App of Thrones

Curso de desarrollo de aplicaciones Android en Kotlin desde 0

Pedro Antonio Hernández López
Software Developer at Bunsan.io
@silmood

KEEP CODING

Pedro Antonio Hernández López
Software Developer at Bunsan.io
@silmood

Instalando nuestro entorno

Descargar Android Studio

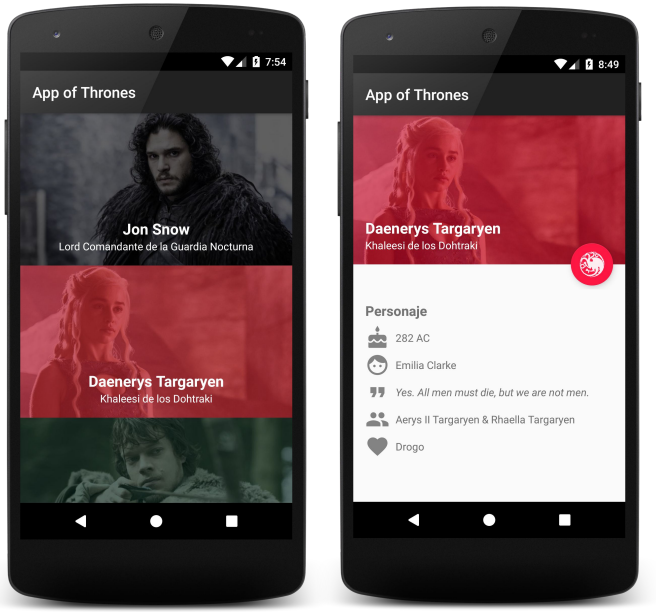
Pasos para instalación

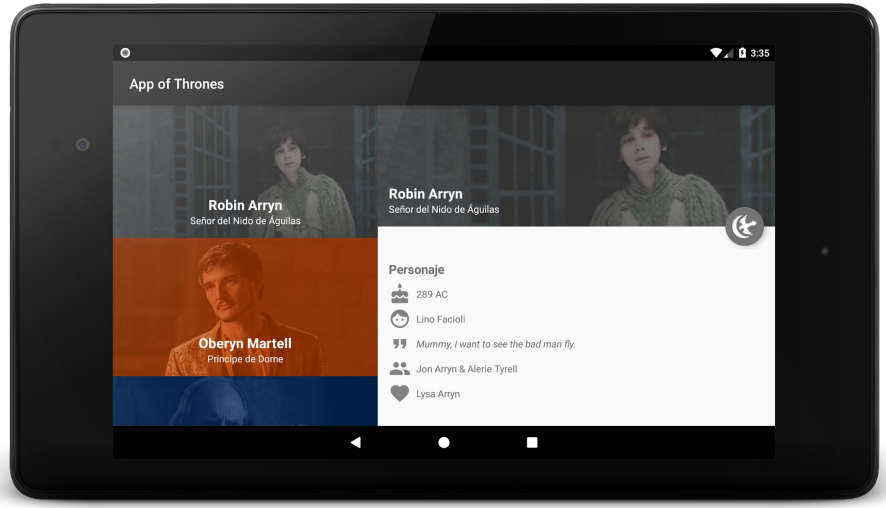
- Instalar JDK
- Instalar Android Studio
- Instalación de plataformas

App of Thrones

Construyendo nuestra primer aplicación en android

App of Thrones





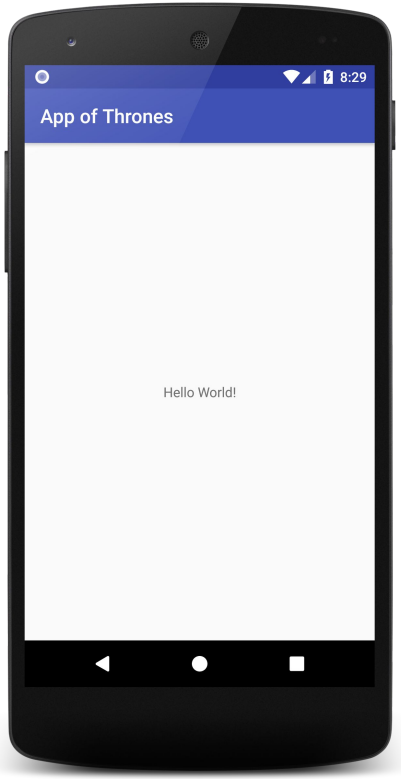
Creando nuestro primer proyecto

Tour por Android Studio

- Navegación del proyecto
- Toolbar
- Consola de ejecución
- Logger
- Debugger
- Terminal integrada
- Log de eventos

Hello Android!

Creemos un emulador



Hello Android!

Run, run, run

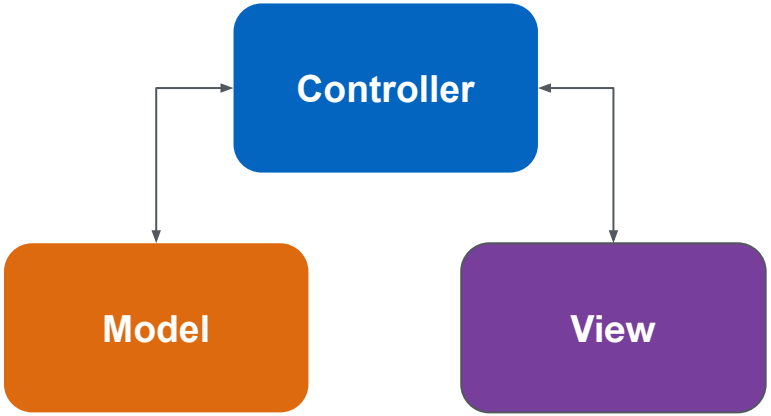
Estructura de un proyecto

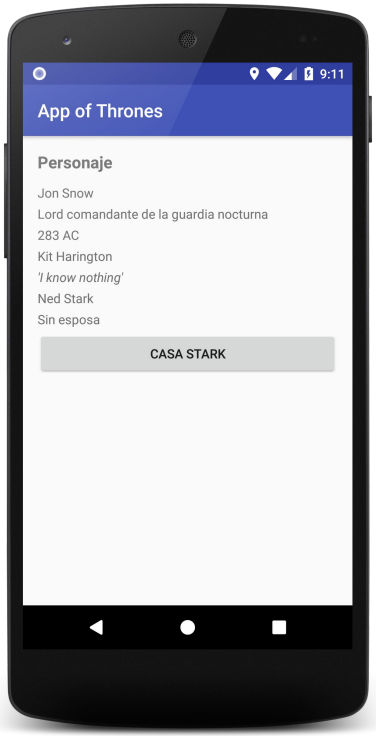
En dónde estaremos trabajando

- java
- res
- gradle scripts

MVC en Android

Nuestra primer meta

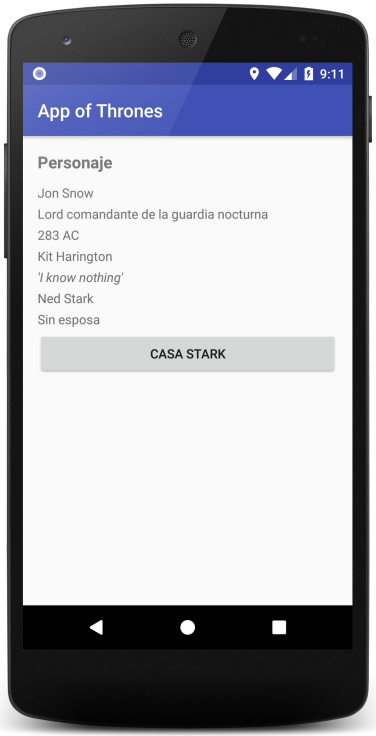




Vistas en Android

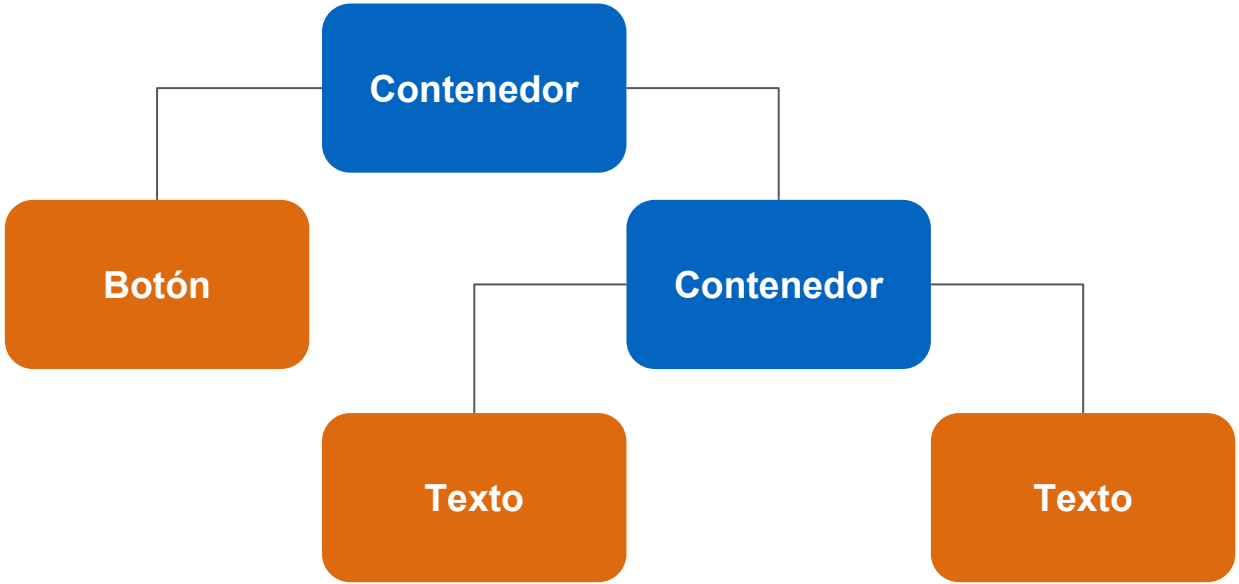
Recursos de vistas

- res
  - layout



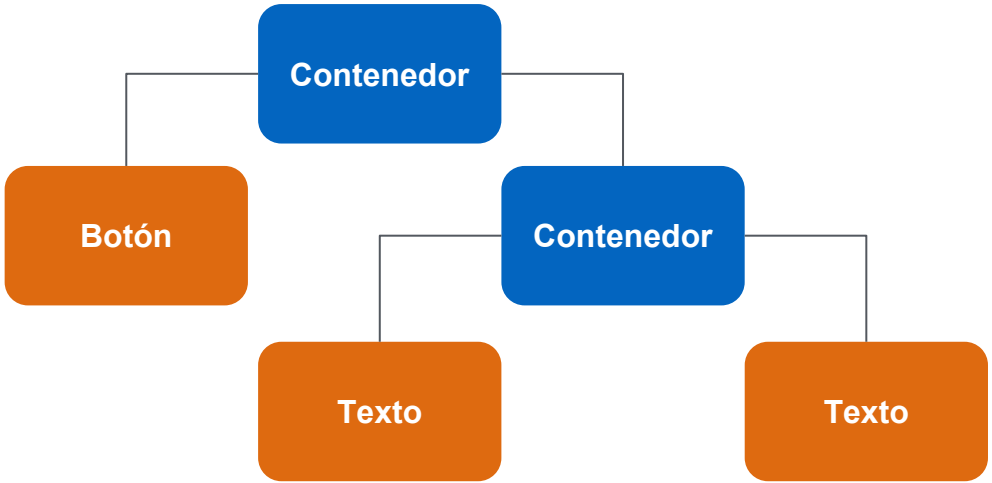
Vistas en Android

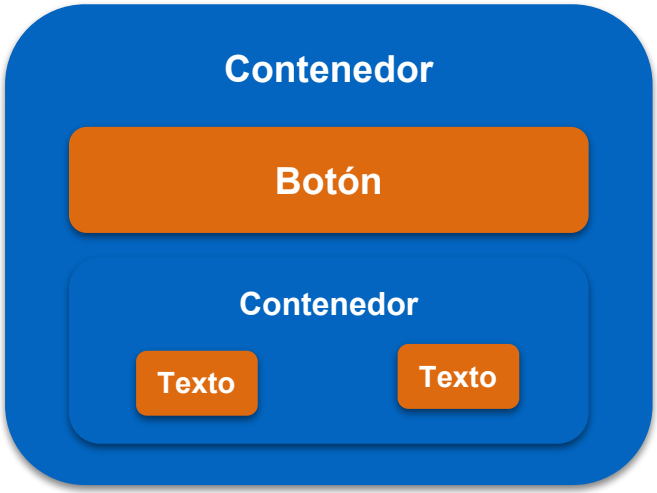
Jerarquía de vistas - Ejemplo



Vistas en Android

Jerarquía de vistas - Ejemplo





Vistas en Android

¿Qué es XML?

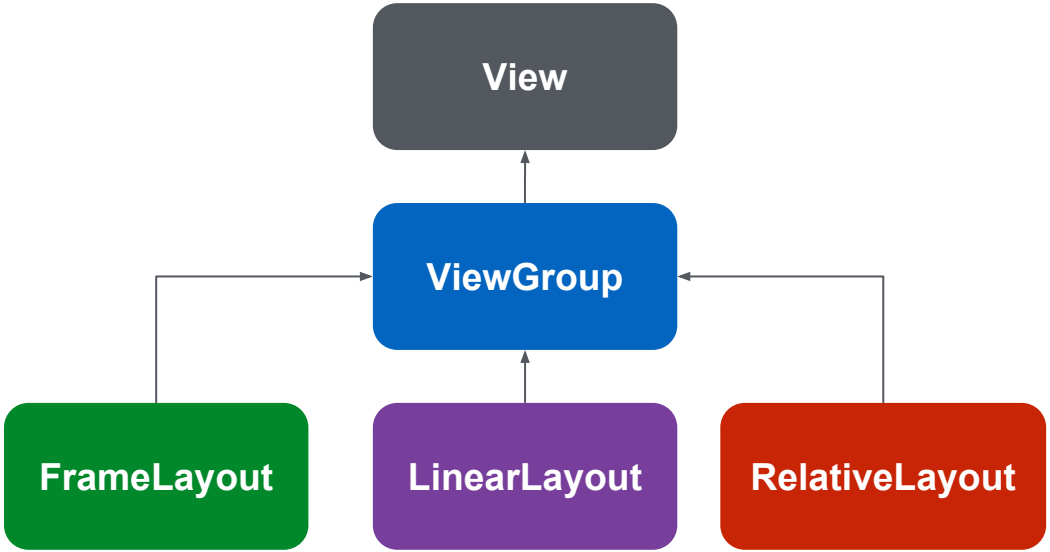
Vistas en Android

¿Qué es XML?

```
<Contenedor ...>
   <Boton ... />
</Contenedor>
```

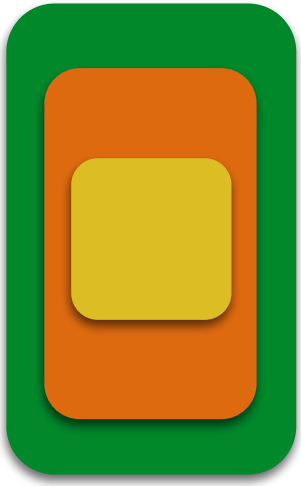
Vistas en Android

Contenedores en Android

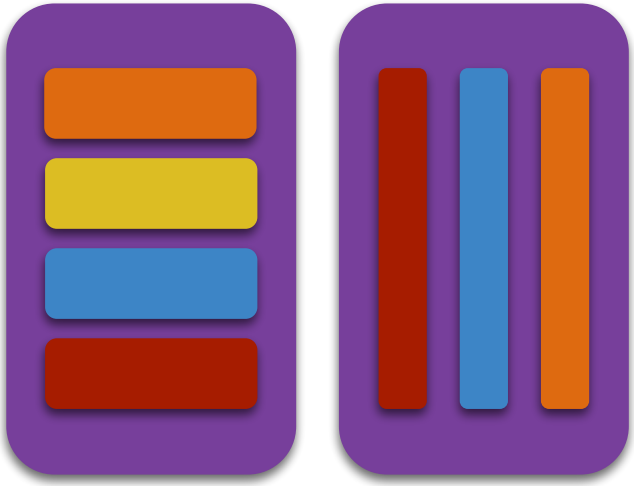


Vistas en Android

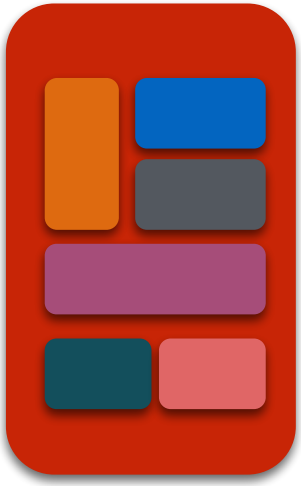
Contenedores en Android - Comportamientos



