Laboratori 1: Projecte Integrat de Software

Projecte Integrat de Software UB 2019/20
Grau d'Enginyeria informàtica

Gerard Puch Camacho

Apps i Android Studio

Pel disseny d'una app, cal tenir en compte els següents aspectes fonamentals.

1. Components d'una aplicació
2. Cicle de vida d'una aplicació
3. Compatibilitat de dispositius
4. Execució de l'aplicació
5. Recursos de l'aplicació
6. Manifest

Components d'una aplicació:

Component: Punt d'entrada pel qual s'accedeix a l'aplicació, 4 tipus:

1.Activitats: Una activitat és el punt d'entrada de l'interacció amb el usuari. Representa una pantalla individual amb una interfície d'usuari. Les activitats s'implementen com a subclasse de la class activity.

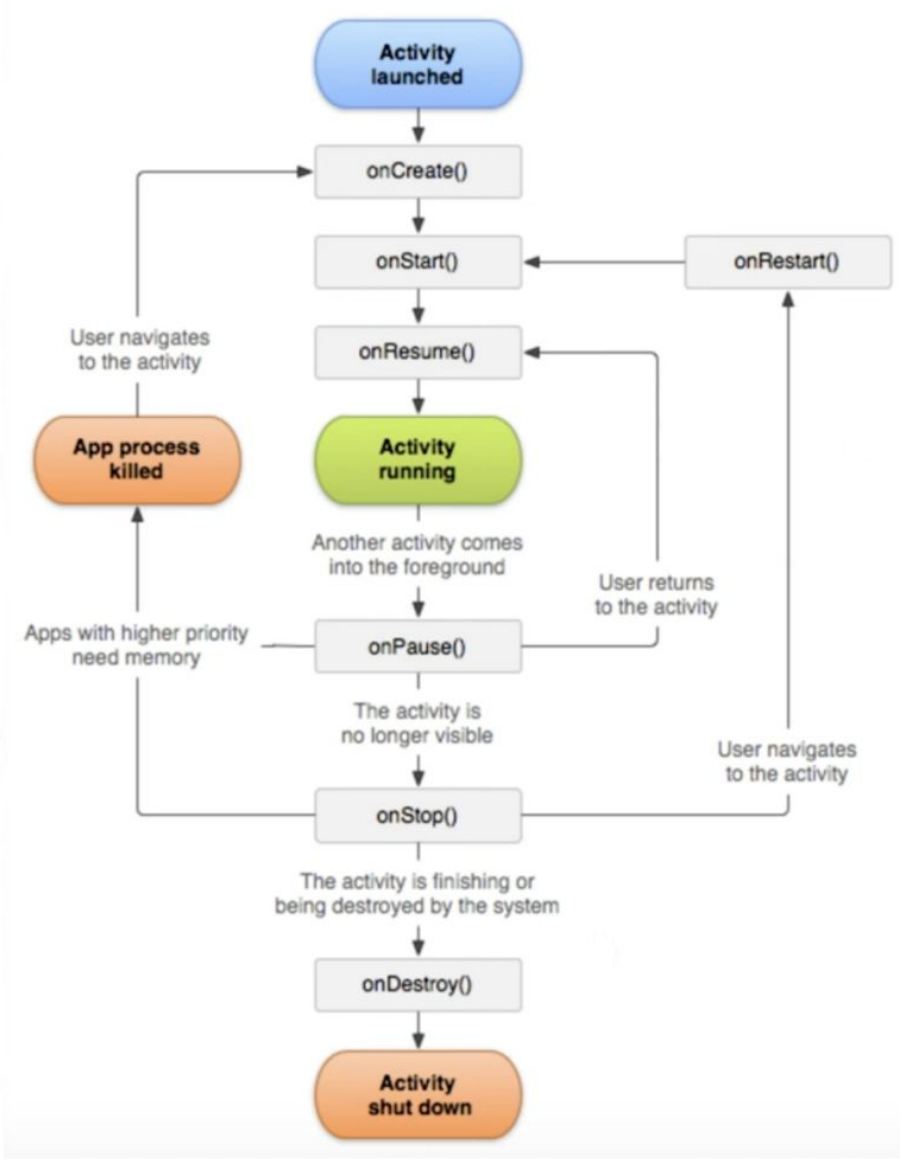
2.Serveis: És un punt d'entrada general que permet mantenir l'execució d'una aplicació en segon pla per diversos motius. És un component que s'executa en segon pla per realitzar activitats d'execució prolongada o per realitzar tasques de procesos remots. (Exemple: Spotify.)

