycler.adapter = ItemAdapter(items) {  
    | toast(String.format("Click en %s", it.farmerId))  
}
```


Retrofit y JobQueue



1.

**Añadir
dependencias**

“

Debemos añadir *Retrofit* y *OkHttp3* a nuestro fichero *build.gradle*

```
compile "com.squareup.okhttp3:okhttp:3.9.0"  
compile "com.squareup.okhttp3:logging-interceptor:3.9.0"  
  
compile ("com.squareup.retrofit2:retrofit:2.3.0"){  
    |   exclude module: 'okhttp'  
}  
compile "com.squareup.retrofit2:converter-gson:2.3.0"
```



2.

**Nuestra primera
interfaz**

“

Tendrá un método que se encargará de hacer una llamada a un servicio

```
interface ResourceService {  
    @GET("resource/hma6-9xbg.json")  
    fun requestResourceList(@Query("category") category: String,  
                           @Query("item") item: String): Call<List<ItemDTO>>  
}
```



3.

Crear instancias

Clase que contiene un *companion object* para inicializar *Retrofit* y *OkHttp*

```
class ApiUtils {  
    companion object {  
        private fun generateOkHttpBuilder(): OkHttpClient {  
            return OkHttpClient().newBuilder()  
                .build()  
        }  
  
        fun generateRetrofitInstance(): Retrofit {  
            return Retrofit.Builder()  
                .baseUrl(AppConstants.ENDPOINT)  
                .client(generateOkHttpBuilder())  
                .addConverterFactory(GsonConverterFactory.create())  
                .build()  
        }  
    }  
}
```



4.

Modelado

“

ItemDto. Objeto de transferencia de datos

```
class ItemDto(  
    @SerializedName("item") val item: String,  
    @SerializedName("business") val business: String,  
    @SerializedName("farmer_id") val farmerId: String,  
    @SerializedName("category") val category: String,  
    @SerializedName("l") val l: String,  
    @SerializedName("farm_name") val farmName: String,  
    @SerializedName("phone1") val phone1: String  
)
```

“

ItemModel. Objetos que usará
nuestro adapter

```
class ItemModel(  
    val item: String?,  
    val business: String?,  
    val farmerId: String?,  
    val category: String?,  
    val l: String?,  
    val farmName: String?,  
    val phone1: String?  
)
```



6.

Mapper

“

ItemMapper. DTO → Model

```
class ItemMapper {  
    fun transform(items: List<ItemDTO>): List<ItemModel> {  
        return items.map { transform(it) }  
    }  
  
    fun transform(item: ItemDTO): ItemModel {  
        return ItemModel(item.item,  
            item.business,  
            item.farmerId,  
            item.category,  
            item.l,  
            item.farmName,  
            item.phone1)  
    }  
}
```

A thick, light blue diagonal line runs from the top right corner towards the bottom left, separating the white background on the left from the solid blue background on the right.

7.

Dependencia y configuración de JobQueue

Pasos

1. Añadir dependencia

```
compile 'com.birbit:android-priority-jobqueue:2.0.0'
```

2. Inicializamos la interfaz

```
val resourceService = ApiUtils  
    .generateRetrofitInstance()  
    .create(ResourceService::class.java)
```

3. Realizamos la llamada

```
val call = resourceService.requestResourceList("Fruit", "Peaches")
```

Pasos

4. Después de ejecutar la llamada transformamos los DTOs a objetos de dominio

```
val result = call.execute().body()
val items = ModelMapper().transform(result!!)
```

Pasos

5. Creamos el manager que controlará la ejecución del *job*

```
val builder = Configuration.Builder(this)
    .minConsumerCount(1)
    .maxConsumerCount(3)
    .loadFactor(3)
    .consumerKeepAlive(120)

val jobManager: JobManager = JobManager(builder.build())
val serviceJob: GetResourceListJob = GetResourceListJob(Params(50).requireNetwork(), this)
jobManager.addJobInBackground(serviceJob)
jobManager.start()
```




8.

Crear Job - onRun

“

Constructor

```
class GetResourceListJob(params: Params?, val view: MainView) : Job(params) {
```

Interfaz *MainView*

```
interface MainView {  
    fun setDataSet(items: List<ItemModel>)  
}
```

“

Lanzar resultado en primer plano

```
val uiHandler = Handler(Looper.getMainLooper())
val runnable = Runnable {
    | view.setDataSet(items)
}
uiHandler.post(runnable)
```

Implementar método de la interfaz

```
override fun setDataSet(items: List<ItemModel>) {
    | mainRecycler.layoutManager = LinearLayoutManager(context: this)
    | mainRecycler.adapter = ItemAdapter(items) {
        | toast(String.format("Click en %s", it.farmerId))
    }
}
```

Base de datos





1.

Realm

Build.gradle

- **Del proyecto.** Es necesario añadir en buildscript -> dependencies el siguiente código.

```
classpath "io.realm:realm-gradle-plugin:4.3.1"
```

- **Del módulo.** Tenemos que añadir el siguiente plugin.

```
apply plugin: 'realm-android'
```



2.

Inicializar Realm

“

Es necesario inicializar la instancia de Realm en la clase MyApplication y asignar dicha clase en el fichero AndroidManifest.

```
class MyApplication: Application() {  
    override fun onCreate() {  
        super.onCreate()  
        Realm.init( context: this)  
    }  
}
```

```
<application  
    android:name=".MyApplication"
```




3.

**Añadir
configuraciones a
la instancia**

“

Podemos añadir configuraciones a la instancia de Realm, tales como nombre de la base de datos, migrado, versión, etc.

```
val config = RealmConfiguration.Builder()  
    .name( filename: "openwebinars.realm").build()  
Realm.setDefaultConfiguration(config)
```



4.

**Obtener la
instancia de
Realm**

“

Para obtener la instancia de Realm
usaremos el método
getDefaultInstance

```
Realm.getDefaultInstance()
```



5.

Objeto Realm



6.

**Ejecutar en una
transacción**

“

❏ Síncrona

```
realm.executeTransaction {  
  
}
```

❏ Asíncrona

```
realm.executeTransactionAsync {  
  
}
```




7.

Guardar un dato

“

Usaremos el método ***copyToRealmOrUpdate*** y le pasaremos el objeto a guardar. Se encargará de insertar o editar el dato si ya existiese.

```
realm.copyToRealmOrUpdate(CarModel(text, text))
```



8.

**Obtener todos los
datos de una tabla**

“

Haremos una consulta a la tabla y obtendremos todos los resultados.

```
realm.where<CarModel>().findAll().toList()
```



9.

**Encontrar y
eliminar un dato**

“

Haremos una consulta a la tabla y obtendremos el primer valor que cumple la condición, posteriormente lo eliminamos.

```
realm.where<CarModel>().contains( fieldName: "brand", text)  
    .findFirst()?.deleteFromRealm()
```

10.

**Eliminar todos los
datos de una tabla**

“

A partir de la instancia borraremos todos los datos de la base de datos.

```
realm.deleteAll()
```