FrameLayout



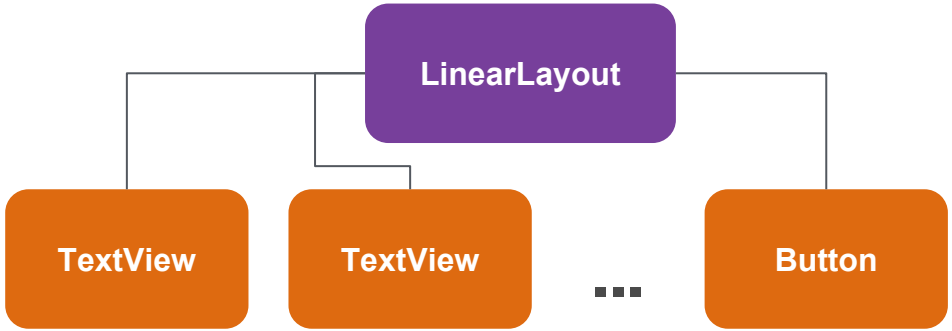
LinearLayout

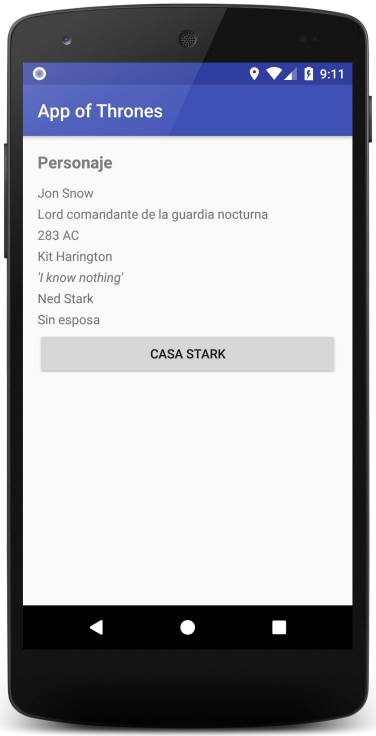


RelativeLayout

Vistas en Android

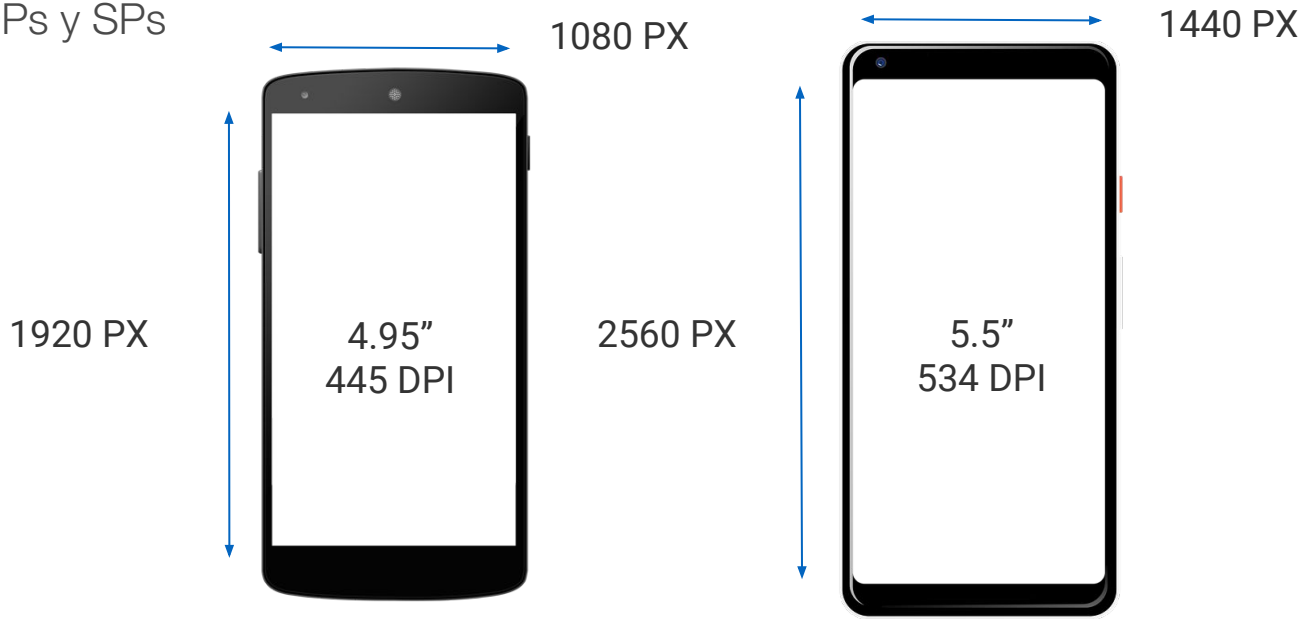
Creando nuestra primer vista





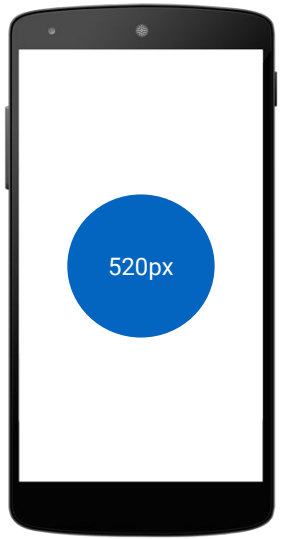
Vistas en Android

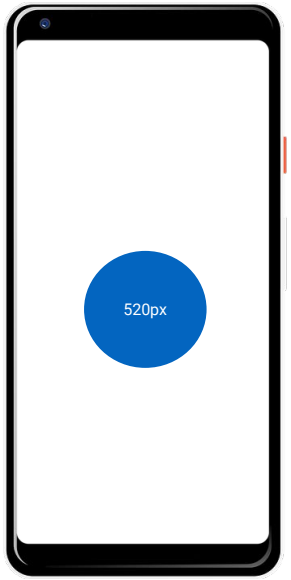
DPs y SPs



Vistas en Android

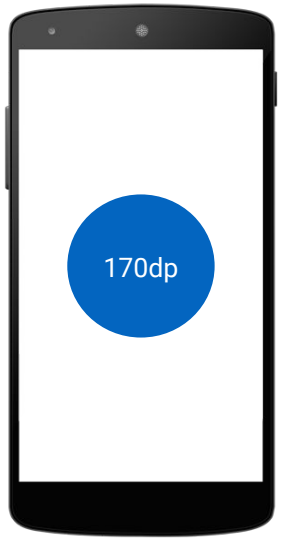
DPs y SPs

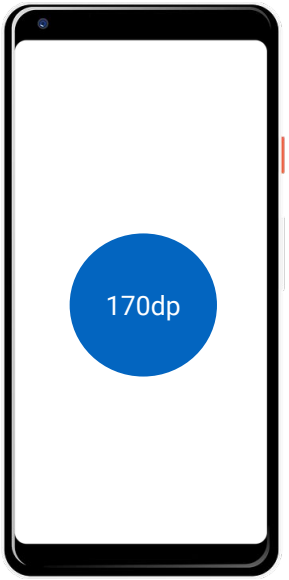




Vistas en Android

DPs y SPs





Vistas en Android

DPs y SPs

mdpi /~160dpi
1dp = 1px

hdpi /~240dpi
1dp = 1.5px

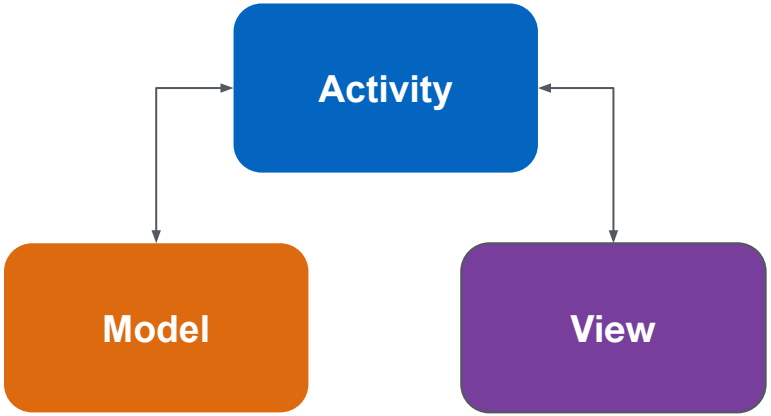
xhdpi /~320dpi
1dp = 2px

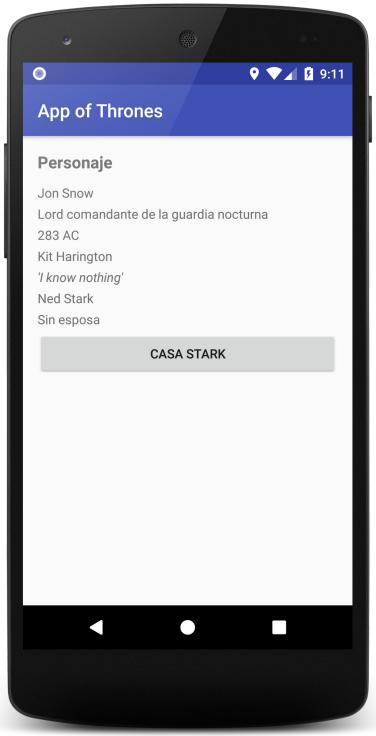
xxhdpi /~480dpi
1dp = 3px

xxxhdpi /~640dpi
1dp = 4px

MVC en Android

Controlador





Bienvenido a Kotlin

Get ready!

KEEP CODING

Kotlin 101

- Historia
- Filosofía
- Funcionamiento
- Interoperabilidad
- Características generales

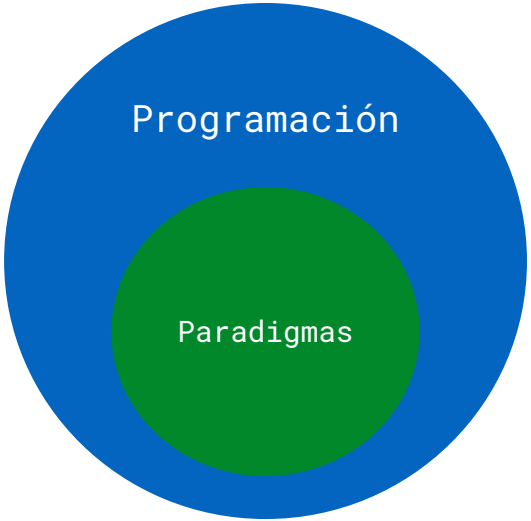
Kotlin 101

Bye Bye ;