3.Receptors d'emissió: És un component de possibilita que el sistema entregui events a l'aplicació fora d'un flux d'usuaris habituals. Això permet que l'aplicació respongui a anuncis d'emissió de tot el sistema. Aquest tipus de components no tenen interfície propia, simplement actuen com portes d'enllaç.

4.Proveidors de contingut: Administra un conjunt compartit de dades de l'aplicació que es poden emmagatzemar en un sistema d'arxius, en una base de dades SQlite, a la web o a qualsevol recurs d'emmagatzematge local a la que tingui accés la teva aplicació.

Cicle de vida d'una aplicació

Les apps tenen diferents processos. És important definir un mock-up inicial i ordenar tots els processos amb els seus respectius layouts. Es pot dividir en 3 procesos cíclics (**onResume**, **onStart**, **onCreate**) depenent de la interfície i els procesos en marxa.



En les funcions del codi, cal definir les següents funcions. És altament recomanable utilitzar la nomenclatura presentada a continuació per definir les activitats:

- **onCreate():** Es crea l'activitat, generada per l'assistent d'Android, defineix tot el que necessita l'activitat.
- **onStart():** Una vegada generada, l'activitat passa a pantalla.
- **onResume():** L'activitat està en pantalla y requereix la interacció de l'usuari.
- **onPause():** Una Altra activitat ocupa la pantalla principal, l'activitat iniciada queda en Standby.
- **onStop():** Activitat a segon pla.
- **onDestroy():** L'activitat es tanca (break), s'alliberen recursos.

Execució de l'aplicació

Podem executar l'aplicació en un dispositiu real o en un emulador generat pel propi Android Studio

Dispositu real:

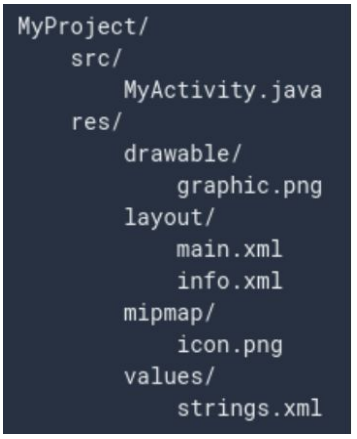
1. Seleccionar l'app d'execució.
2. Seleccionar en la barra d'eines el dispositiu on voleu realitzar l'execució de l'aplicació.
3. Una vegada seleccionats els dos paràmetres anteriors, prémer RUN i el propi programa instalarà l'aplicació.

Emulador:

1. Selecciona Tools>AVD Manager, selecciona el tipus de virtual device.
2. Selecciona el tipus de Hardware en el que desenvoluparàs la teva aplicació.
3. Selecciona la System Image (Android 9.0, Android, 7.1, etc). Recorda que has de seleccionar un OS que sigui compatible amb el seleccionat previament quan vàres crear el projecte Android.

Recursos de l'aplicació

Els recursos són els arxius adicionals i el contingut estàtic que utilitza el teu codi, com els mapes de bits, definicions de disseny, strings d'interfície d'usuari, etc.



Aspectes a tenir en compte:

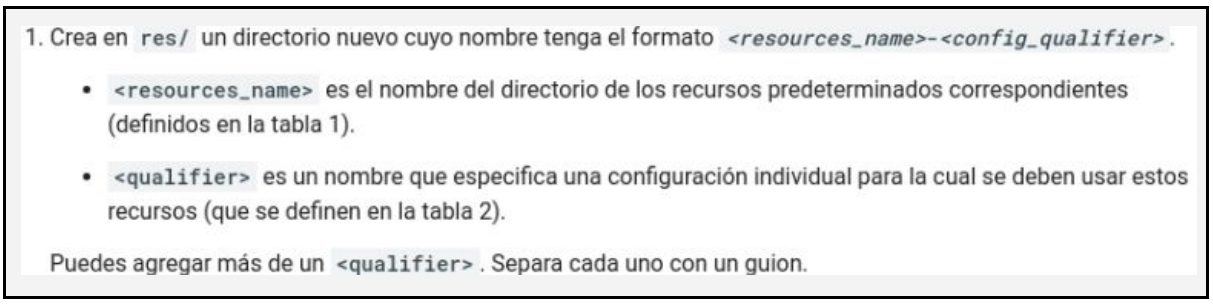
- Cal mantenir-los externalitzats.
- Organitzats a la classe R del teu codi.

Tota la informació més detallada a:
https://developer.android.com/ guide/topics/resources/providing-resources

Tipus de recursos predeterminats:

Directorio	Tipo de recurso
`animator/`	Archivos XML que definen animaciones de propiedades.
`anim/`	Archivos XML que definen animaciones de interpolación de movimiento. En este directorio, también se pueden guardar animaciones de propiedades, pero se prefiere el directorio `animator/` para las animaciones de propiedades, a fin de distinguir entre los dos tipos.
`color/`	Archivos XML que definen una lista de estados de colores. Consulta la sección Recurso de lista de estado de colores.
`drawable/`	Archivos de mapas de bits (`.png`, `.9.png`, `.jpg` y `.gif`) o archivos XML que se han compilado en los siguientes subtipos de recursos de elemento de diseño: • Archivos de mapas de bits • Nueve parches (mapas de bits reajustables) • Listas de estados • Formas • Elementos de diseño de animaciones • Otros elementos de diseño Consulta la sección Recursos dibujables.
`mipmap/`	Archivos de elementos de diseño para diferentes densidades de los íconos de selectores. Para obtener más información sobre la administración de los íconos de selectores con carpetas `mipmap/`, consulta la sección Información general sobre la administración de proyectos.
`layout/`	Archivos XML que definen el diseño de una interfaz de usuario. Consulta la sección Recurso de diseño.
`menu/`	Archivos XML que definen menús de aplicaciones, como un menú de opciones, un menú contextual o un submenú. Consulta la sección Recurso de menú.
`raw/`	Archivos arbitrarios para guardar sin procesar. Para abrir estos recursos con un objeto `InputStream` sin procesar, llama a `Resources.openRawResource()` con el ID del recurso, que es `R.raw.filename`. Sin embargo, si necesitas acceder a los nombres de los archivos originales y a la jerarquía de archivos, puedes considerar la posibilidad de guardar algunos recursos en el directorio `assets/` (en lugar de `res/raw/`). A los archivos de `assets/` no se les asigna un ID de recurso, por lo cual puedes leerlos solamente mediante `AssetManager`.
`values/`	Archivos XML que contienen valores simples, como strings, valores enteros y colores. Los archivos de recursos XML en otros subdirectorios `res/` definen un único recurso basado en el nombre del archivo XML, mientras que los archivos del directorio `values/` describen varios recursos. En el caso de un archivo de este directorio, cada campo secundario del elemento `<resources>` define un único recurso. Por ejemplo, un elemento `<string>` crea un recurso `R.string`, y un elemento `<color>` crea un recurso `R.color`. Dado que cada recurso se define con su propio elemento XML, puedes asignar el nombre que desees al archivo y colocar diferentes tipos de recursos en un archivo. Sin embargo, para mayor claridad, es recomendable que coloques tipos de recursos únicos en diferentes archivos. Por ejemplo, a continuación, se incluyen algunas convenciones de asignación de nombres de archivos para los recursos que puedes crear en este directorio: • arrays.xml para matrices de recursos (matrices escritas). • colors.xml para valores de color. • dimens.xml para valores de dimensión. • strings.xml para valores de strings. • styles.xml para estilos.
`xml/`	Archivos XML arbitrarios que se pueden leer en tiempo de ejecución llamando a `Resources.getXML()`. Aquí se deben guardar diversos archivos de configuración XML, por ejemplo, una configuración que permite búsqueda.
`font/`	Archivos de fuentes, con extensiones como `.ttf`, `.otf` o `.ttc`, o archivos XML que incluyan un elemento `<font-family>`. Para obtener más información sobre las fuentes como recursos, ve a Fuentes en XML.