KEEP CODING

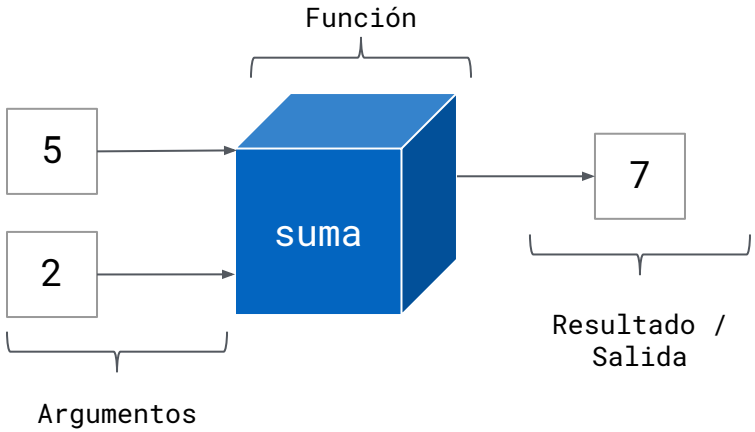
Kotlin 101

Programación Funcional y Programación Orientada a Objetos



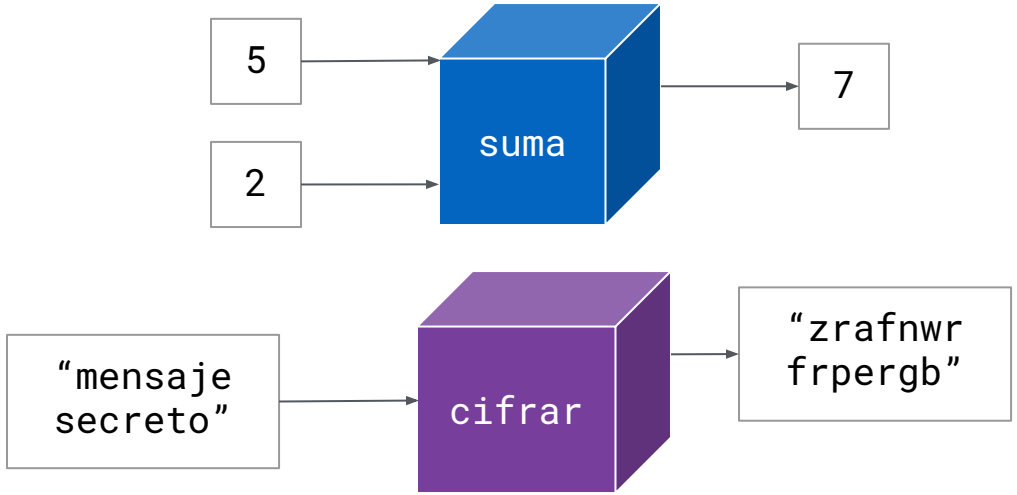
Kotlin 101

Programación Funcional y Programación Orientada a Objetos



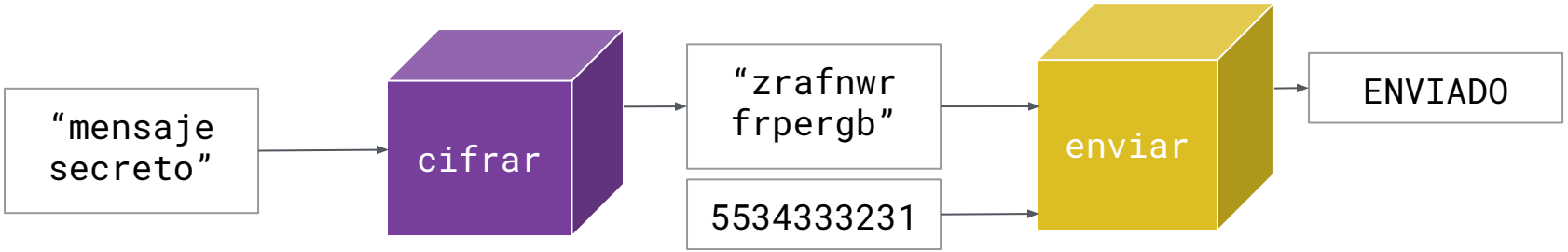
Kotlin 101

Programación Funcional y Programación Orientada a Objetos



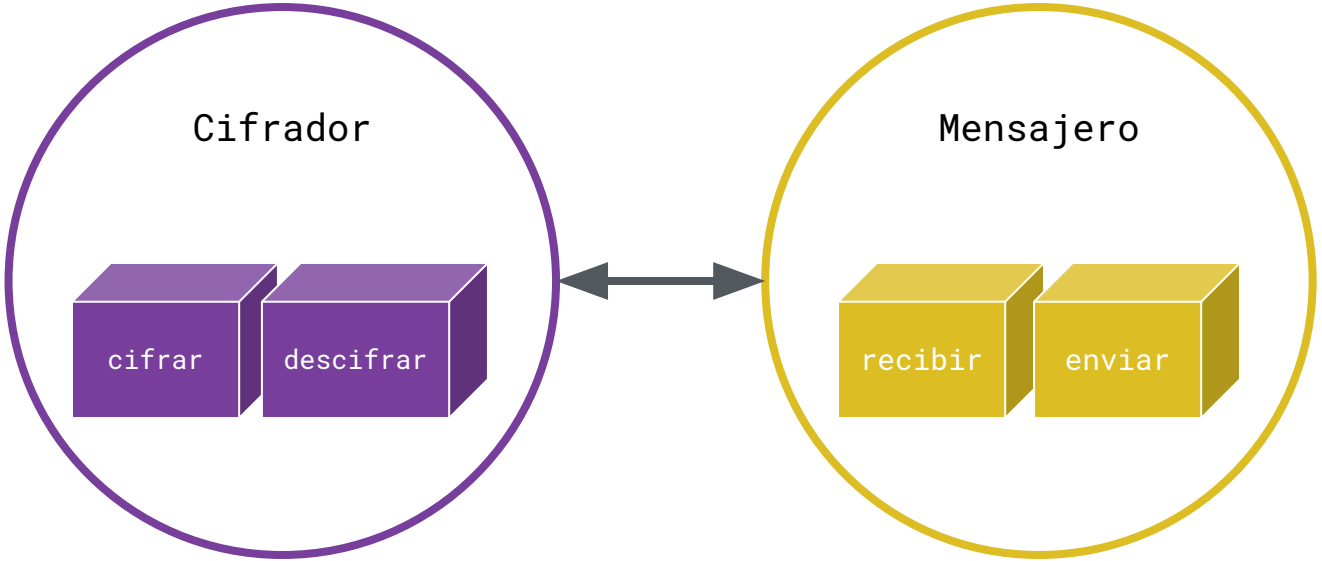
Kotlin 101

Programación Funcional y Programación Orientada a Objetos



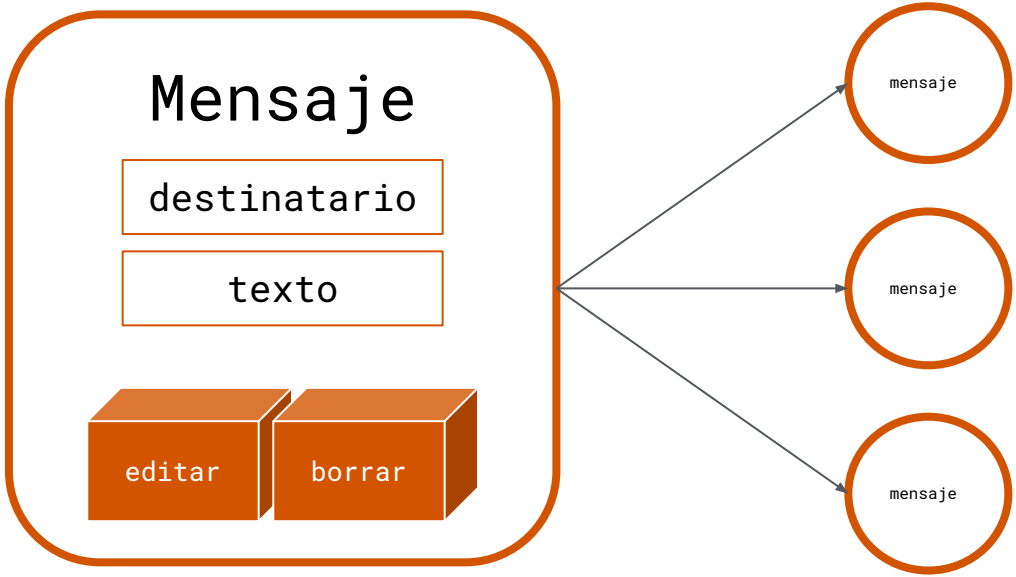
Kotlin 101

Programación Funcional y Programación Orientada a Objetos



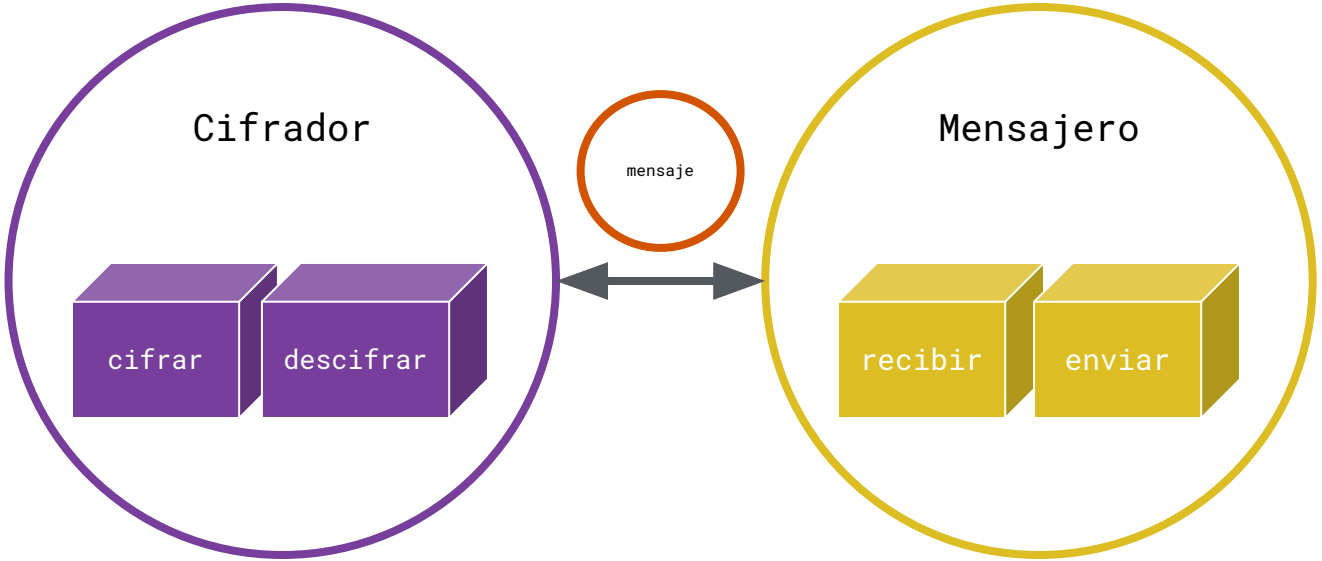
Kotlin 101

Programación Funcional y Programación Orientada a Objetos



Kotlin 101

Programación Funcional y Programación Orientada a Objetos



Kotlin 101

Programación Funcional y Programación Orientada a Objetos

Kotlin 101

Sintaxis - Clases

```
class Person {
    //...
}
```

Kotlin 101

Sintaxis - Constructores

```
class Person constructor(name: String) {
    //...
}
```

Kotlin 101

Sintaxis - Constructores

```
class Person(name: String) {
    //...
}
```

Kotlin 101

Sintaxis - Bloque inicializador

```
class Person(name: String) {
    init {
        print("Hello, my name is ${name}")
    }
}
```

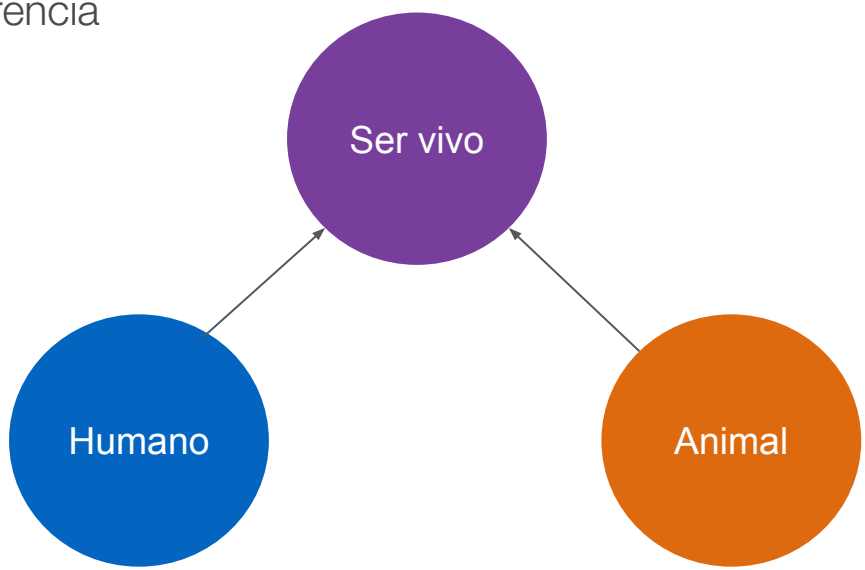
Kotlin 101

Sintaxis - Propiedades de una clase

```
class Person(val name: String) {
    init {
        print("Hello, my name is ${name}")
    }
}
```

Kotlin 101

Sintaxis - Herencia



Kotlin 101

Sintaxis - Herencia

```
open class Organism(name: String) {
    init {
        print("I'm Alive!")
    }
}
```

Kotlin 101

Sintaxis - Herencia

```
class Person(name: String) : Organism(name) {
    //..
}

class Animal(name: String, habitat: String) : Organism(name) {
    //..
}
```

Kotlin 101

Sintaxis - Métodos de una clase

```
fun greeting(to: String) : String {
    return "Hello ${to}"
}
```

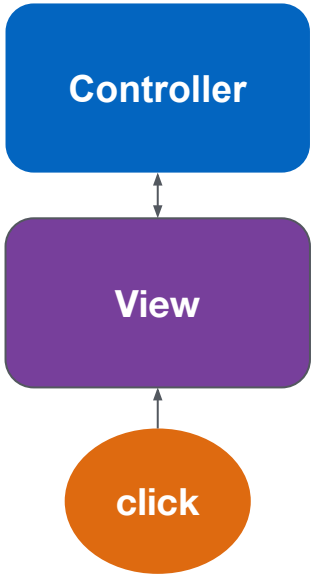
Kotlin 101

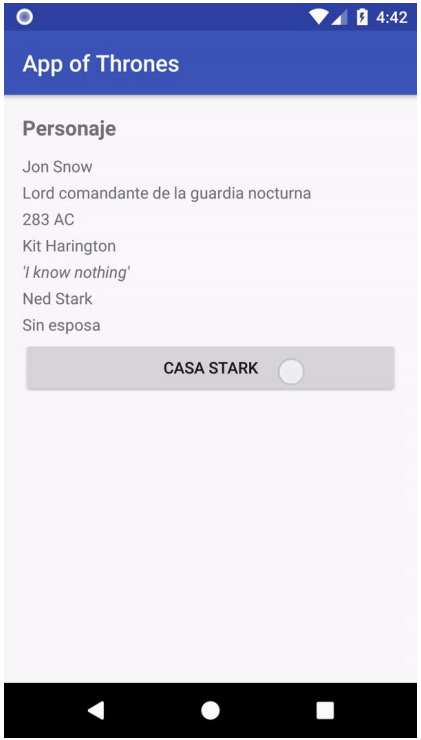
Sintaxis - Sobreescritura de métodos

```
open class Organism(val name: String) {
    open fun eat() {
        print("Yummy!")
    }
}
class Plant(name: String, val habitat: String) : Organism(name) {
    override fun eat() {
        super.eat()
        print("Photosyntesis!")
    }
}
```

El siguiente objetivo

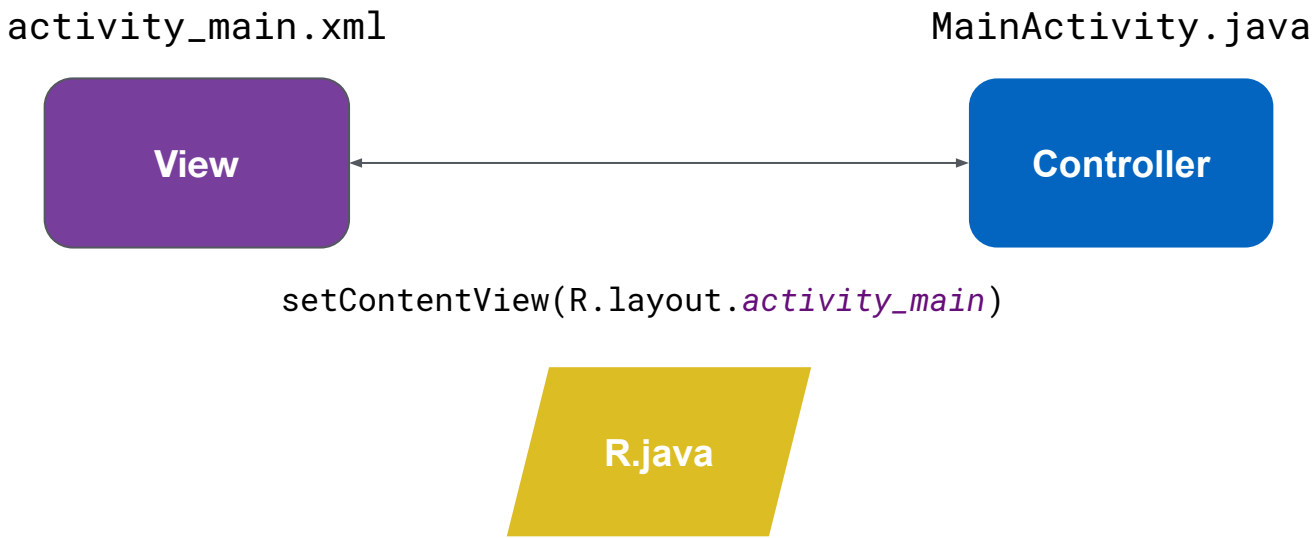
Interactuando con el usuario





Asignar vista a un controlador

Asignando una vista a una actividad



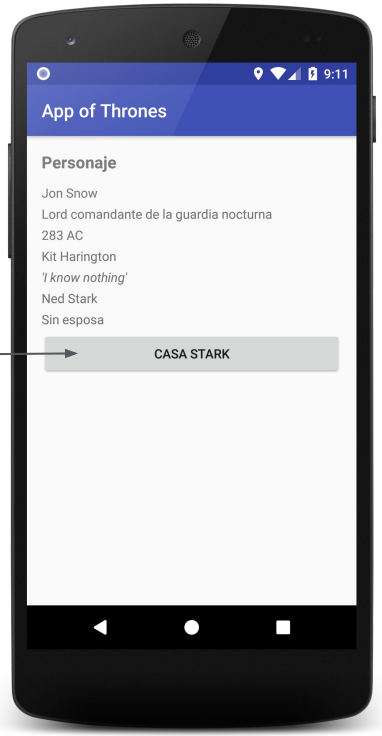
Accediendo a la vista

desde nuestro controlador

Activity

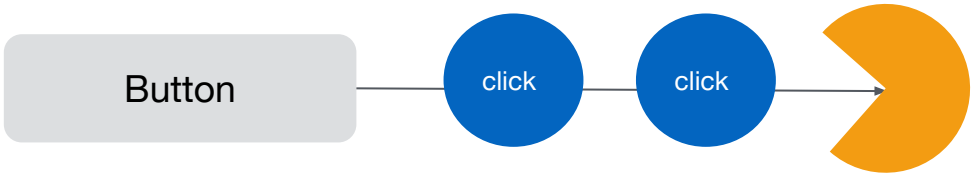
```
findViewById(R.id.button)
```

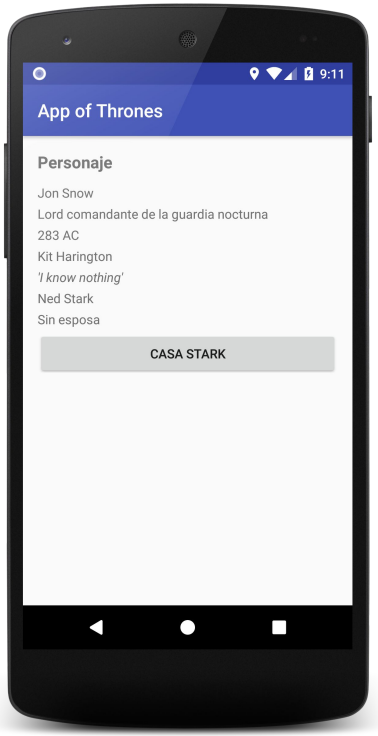
```
@+id/button
```



Escuchando eventos

Click Listener



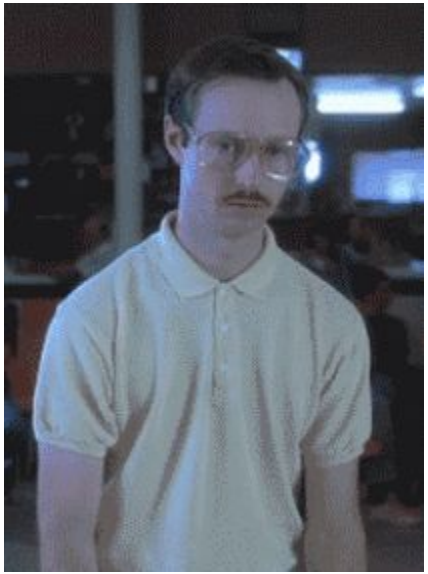


Lambdas

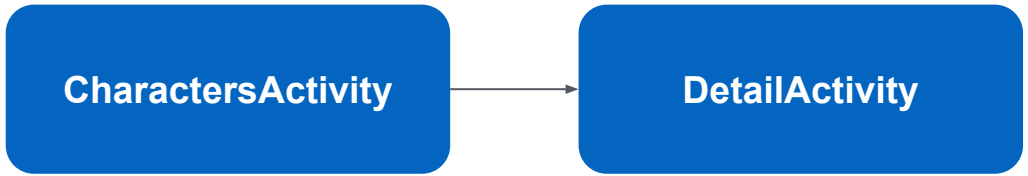
Funciones de primer orden

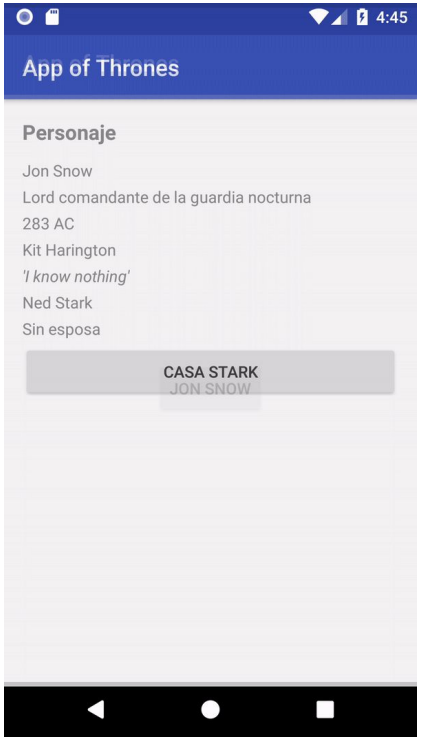
```
val greeting: () -> Unit = {
    print("Hello lambda!")
}
greeting.invoke()
```

Mission Completed



Nuestro siguiente reto





Creando una nueva actividad

ToDo List

- ➔ Añadir nuestra nueva actividad al archivo `Manifest.xml`
- ➔ Crear una clase que herede de `AppCompatActivity`
- ➔ Crear la vista de nuestra actividad
- ➔ Enlazar la vista con nuestra actividad

¿Qué es el archivo Manifest?

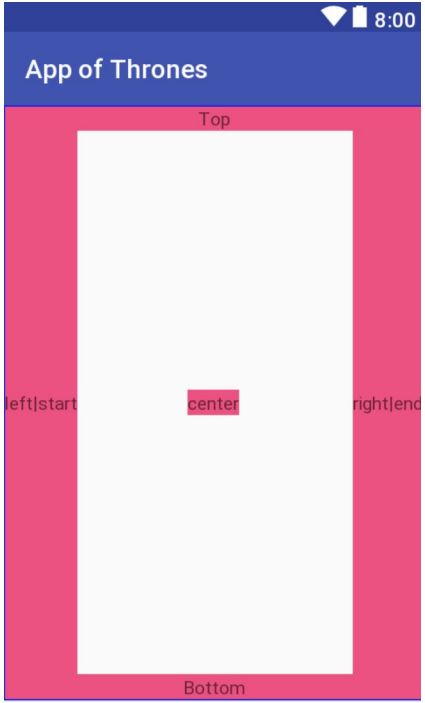
¿Por qué lo necesitamos?

- Permisos
- Icono de tu aplicación
- Nombre de la aplicación
- Activities

Gravity

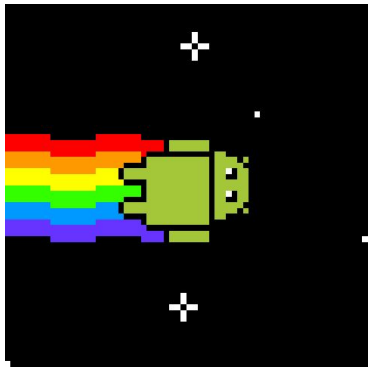
Alineación de elementos

```
android:gravity
android:layout_gravity
```



Logger

Una herramienta básica para el desarrollo

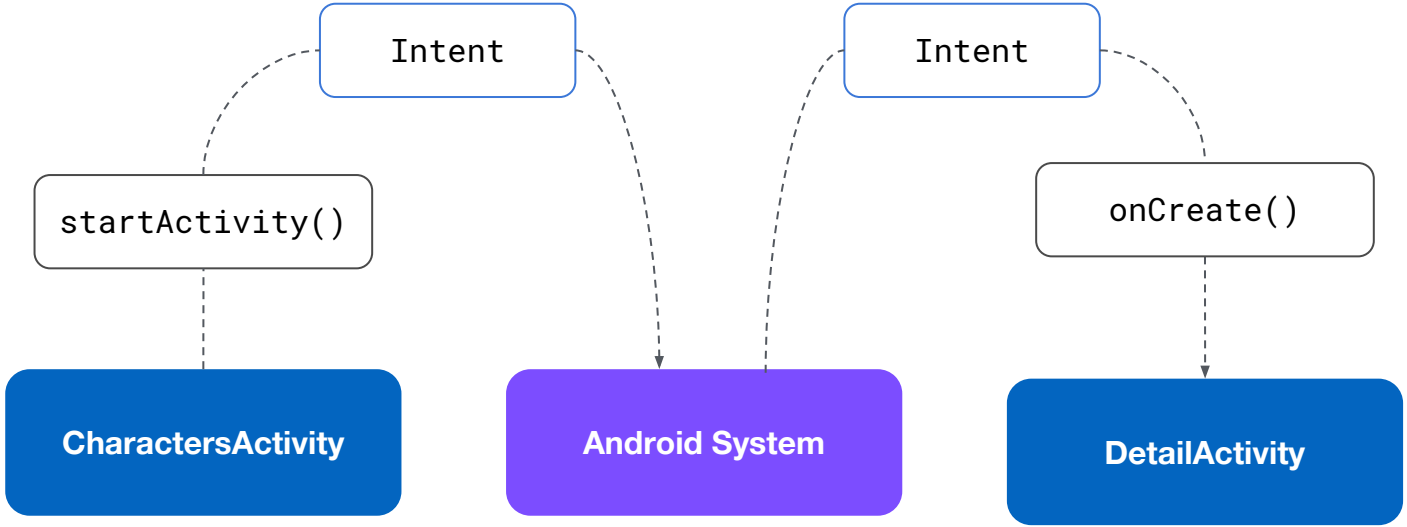


```
Log.d(String tag, String msg)
Log.w(String tag, String msg)
Log.e(String tag, String msg)
Log.v(String tag, String msg)
Log.wtf(String tag, String msg)
```

KEEP CODING

Intents

Navegación entre Actividades



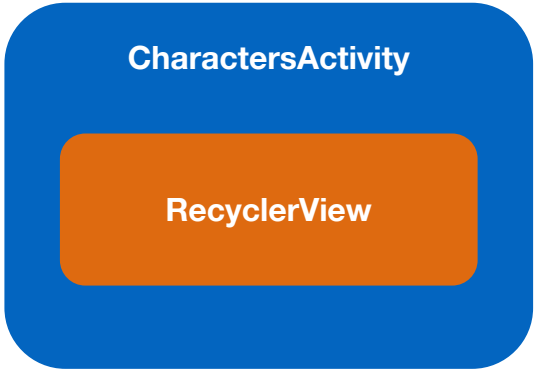
OnClick

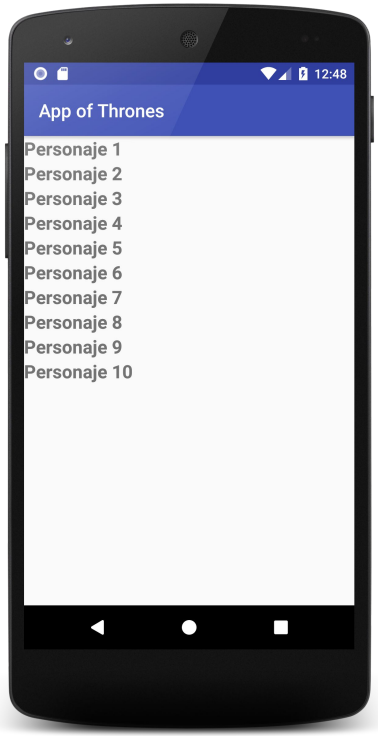
Estableciendo un onClickListener desde XML

```
android:onClick="showDetails"
```

Siguiente misión

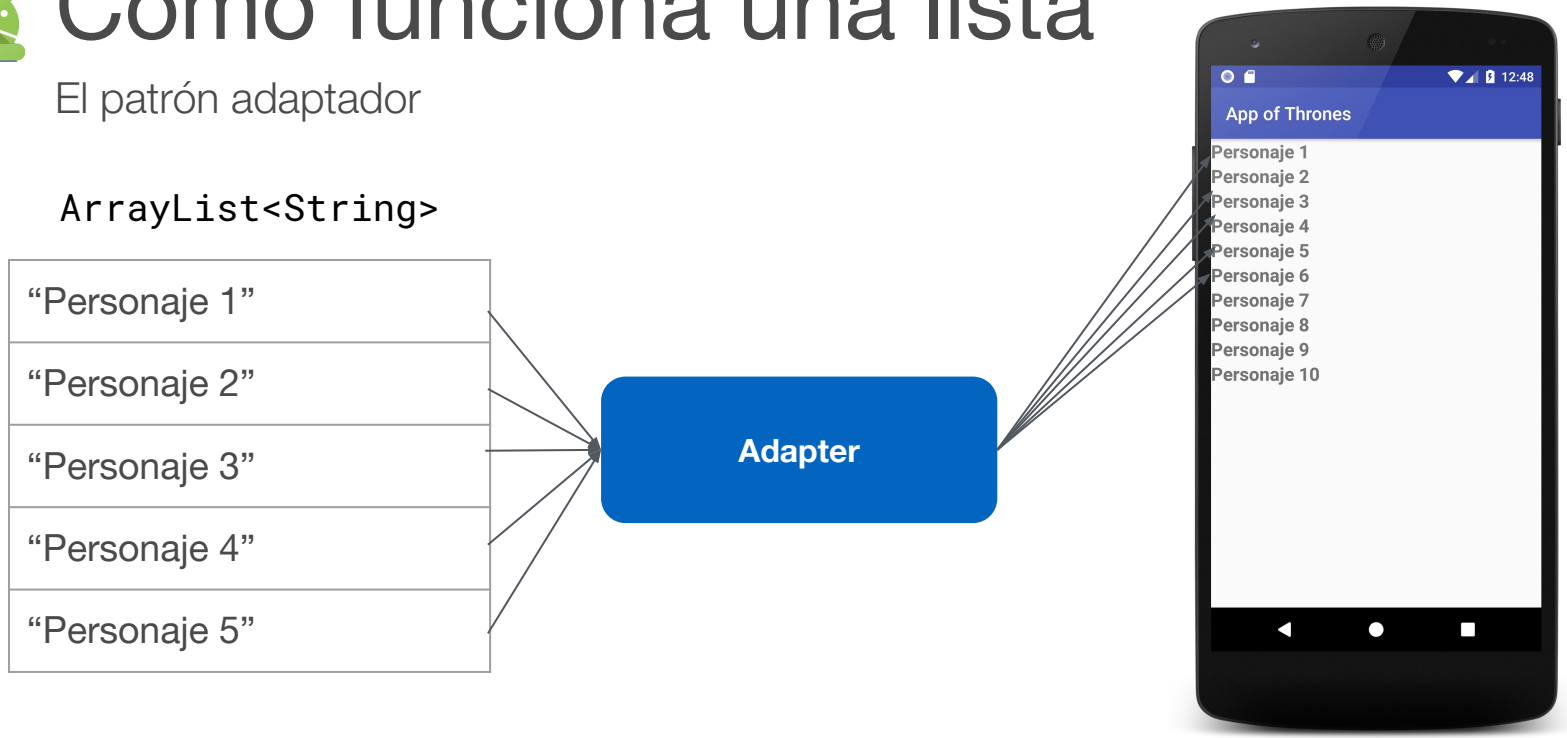
RecyclerView





Cómo funciona una lista

El patrón adaptador



Creando una lista

ToDo List

- ➔ Creación del modelo de datos
- ➔ Creación de la vista Item
- ➔ Añadir RecyclerView a nuestro controlador
- ➔ Creación de un adaptador

KEEP CODING

Bye POJO, Hello Data Class

Creación de un Data Class

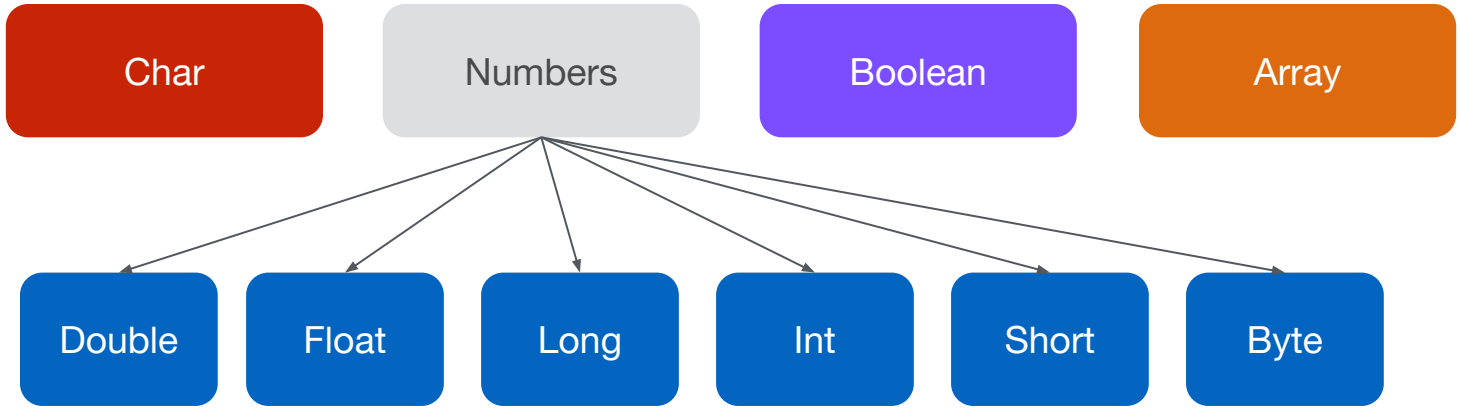
```
data class Character(var name: String, var title: String)
```

La mutabilidad es importante

var y val

```
var -> Variable
val -> Constante
```

Tipos Básicos



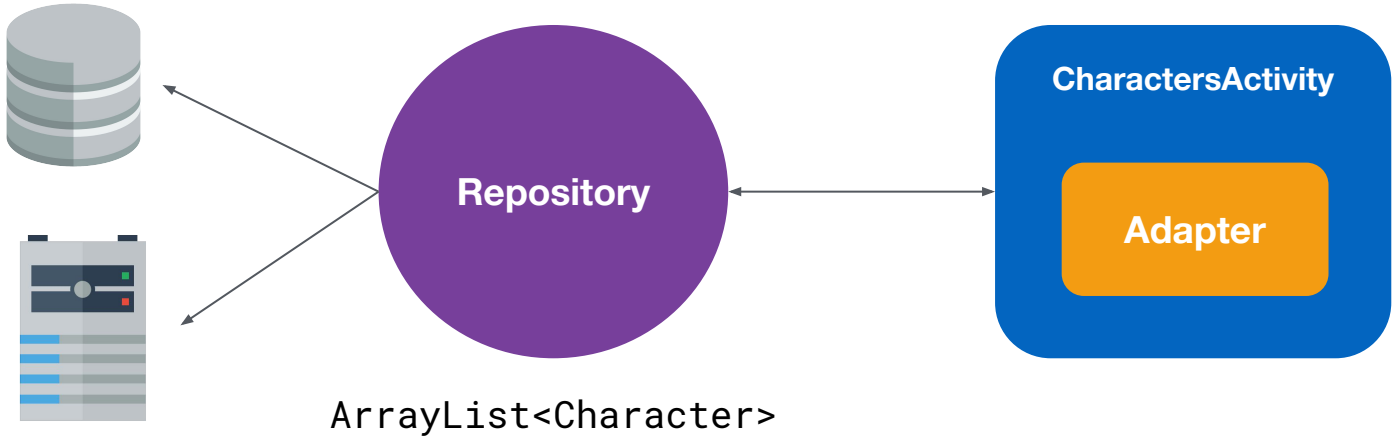
Argumentos opcionales

Y argumentos nombrados

```
data class Character(var name: String = “José Nieves”,
                     var title: String)

val character: Character = Character(title = “Comandante”)
```

Creando nuestra fuente de datos

Kotlin Object

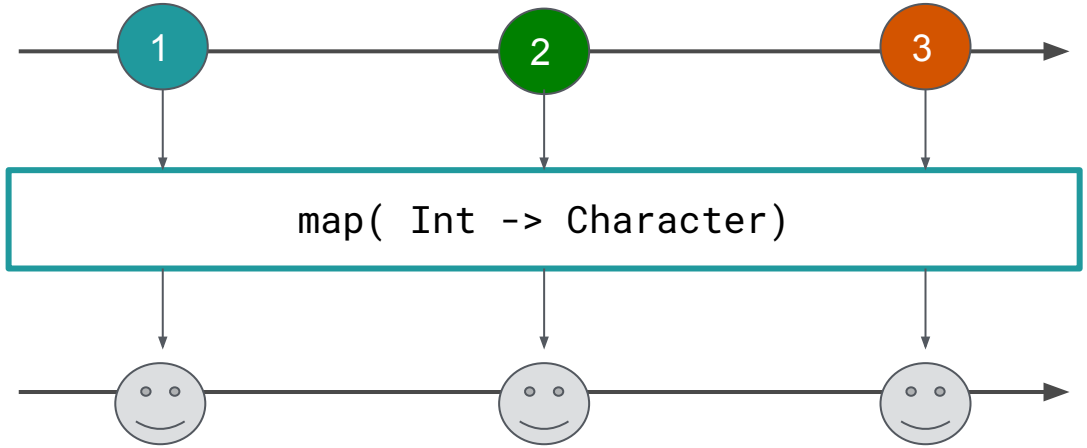


Objects: Singleton en Kotlin

Kotlin Object

```
object CharactersRepo {
    val characters: MutableList<Character> = mutableListOf()
}
```

Ranges y Operadores funcionales



Custom getters

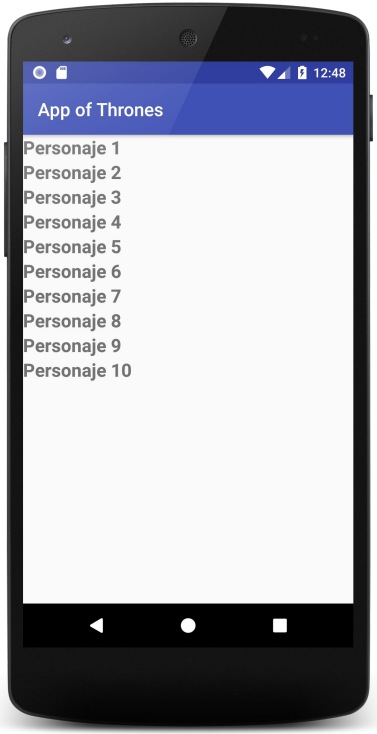
Escribir getter para una propiedad

```
val characters: MutableList<Character> = mutableListOf()
get() {
    if (field.isEmpty())
        field.addAll(dummyCharacters())

    return field
}
```

Vista de nuestro Item

Escribiendo una vista para una lista

TextView



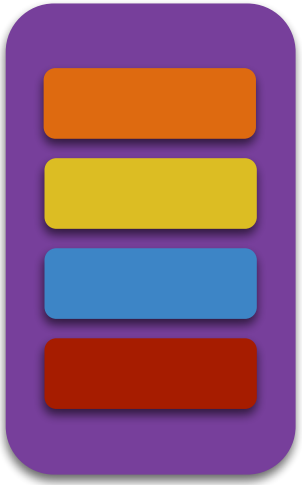
Importando una dependencia

Accediendo a nuestro archivo app.gradle

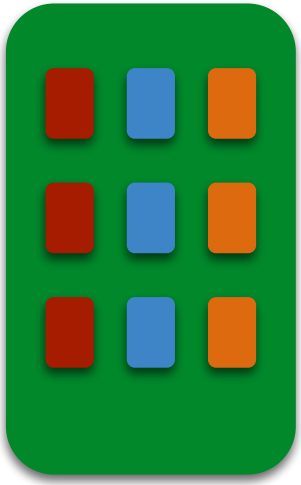

```
implementation 'com.android.support:recyclerview-v7:x.x.x'
```

Configurando un RecyclerView

LayoutManager



LinearLayoutManager



GridLayoutManager

Creando un RecyclerAdapter

Funcionamiento de un RecyclerView

Lista vacía
LinearLayoutManager

Creando un RecyclerAdapter

Funcionamiento de un RecyclerView

Lista vacía
LinearLayoutManager

```
RecyclerView.Adapter
  • getItemCount
  • onCreateViewHolder
  • onBindViewHolder
```

Clase Abstracta

Definición

```
abstract class Dragon() {
     fun fly() {
          println("\*/ -*- \*/ -*-")
     }

     fun roar() {
          println("Rawwwwwrrrrr")
     }

     abstract fun attack()
}
```

Clase Abstracta

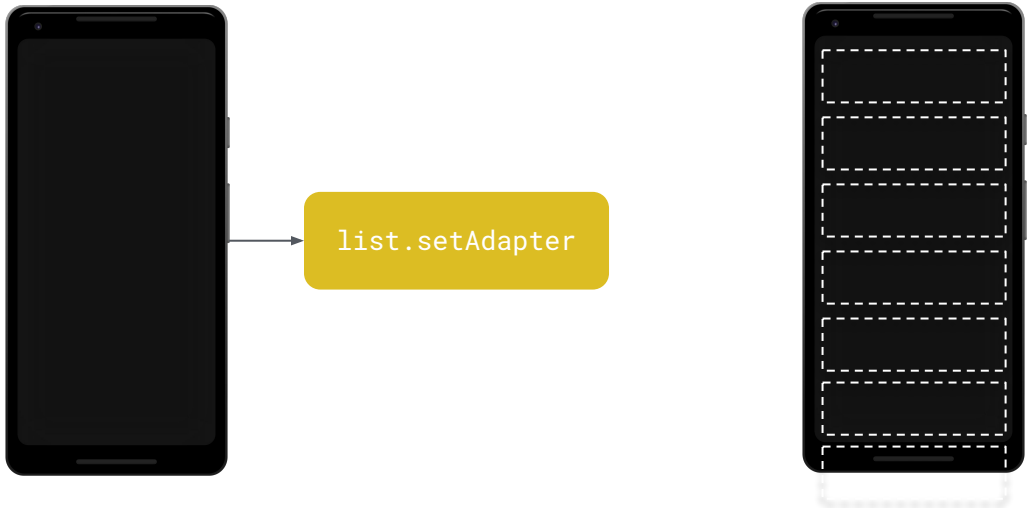
Implementación

```
abstract class Dragon() {
     fun fly() {
          println("\\*/ -*- \\*/ -*-")
     }

     fun roar() {
          println("Rawwwwwrrrrr")
     }

     abstract fun attack()
}
```

```
class FireDragon : Dragon() {
     override fun attack()
          println("Fire fire fire! ~~~")
     }
}

class IceDragon : Dragon() {
     override fun attack()
          println("Freeezee! ***")
     }
}
```