Recursos alternatius:
En algunes circumstàncies, es requereix de recursos molt específics segons el terminal on s'executi l'aplicació. Els diferents idiomes, les diferents qualitats d'imatge o diferents connexions de xarxa poden ser exemples de recursos alternatius.


1. Crea en `res/` un directorio nuevo cuyo nombre tenga el formato *`<resources_name>-<config_qualifier>`*.
   - `<resources_name>` es el nombre del directorio de los recursos predeterminados correspondientes (definidos en la tabla 1).
   - `<qualifier>` es un nombre que especifica una configuración individual para la cual se deben usar estos recursos (que se definen en la tabla 2).

   Puedes agregar más de un `<qualifier>`. Separa cada uno con un guion.

En el següent enllaç podreu tots els exemples de recursos alternatius:
https://developer.android.com/guide/topics/resources/providing-resources

Manifest de l'aplicació

L'arxiu Manifest descriu l'informació essencial de l'aplicació per les eïnes de creació de Android, el SO i Google Play. Tindrà el nom AndroidManifest.xml a l'arrel del projecte.

1. El nom del paquet de l'aplicació. Servirà posteriorment per identificar l'app a Google Play.
2. S'ha de declarar tots els components de l'aplicació i les seves característiques bàsiques.
3. Permisos per la interacció de la app amb altres components o apps del sistema. Per exemple, càmara del telèfon. Tots els permisos queden registrats al Manifest una vegada declarats, i es sobreescriuen automàticament.
4. Requisits de software i hardware: El Manifest és l'arxiu de lectura que permet establir els paràmetres de compatibilitat de la app amb el SO/terminal.

Manifest és probablement el document més important de la app. Antigament, es modificava manualment. Les noves actualitzacions d'Android permeten una actualització automàtica del document a mesura que es modifica o es fan canvis en el codi del projecte.

Pràctica guiada Kotlin i Android

Operadors

Operació	Nomenclatura Estandard	Objecte
Suma	+	x.plus(y)
Resta	-	x.minus(y)
Multiplicació	*	x.times(y)
Divisió	/	x.div(y)

Nota: A la divisió el tipus del resultat de la divisió serà el mateix que el dividend.

Variables

Tot i que podem especificar el tipus d'una variable, el compilador detecta quin és. Un cop el tipus d'una variable és fixat, no es pot canviar.

	No es pot canviar el valor (Immutable)	Es pot canviar el valor (Mutables)

Numèric	`val number = 1`
	`number = 2`
	error: val cannot be reassigned
		`var number = 2`
		`number = 50`

Cadena de caràcters	`val blackColor = "black"`
	`blackColor = "white"`
	error: val cannot be reassigned
		`var blackColor = "black"`
		`blackColor = "white"`

Llistes	`val myList = listOf(0,1,2,3,4)`
	
	`val myList = listOf<Int>(0,1,2,3,4)`
		`var myList = mutableListOf(0,1,2,3,4)`
		
		`myList.add(5)`
		`myList.remove(0)`
		`myList.contains(5)`
		`myList.sum()`

Null (Nullability)

Kotlin ajudar a evitar null pointer exceptions.

```
var number : Int = null
```

error: null can not be a value of a non-null type Int

S'ha d'indicar que una variable pot ser nula amb el següent indicador:

```
var number : Int? = null
```

Com dir al compilador que una variable pot ser null o contenir nulls:

La llista pot tenir elements null:

```
var llista : List<String?> = listOf(null, null)
```

La llista pot ser null:

```
var llista : List<String>? = null
```

La llista i/o els elements de la llista poden ser null:

```
var llista : List<String?>? = null
```

Bucles

X .. Y és un range de X a Y

For	`for (c in 1..5) print(c)`
	`for (c in 5 downTo 1) print(c)`
	`for (c in 3..6 setp 2) print(c)`
	
	`for (c in 'b'..'g') print(c)`
	
	`for ((index, element) in myList.withIndex()){`
	`println("Index: $index; Element: $element")`
	`}`

While	`while (cond){`
	`//code`
	`}`

Repeat	`repeat(times){`
	`//code`
	`}`

Statement

If/else

```
if (condition) {
     println("True Condition")
} else {
     println("False Condition")
}

if (number in 1..100) println(number)

val isHot = if (number > 90) true else false
```

When

```
when (numberOfFish) {
     0 -> println("Empty    tank")
     in 1..50 -> println("Got fish!")
     else -> println("Perfect!")
}
```

L'expressió **when** és com **case** o **switch** en altres llenguatges.

Funcions

Programa Kotlin	Funció Kotlin

```
fun printHello () {
     println ("Hello Kotlin")
}

fun foo(bar: Any, baz: Int = 22):
Unit {
     //code
}

fun foo() : Boolean = return true
```

```
fun main (args: Array<String>) {
    println("Hello ${args[0]} ")
}
```

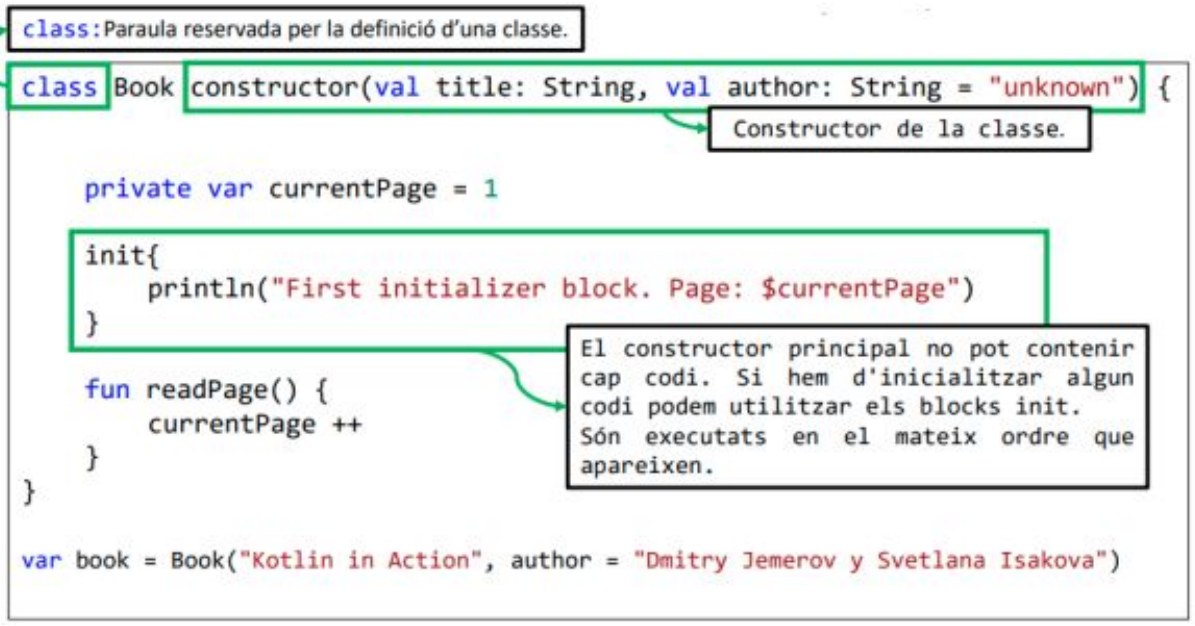
fun: Paraula reservada per la definició d'una funció.

Arguments de la funció. Cal definir primer el nom i després el tipus. Aquests arguments poden tenir valors per defecte.

Després de definir el nom de la funció i els arguments, podem especificar el tipus de dades que retornarà. En cas contrari el compilador l'inferirà.

Classes

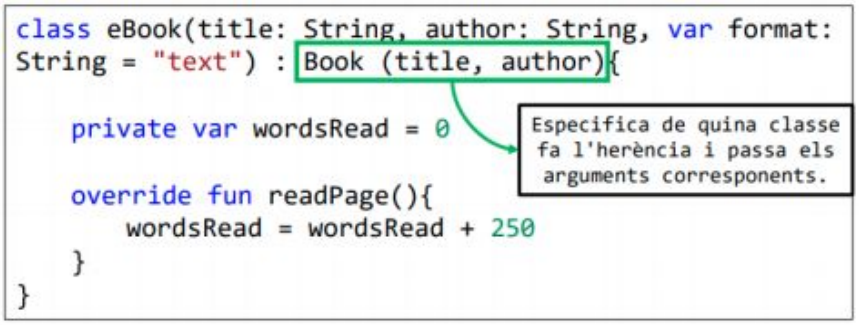
Les classes i els mètodes són **públics i finals** (no permeten fer herencia) per defecte.




```
class Book constructor(val title: String, val author: String = "unknown") {

    private var currentPage = 1

    init{
        println("First initializer block. Page: $currentPage")
    }

    fun readPage() {
        currentPage ++
    }
}

var book = Book("Kotlin in Action", author = "Dmitry Jemerov y Svetlana Isakova")
```