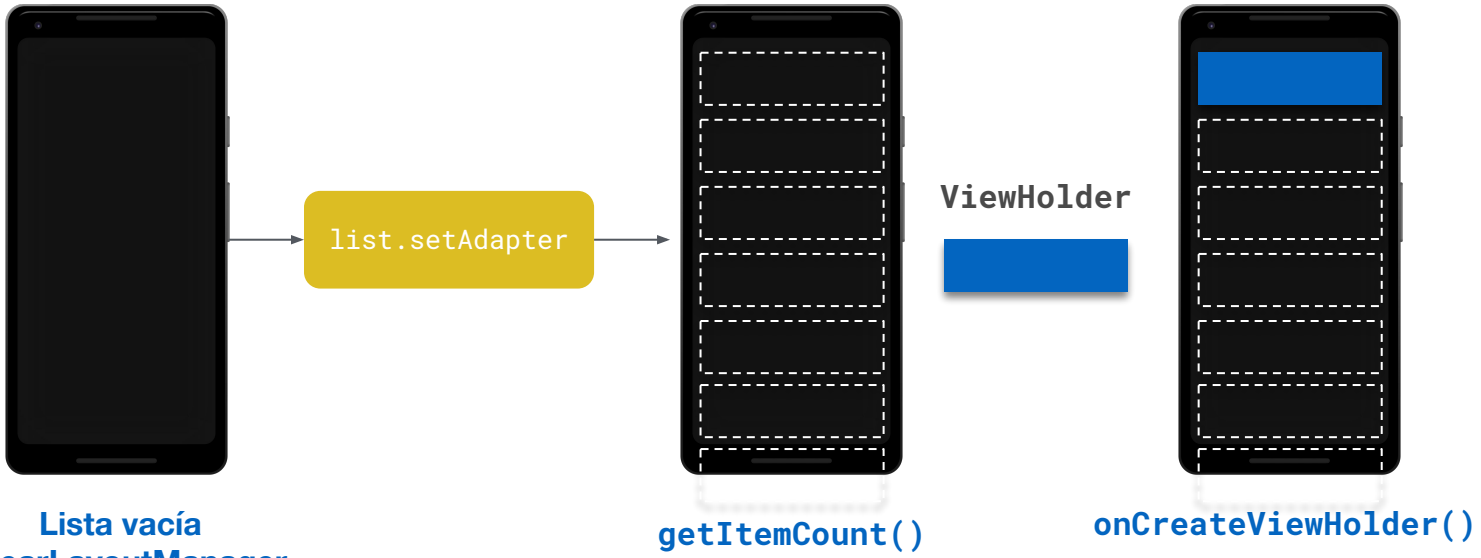
Creando un RecyclerAdapter

Funcionamiento de un RecyclerView



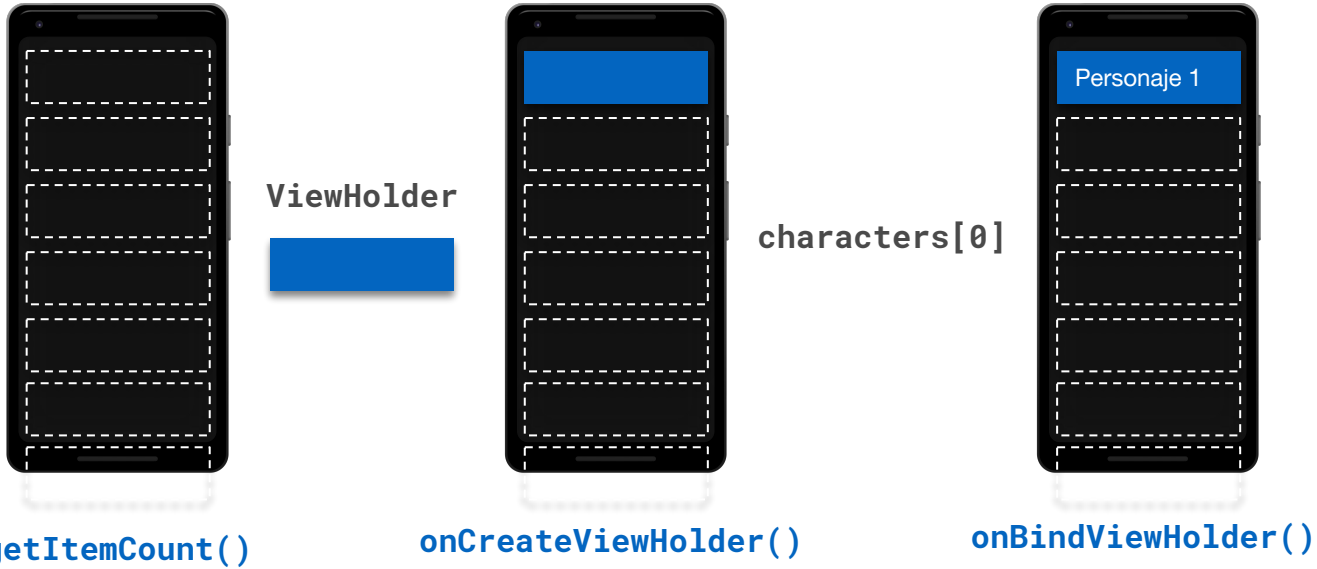
Creando un RecyclerAdapter

Funcionamiento de un RecyclerView



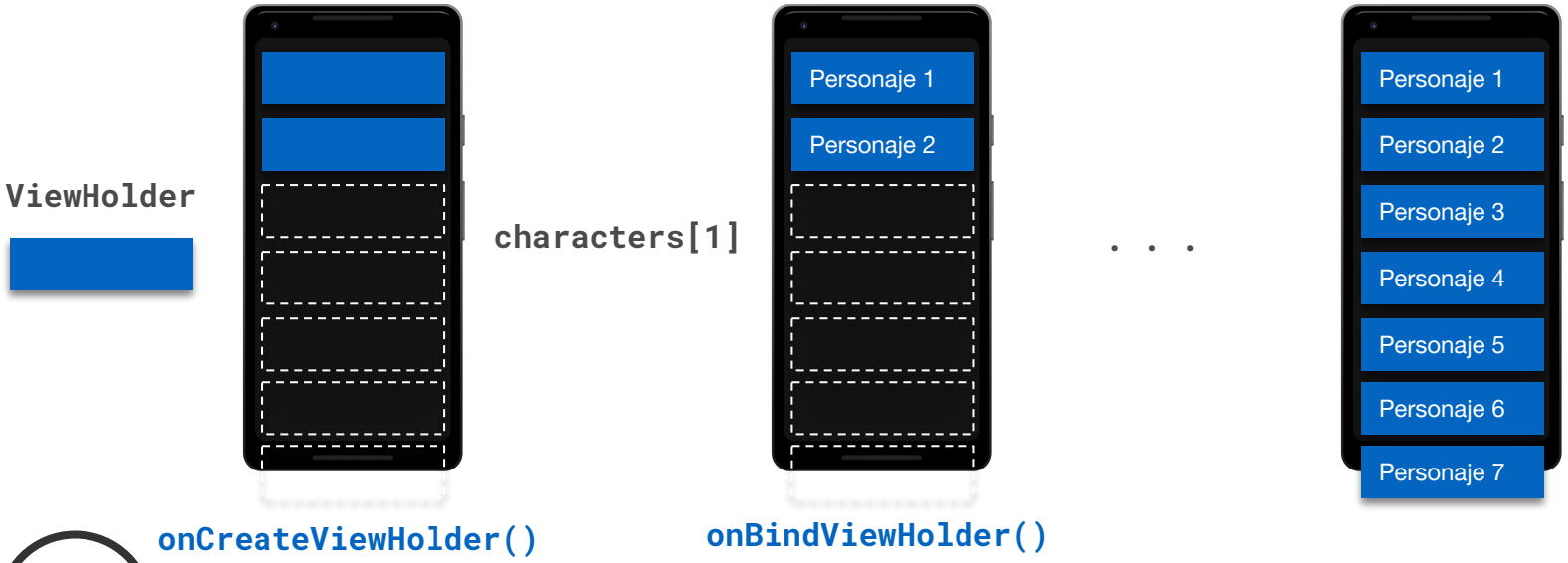
Creando un RecyclerAdapter

Funcionamiento de un RecyclerView



Creando un RecyclerAdapter

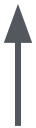
Funcionamiento de un RecyclerView



Creando un RecyclerAdapter

Funcionamiento de un RecyclerView

`onBindViewHolder()`



Swipe up

Generics

Abstracción de tipos

```
val smaug: FireDragon = FireDragon()
val viserion: FireDragon = FireDragon()

val army: List<FireDragon> = listOf<FireDragon>(viserion, smaug)
```

Generics

Abstracción de tipos

```
fun <DRAGON_TYPE : Dragon> armyAttack(army: List<DRAGON_TYPE>){
    for(dragon in army) {
        dragon.attack()
    }
}
```

Mutable o No Mutable

Colecciones de datos

```
listOf<Character>()
```

Mutable o No Mutable

Colecciones de datos

```
mutableListOf<Character>()
```

“Personaje 1”
“Personaje 2”
“Personaje 3”
“Personaje 4”
“Personaje 5”

Inferencia de tipos

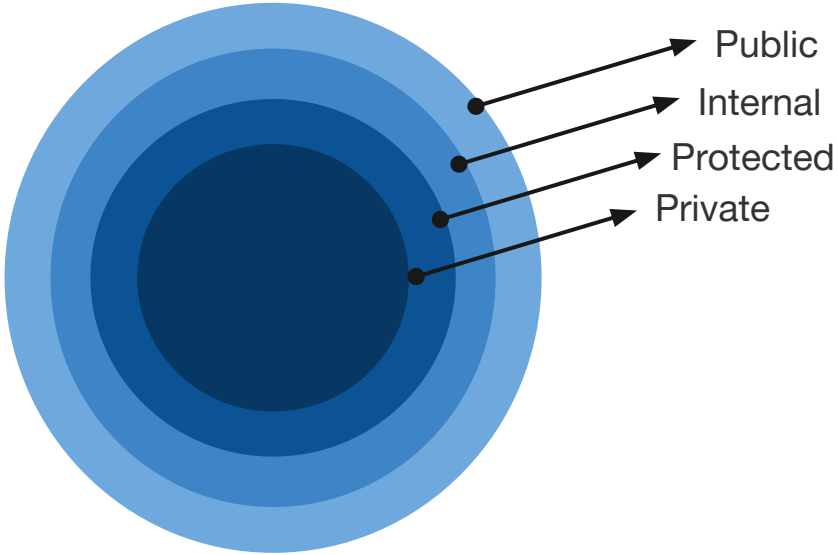
No, no es un tipado dinámico

```
var tiny = FireDragon()
```

```
tiny = IceDragon()
```

Público o Privado?

Niveles de visibilidad



Private

Nivel de clase

```
class RedMountainDragon: FireDragon() {
     override fun attack() {
          super.attack();
          rainOfFire();
     }

     private fun rainOfFire() {
          prinln("The droplets are on fire!")
     }
}
```

Private

Nivel de clase

```
class RedMountainDragon: FireDragon() {
     override fun attack() {
          super.attack();
          rainOfFire();
     }

     private fun rainOfFire() {
          prinln("The droplets are on fire!")
     }
}
```

```
val xin = RedMountainDragon()

xin.attack()

xin.rainOfFire()
```


Protected

Nivel de clase y subclases

```
class DarkDragon: Dragon()  {
     override fun attack() {
          println("Dark storm! ####")
     }

     protected fun reincarnate(): Dragon
     {
          return DarkDragon()
     }
}
```

Protected

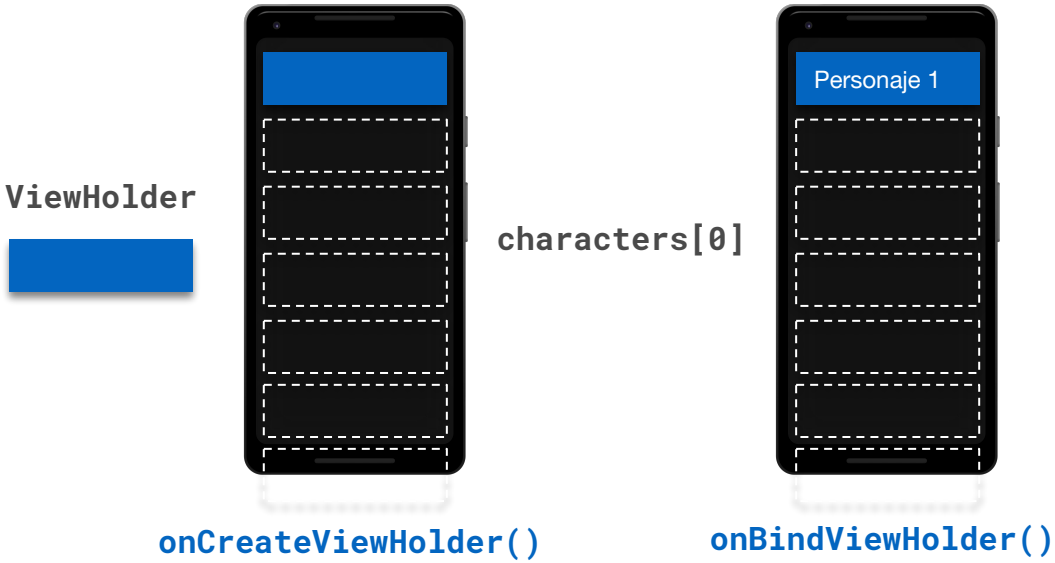
Nivel de clase y subclases

```
open class DarkDragon: Dragon()  {
     override fun attack() {
          println("Dark storm! ####")
     }

     open protected fun reincarnate(): Dragon {
          return DarkDragon()
     }
}
```

```
class BlackDungeonDragon: DarkDragon()  {

     fun sacrifice(): Dragon {
          return reincarnate();
     }

     override fun reincarnate(): Dragon {
          return IceDragon()
     }
}
```

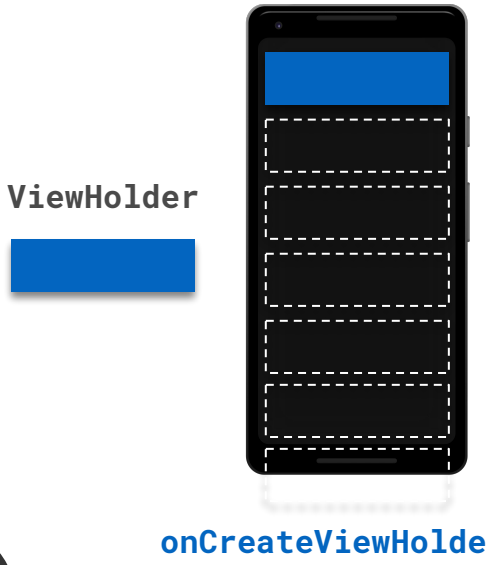
Creando nuestro ViewHolder

Nuestra plantilla de cada item de la vista



Creando nuestro ViewHolder

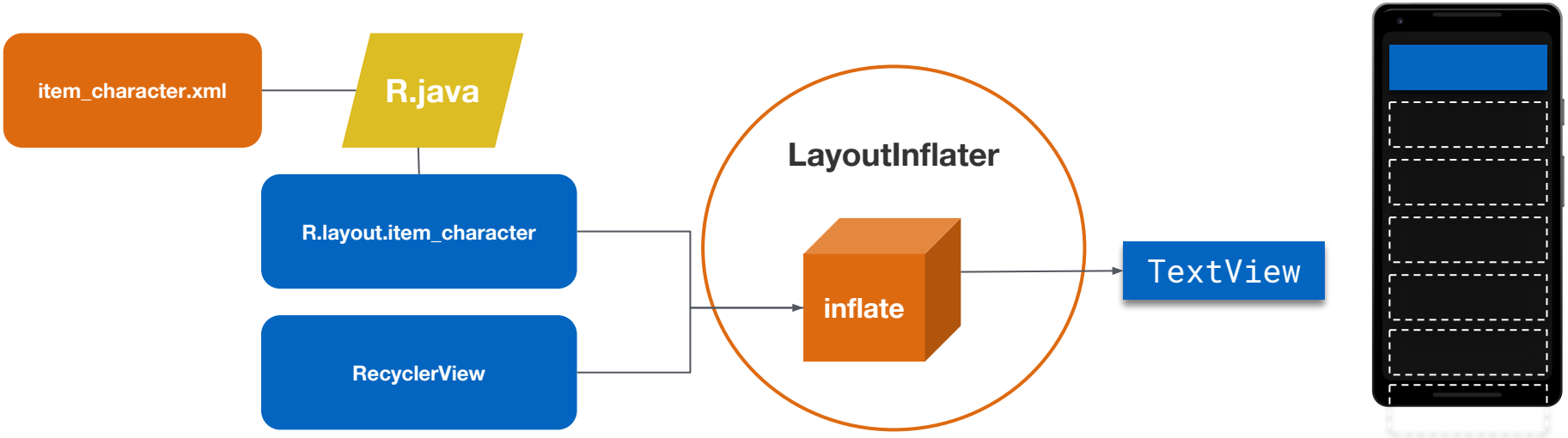
ToDo List



- ➔ Crear el archivo de vista que usará el ViewHolder
- ➔ Inflar ese archivo de vista
- ➔ Instanciar un ViewHolder a partir de ese archivo de Vista

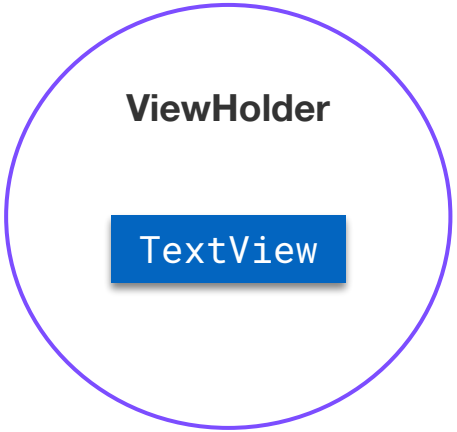
Proceso de Inflado de una vista

Cómo pasamos de un archivo XML a algo visible en la pantalla



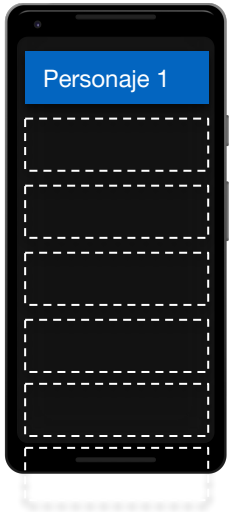
Instanciar el ViewHolder

Cómo pasamos de un archivo XML a algo visible en la pantalla



Conectando el modelo de datos con el ViewHolder

```
override fun onBindViewHolder(holder: CharacterViewHolder, position: Int)
```



`characters[position]`

Null safety

Evitando un null pointer exception de mil millones de dólares

?

Si ves este símbolo, significa que el objeto puede ser nulo. ***Cuidado***

Custom Setter

Evitando un null pointer exception de mil millones de dólares

```
abstract class Dragon {
    var age: Int = 0
    set(newAge) {
        if(newAge >= 0)
            field = newAge
    }
    //...
}
```

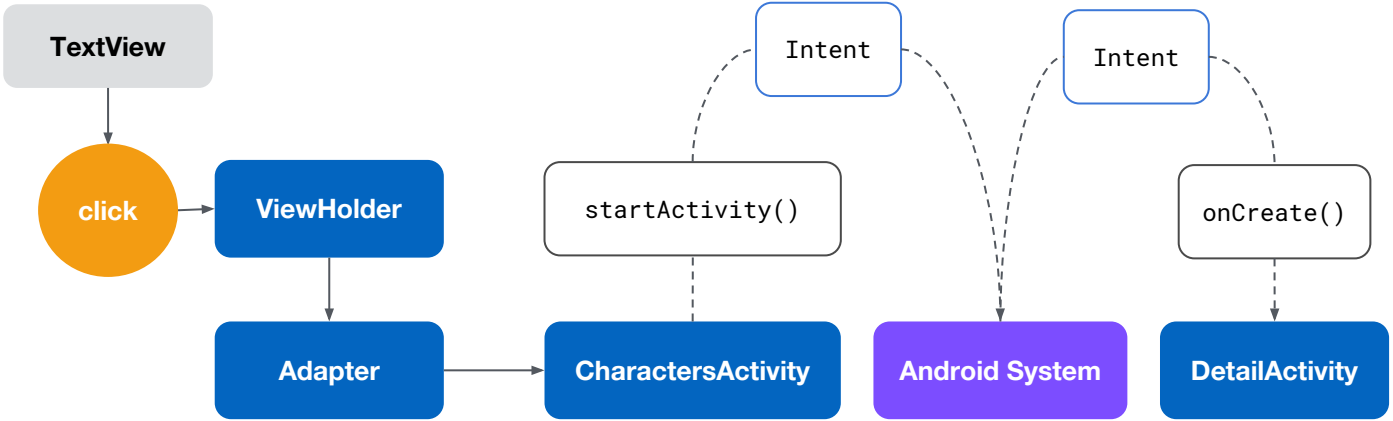
Notificaciones al adaptador

Actualizar los datos del adaptador para su visualización

- `notifyDataSetChanged()`
- `notifyItemInserted()`
- `notifyItemChanged()`
- `notifyItemMoved()`
- `notifyItemRemoved()`
- `notifyItemRangeInserted()`
- `notifyItemRangeChanged()`
- `notifyItemRangeRemoved()`
- `notifyItemRangeMoved()`

La práctica hace al maestro

Implementando un click listener dentro de nuestro RecyclerView



La práctica hace al maestro

ToDo list

- ➔ Añadir ClickListener al ItemView
- ➔ Comunicar evento a través de una lambda

Utilidades null safety

Operador let

```
?.let
```

*Si lo que está a la izquierda **no** es nulo, ejecuta lo siguiente*

Limpieza de código

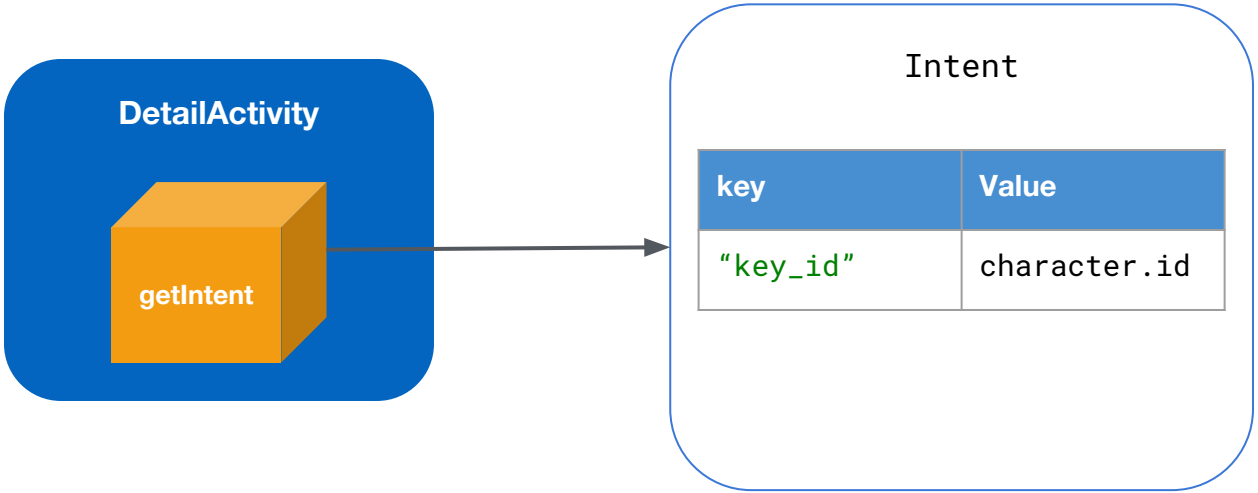
inicializador lazy

```
by lazy
```

La primera vez que llames a esta instancia, ejecuta previamente el bloque inicializador

Datos de una actividad a otra

putExtra()

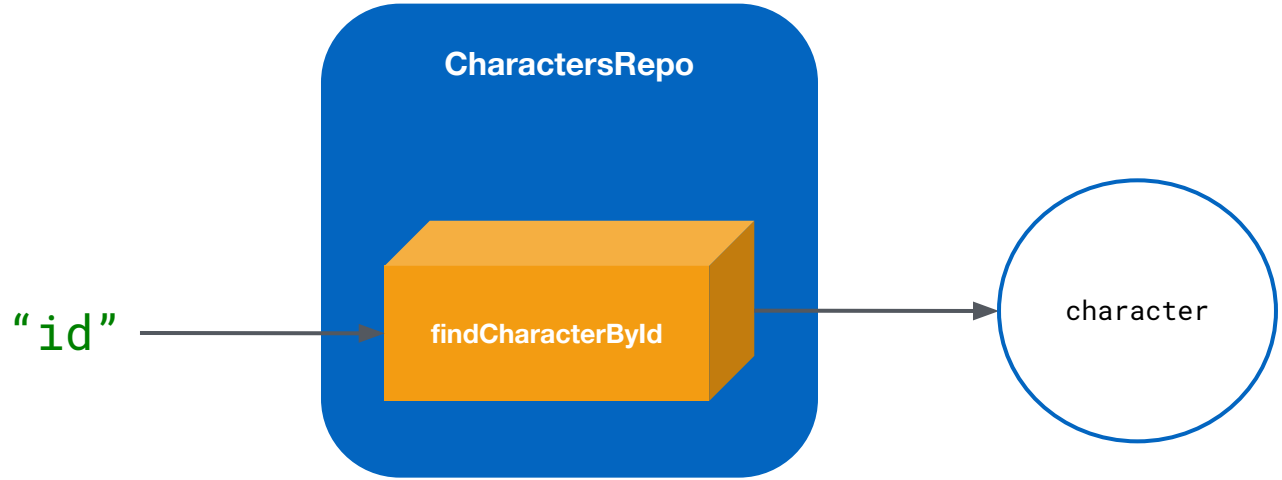


Debugging

Stop the code!

Recuperando datos del personaje

Operador find() y operador with



Utilizando Kotlin Extensions

No más findViewById()

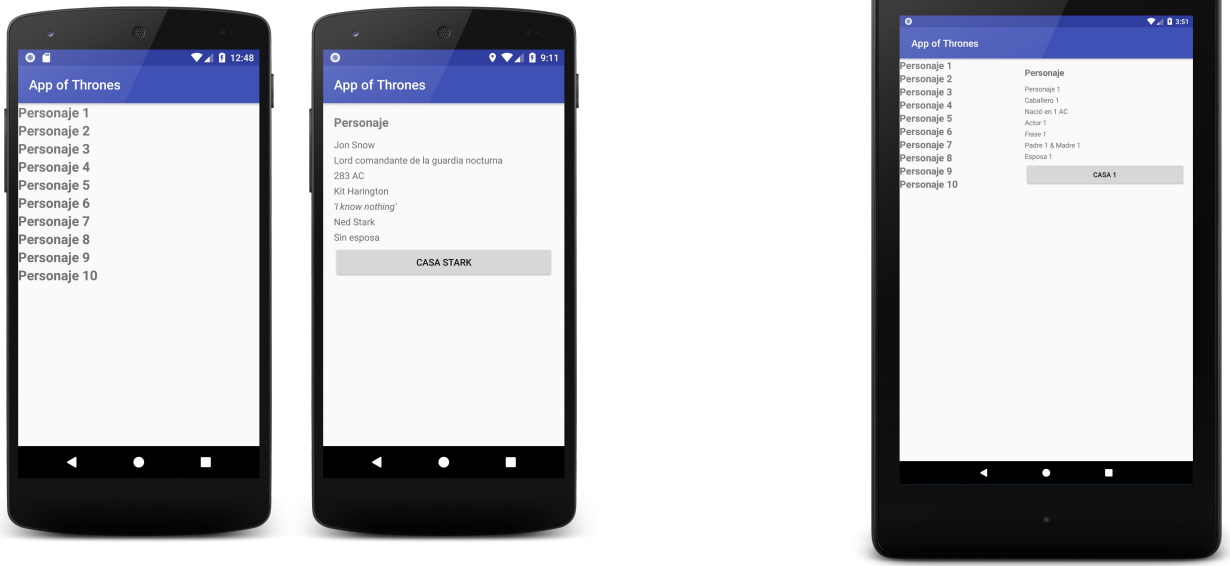
```
apply plugin: 'kotlin-android-extensions'
```

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

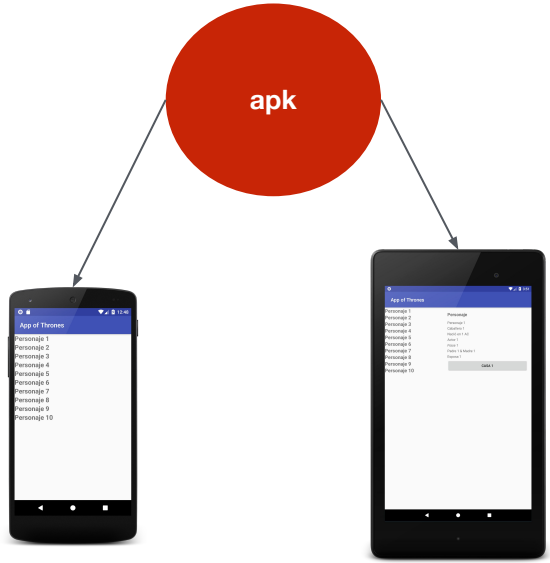
```
labelName.text = name
```

You’re the best!

Ahora con soporte para Tablet



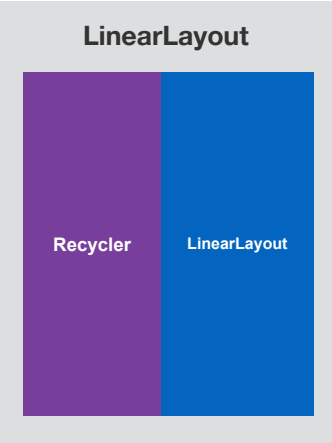
Cualificadores de configuración



- Idioma
- Orientación
- Modo nocturno
- Versión de Android
- Dimensiones de pantalla

Recursos para distintas configuraciones

- layout
  - activity_characters.xml
- layout-sw600dp
  - activity_characters.xml

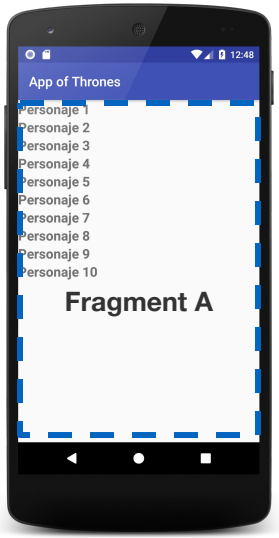


`android:layout_weight`

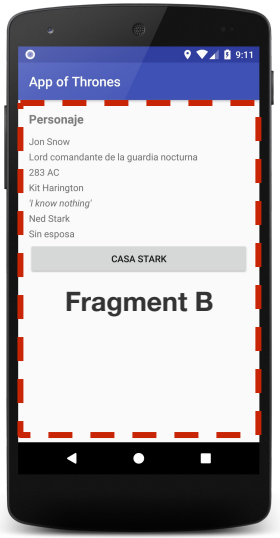
¿Y ahora?

Bienvenido a Fragments

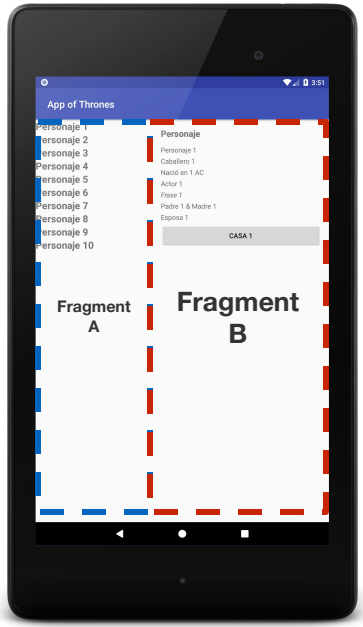
Reutilizando el código de nuestros controladores



CharactersActivity



DetailActivity



CharactersActivity

Creando un Fragmento

Reutilizando el código de nuestros controladores

- ➔ Los fragmentos siempre estarán ligados con una actividad
- ➔ Los fragmentos son parte de la vista de una actividad
- ➔ Los fragmentos declaran su propia vista

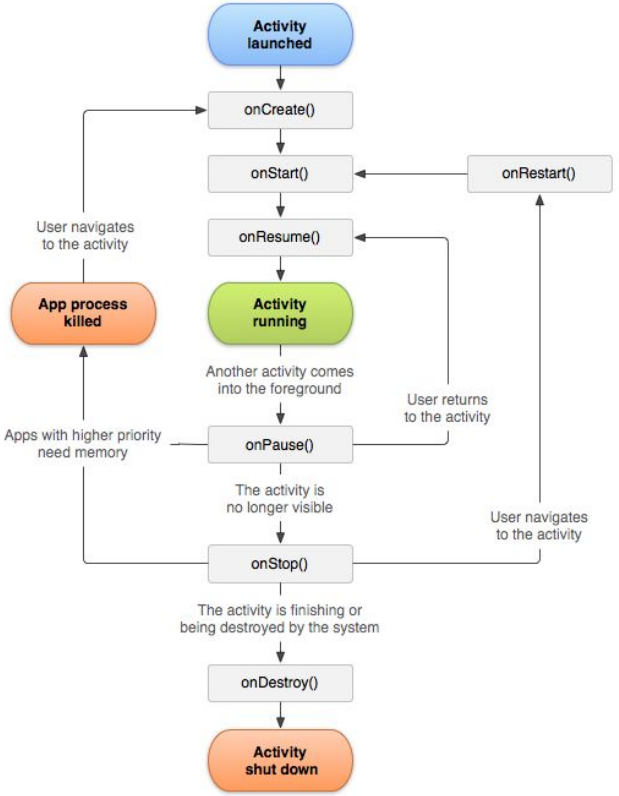
Creando un Fragmento

ToDo list

- ➔ Crear una clase que herede de fragment
- ➔ Inflar la vista del fragmento
- ➔ Crear una instancia de la clase del fragmento desde la actividad
- ➔ Añadir el fragmento a la UI de la actividad con ayuda del FragmentManager

El ciclo de vida de una actividad

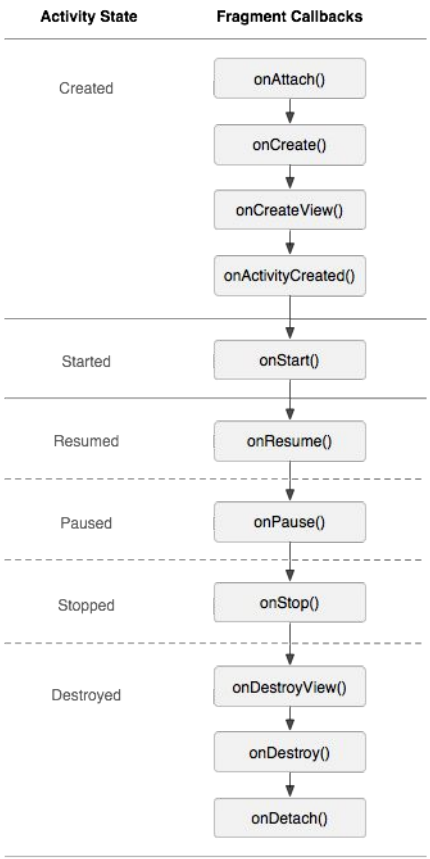
Comprobando los flujos principales



https://developer.android.com/guide/components/activities/activity-lifecycle.html

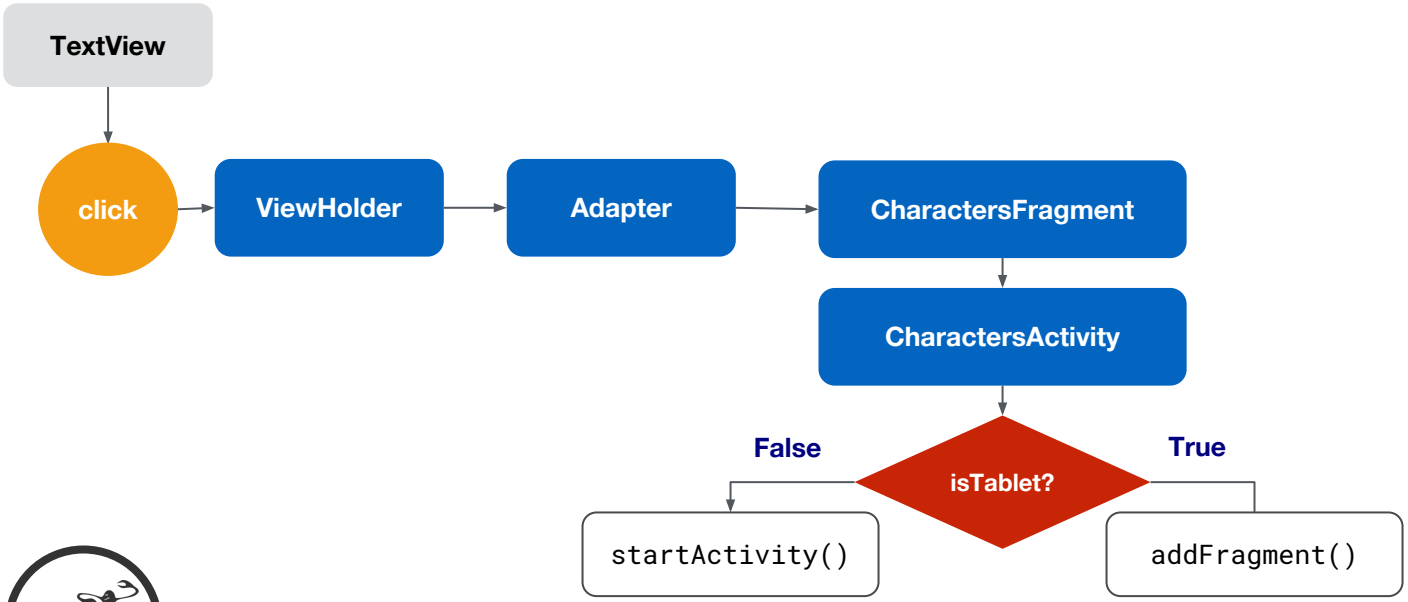
El ciclo de vida de un fragmento

Su relación con el ciclo de vida de una actividad



https://developer.android.com/guide/components/fragments.html

Comunicando un fragmento con una actividad



Interfaces en Kotlin

Composición de clases. Añadiendo habilidades a una clase

```
interface Transport {
     fun ride()
}
```

```
class Dragon() : Transport {
     //..
     override fun ride() {
          println("Fly with me!")
          fly()
     }
}
```

Poniendo en práctica el concepto de fragmentos

- ➔ Crear el fragmento DetailFragment
- ➔ Implementar lógica para añadir fragmento de detalle en caso de que el layout sea de una tableta

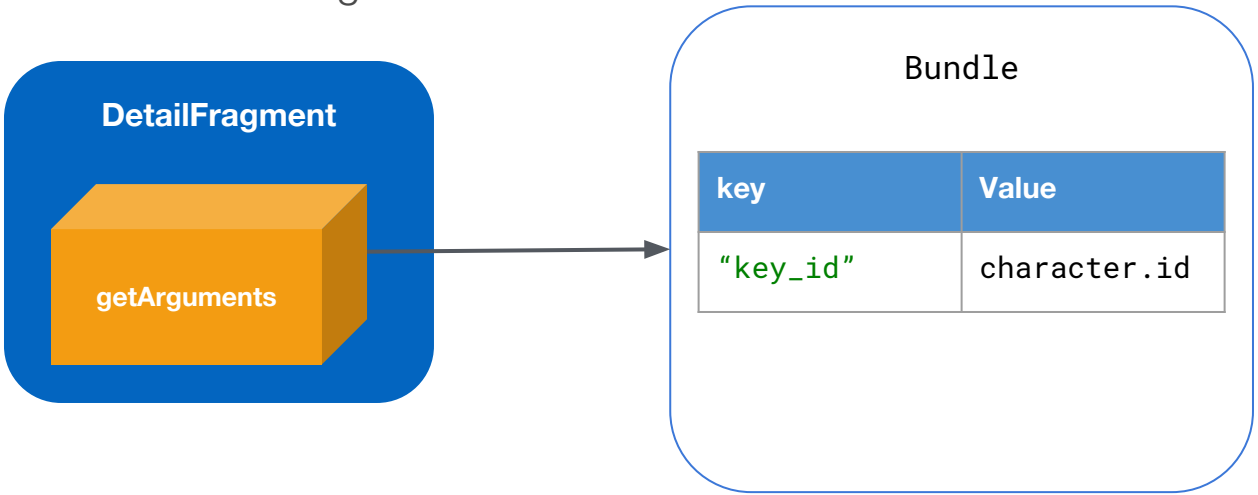
Funciones inline

Útiles para reusar bloques pequeños de código

```
private fun isDetailViewAvailable() = findViewById(R.id.detail_container) != null
```

Pasando argumentos a un Fragmento

Bundle dentro de fragmento



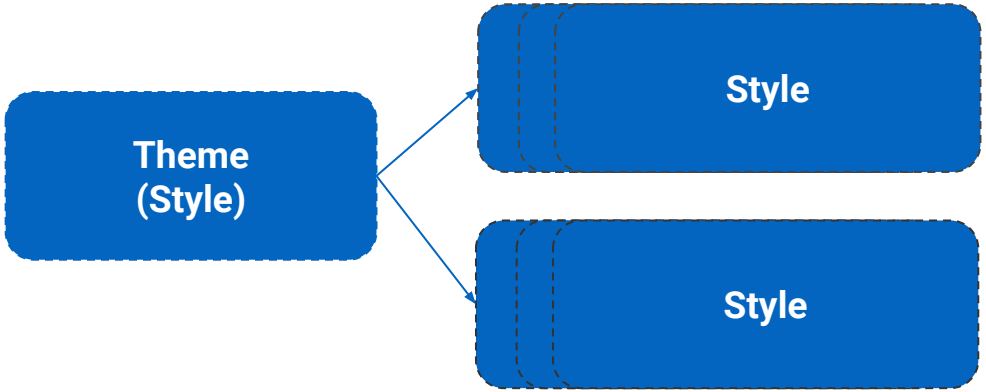
Métodos estáticos en una clase

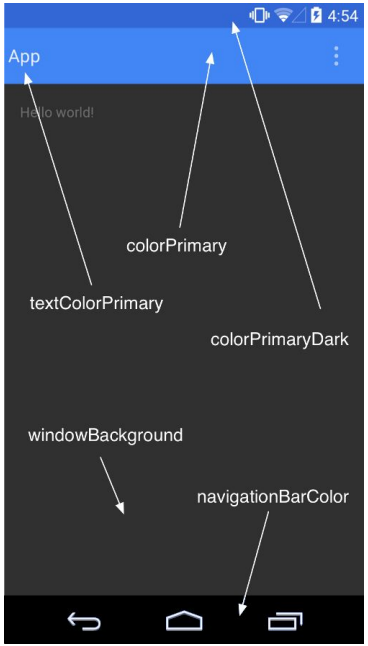
Companion Object

```
companion object {
    fun newInstance(characterId: String): DetailFragment {
        //..
    }
}
```

Cambiando el tema de nuestra app

Material Design Theme





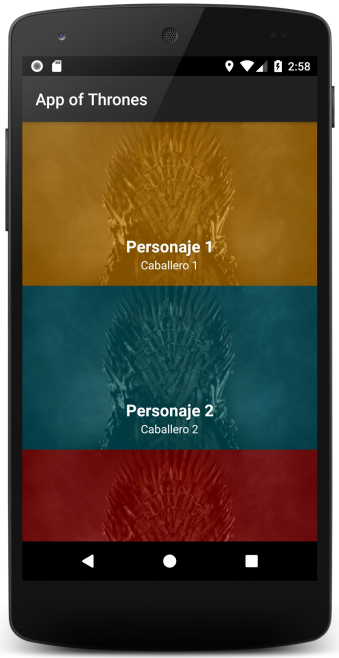
Usando RelativeLayout

Cambiando la vista de nuestros items

Añadir los recursos del siguiente directorio

https://goo.gl/2pYzvk

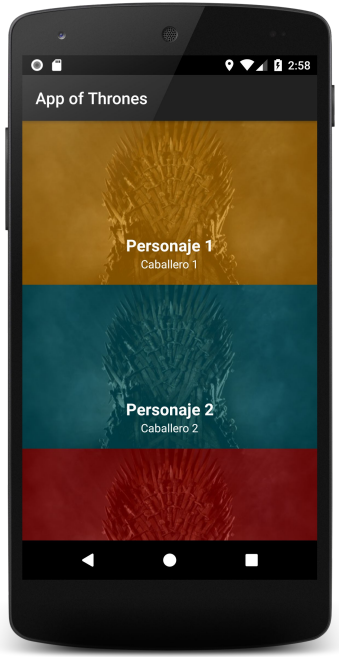
Descargar, copiar y pegar en la carpeta **res**



Usando RelativeLayout

Cambiando la vista de nuestros items

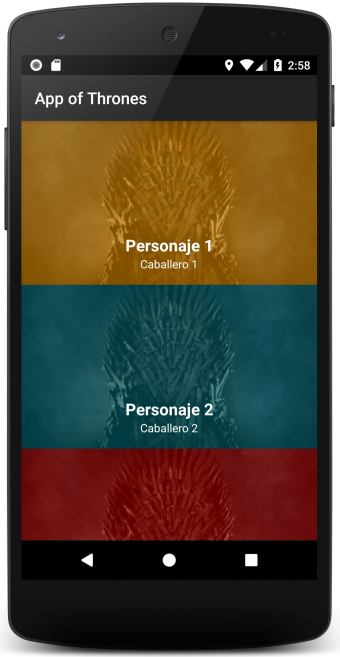
- `layout_above`
- `layout_below`
- `layout_toLeftOf`
- `layout_toRightOf`
- `layout_alignBottom`
- `layout_alignTop`
- `layout_alignRight`
- `layout_alignLeft`
- `layout_alignBaseline`



Cambiando el color de cada casa

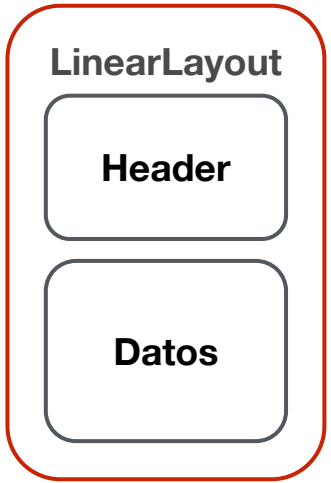
Companion Object

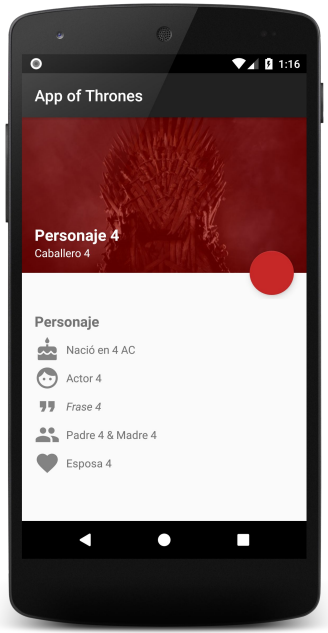
```
fun getOverlayColor(houseId: String): Int
fun getBaseColor(houseId: String): Int
```



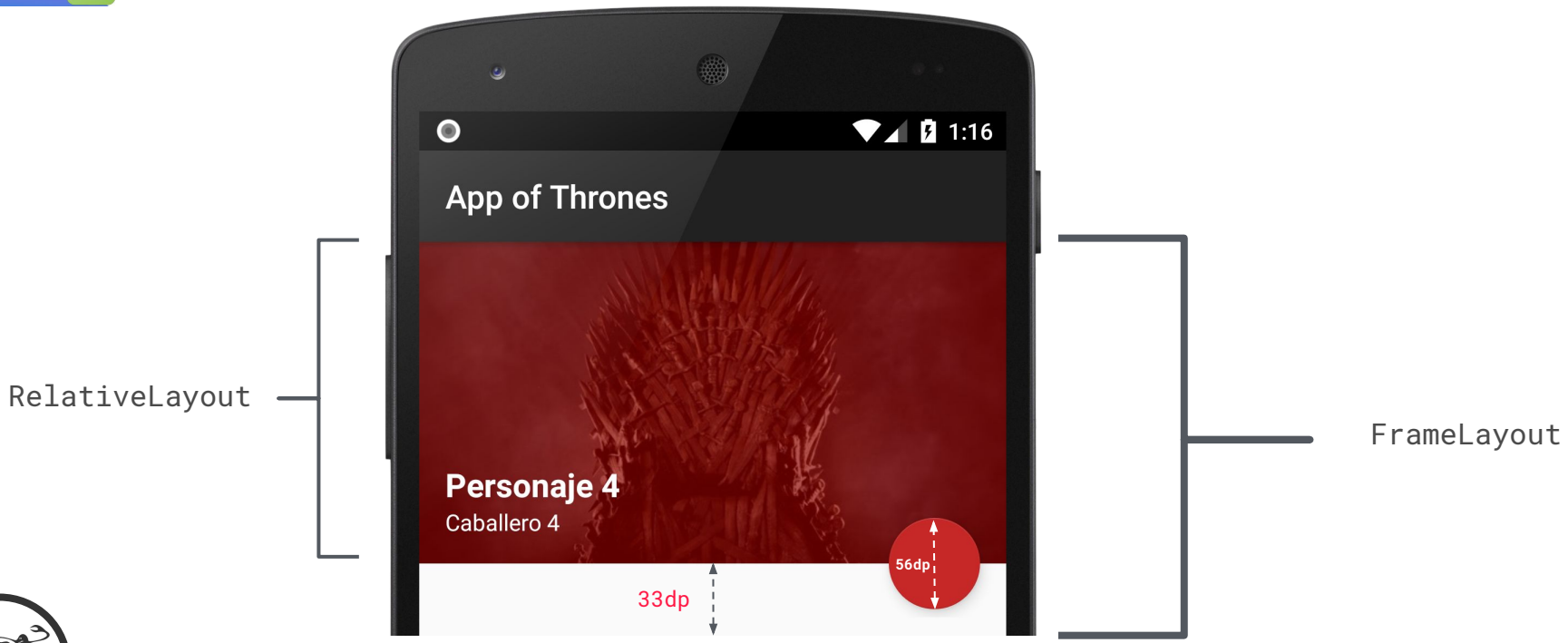
Una UI acorde a material design

Companion Object

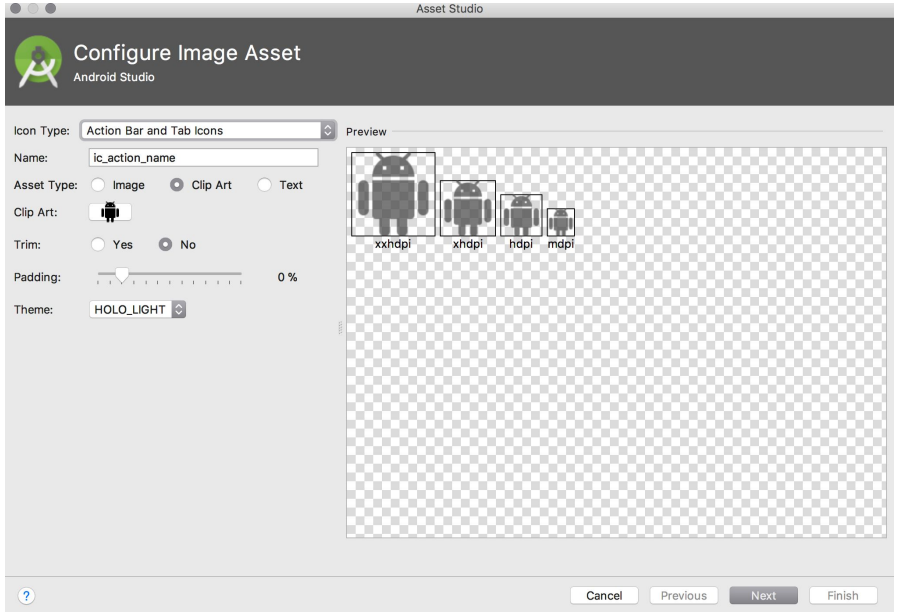


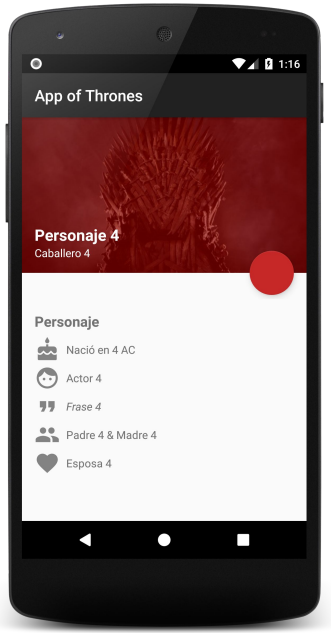


Header



Creación de iconos en Android Studio



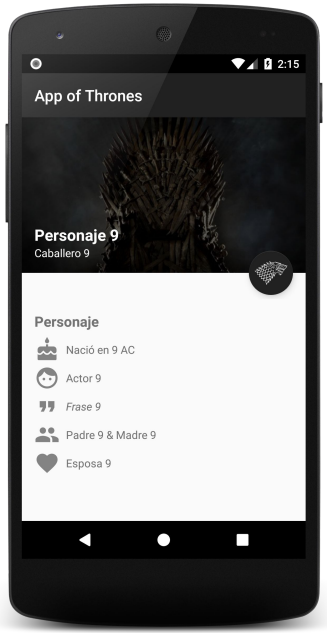


Añadiendo emblemas

Añadir los recursos del siguiente directorio

https://goo.gl/8XgrMr

Descargar, copiar y pegar en la carpeta **res**

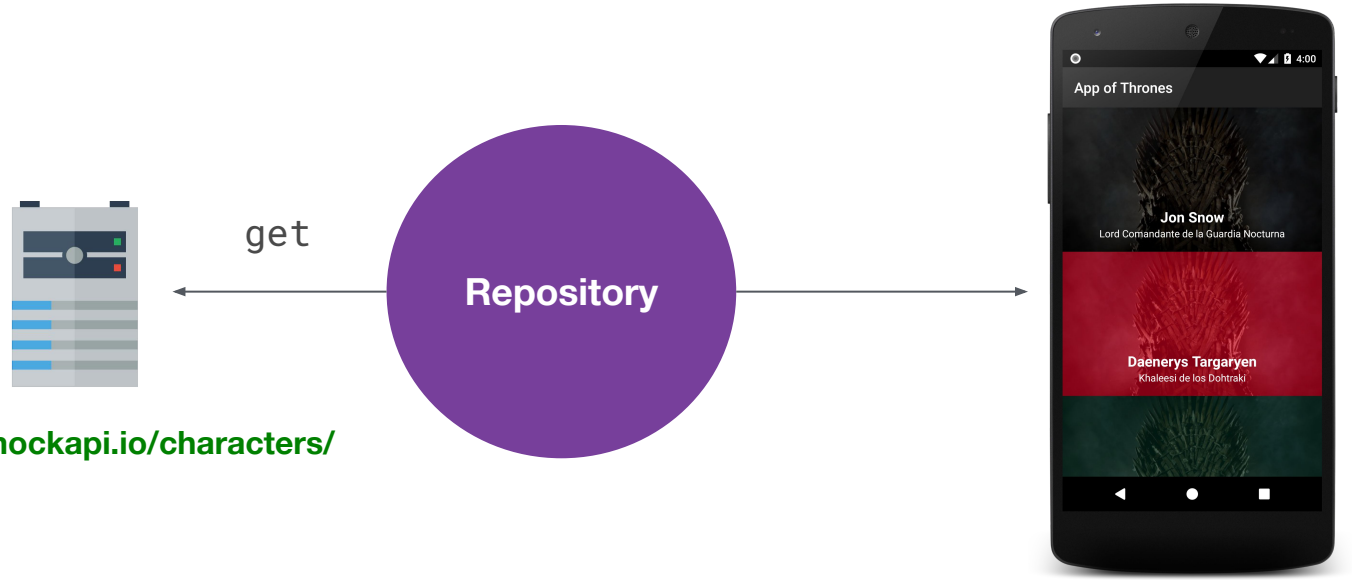


Scroll view

- layout
  - fragment_detail.xml
- layout-sw320dp-land
  - fragment_detail.xml
- layout-sw420dp-land
  - fragment_detail.xml

Arquitectura cliente servidor

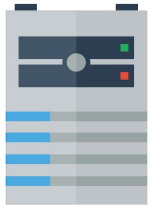
Comunicación a través del protocolo HTTP



http://apid.mockapi.io/characters/

Creando nuestro servidor de pruebas

Utilizando una mockAPI para simular nuestro servidor



http://apid.mockapi.io/characters/

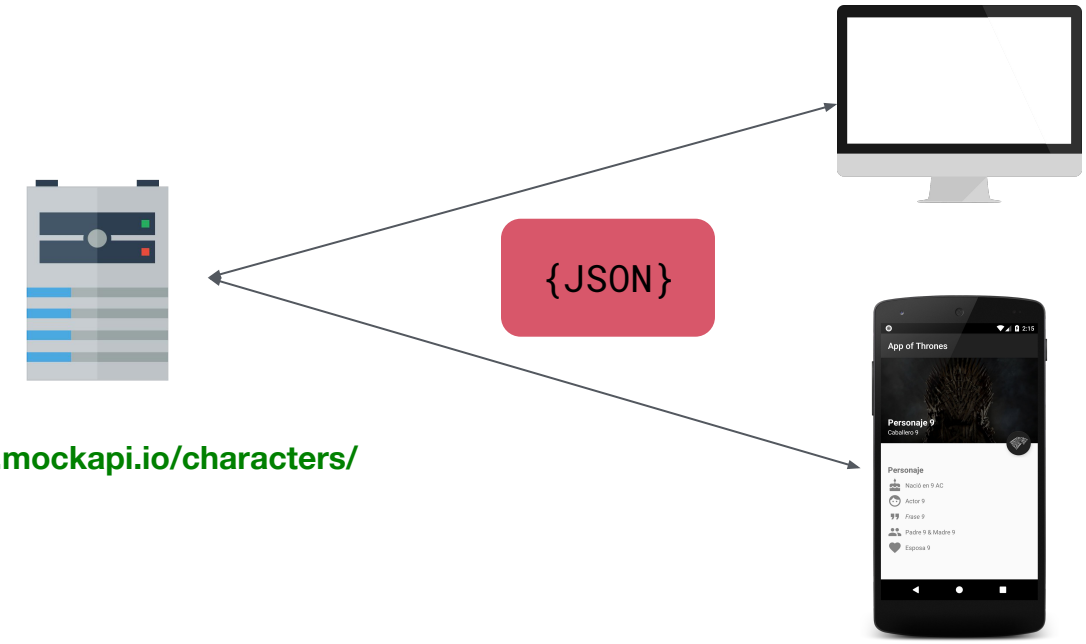
Ingresar a

http://www.mockapi.io/

Crear una cuenta

Creando nuestro endpoint

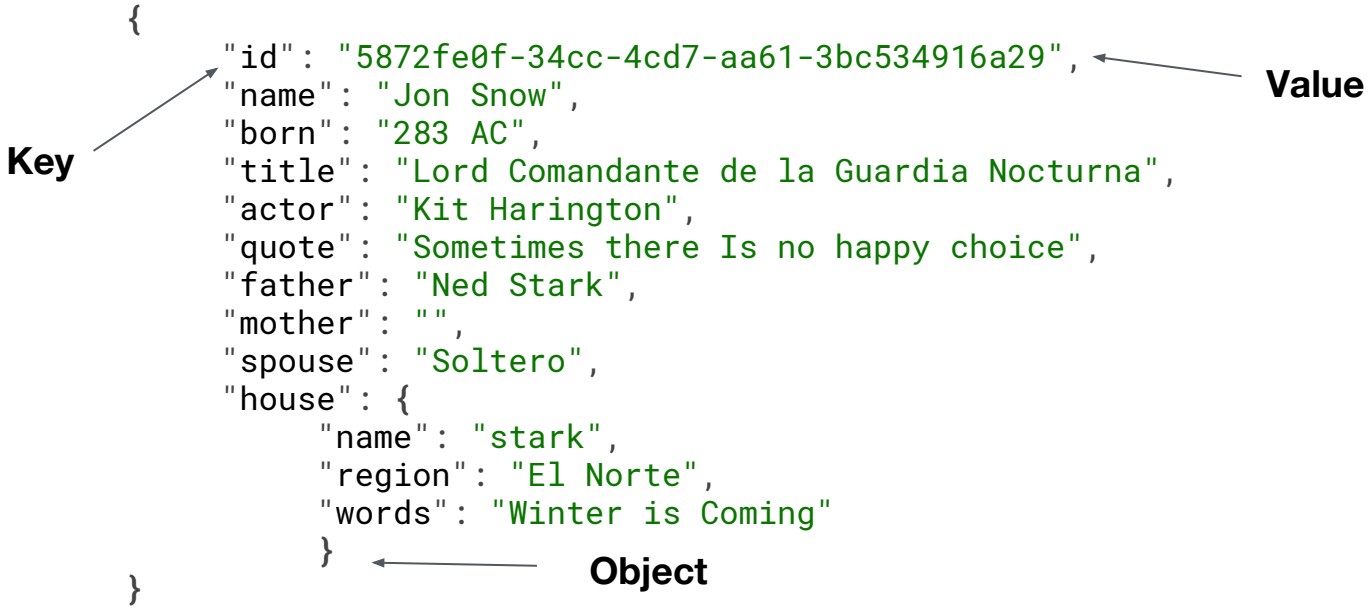
Enviando datos via JSON



http://apid.mockapi.io/characters/

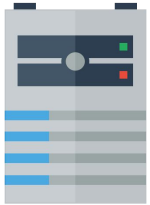
Javascript Object Notation

Sintaxis



Creando nuestro endpoint

Enviando datos via JSON



http://apid.mockapi.io/characters/

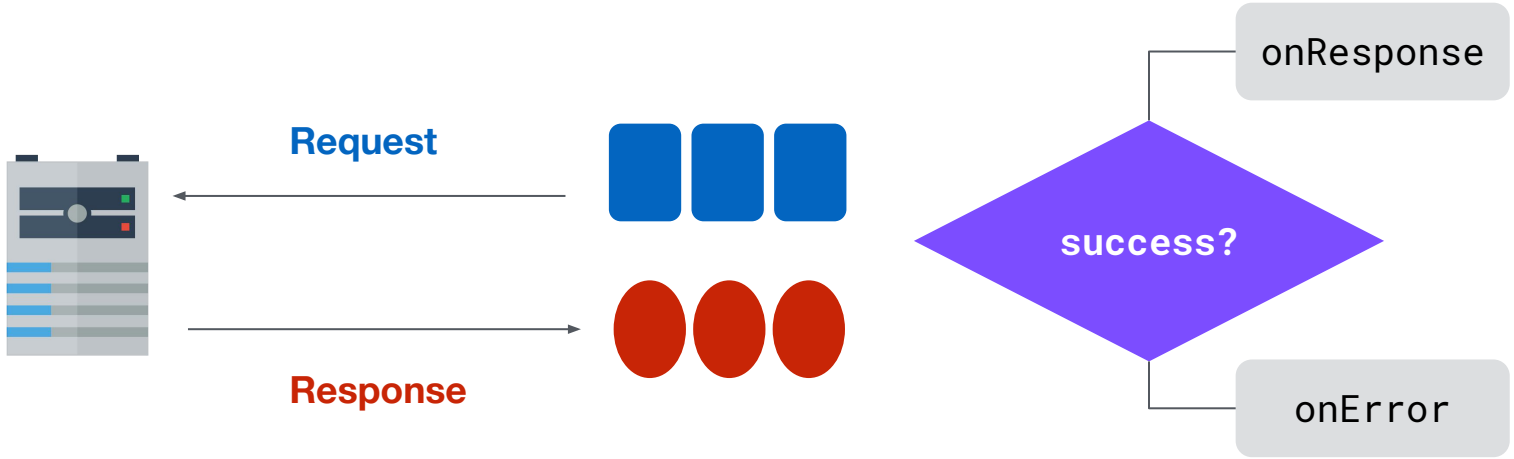
Acceder el gist

https://goo.gl/VQjQsX

Copiar y pegar el archivo JSON en mockAPI

Volley

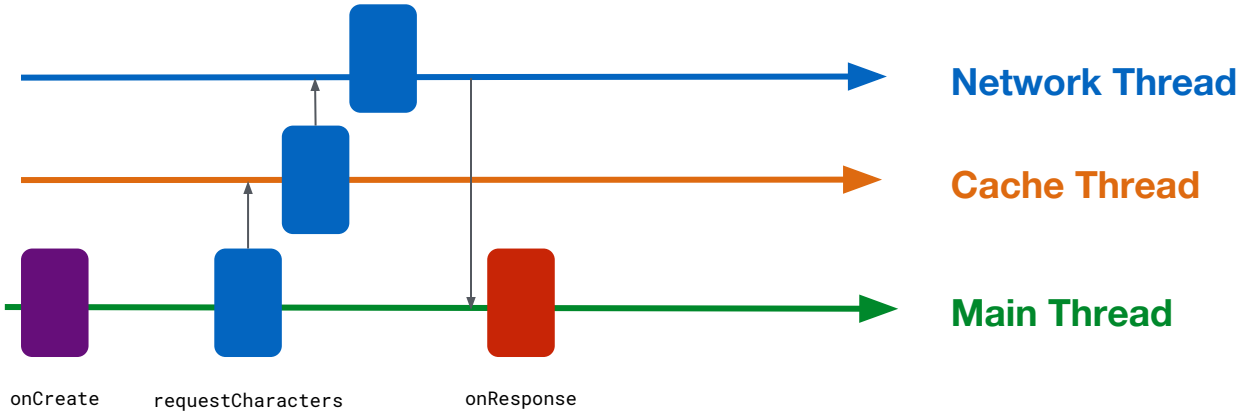
Escribiendo una petición desde nuestro cliente Android




```
implementation 'com.android.volley:volley:1.1.0'
```

Tareas asíncronas en android

Escribiendo una petición desde nuestro cliente Android



De JSON a un Character

Qué hacer en caso de una respuesta exitosa

Una UI amigable para el usuario

Barra de carga y botón de reintento


```
progressBar.visibility = View.INVISIBLE
layoutError.visibility = View.VISIBLE
```

Picasso

Cargando imágenes desde una URL

```
Picasso.get()
    .load(value.img)
    .placeholder(R.drawable.test)
    .into(itemView.imgCharacter)
```

```
implementation 'com.squareup.picasso:picasso:x.x.x'
```

Creando un DialogFragment

Poniendo en práctica nuestros conocimientos de fragmentos

- ➔ Crear interfaz del dialog
- ➔ Crear clase que herede de DialogFragment
- ➔ Inflar la vista dentro del método OnCreateDialog
- ➔ Crear el diálogo de alerta
- ➔ Mostrarlo con ayuda del FragmentManager

KEEP CODING

Creando el icono de nuestra app

El último detalle

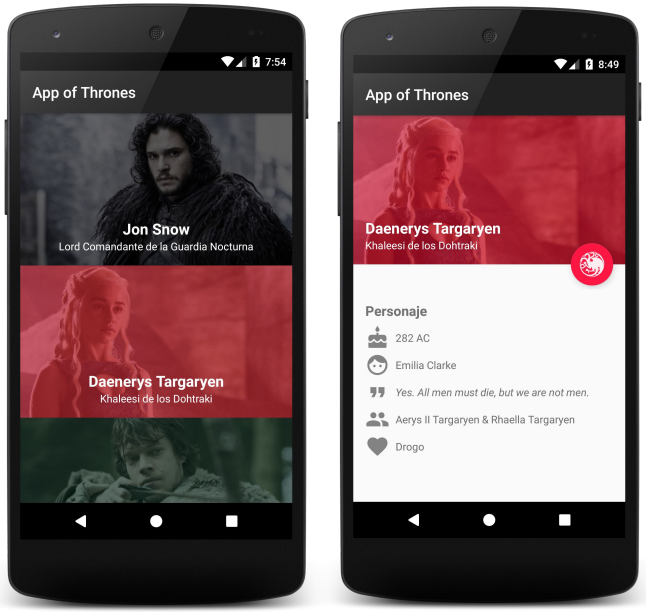
Descarga la imagen

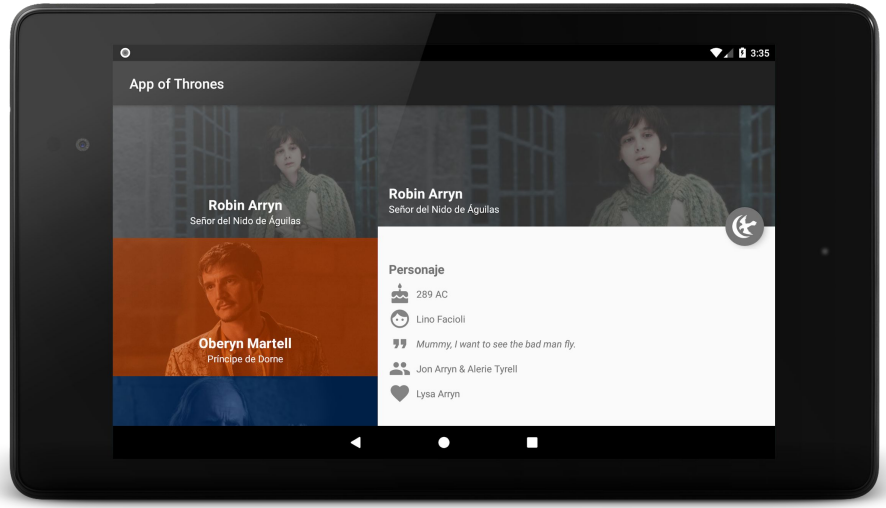
https://goo.gl/KrTRu3

Generemos nuestro icono

KEEP CODING

Nuestra aplicación está lista





Felicidades!

App of Thrones

Curso de desarrollo de aplicaciones Android en Kotlin desde 0

Pedro Antonio Hernández López
Software Developer at Bunsan.io
@silmood

KEEP CODING