Herència

Per fer Herència d'una classe, cal afegir-li la paraula open abans de la definició i les funcions.




```
class eBook(title: String, author: String, var format:
String = "text") : Book (title, author){

    private var wordsRead = 0

    override fun readPage(){
        wordsRead = wordsRead + 250
    }
}
```

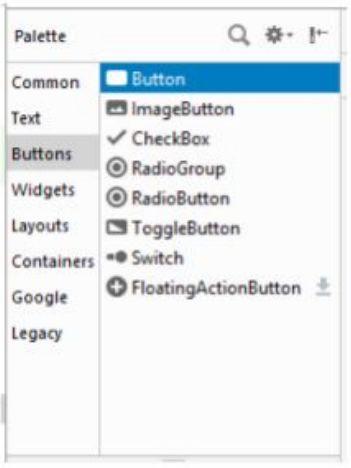
Canviar el missatge que apareix per pantalla

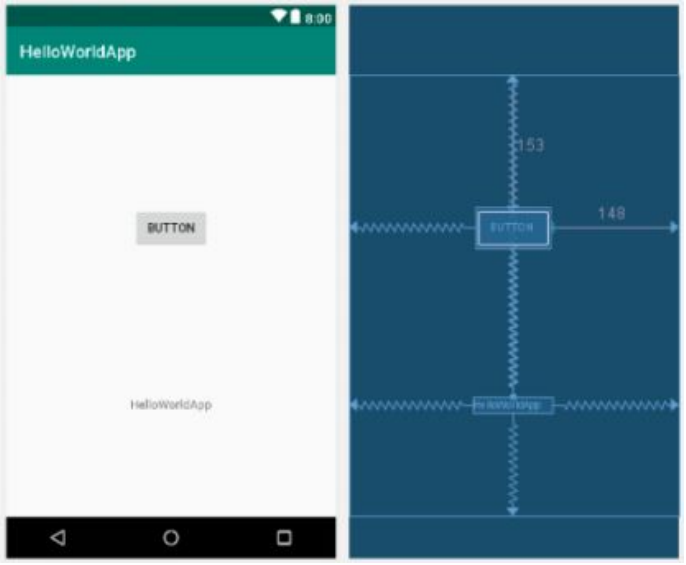
1. Definir el nou string a **res>values>strings.xml**: És important definir totes les cadenes de caràcters al fitxer string.xml, per una manipulació més ràpida i eficient. També ens permetrà poder traduir la nostra aplicació a altres idiomes.
2. Vincular el string creat al TextView del Layout.

```
<TextView
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
```

Afegir un botó a la vista

Afegir un botó a partir de la paleta de disseny dels Layouts. Simultàniament, podríem introduïr un botó definint-lo des del Layout.





Afegir un Toast al prémer el botó

Un **toast** és un missatge curt que apareix breument a la part inferior de la pantalla.

1. Afegir un nou mètode que rebrà un “click event”.

```
fun toastMe(view: View) {
    val myToast = Toast.makeText( context: this,  text: "Hello Toast!", Toast.LENGTH_SHORT)
    myToast.show()
}
```

2. A la pestanya Text del fitxer del Layout, afegir la propietat **android:onClick** al botó Toast.

```
<Button
        android:text="Button"
        android:onClick="toastMe"
        android:id="@+id/button"
```

Canviar el text al prémer el botó

1. Afegir una **id** al TextView en cas de no tenir-ne. Aquesta **id**ens permetrà identificar l’element des del codi.
2. Crear una nova funció que serà cridada al prémer el botó. Aquesta haurà de buscar el TextView i modificar-lo.
3. A la pestanya Text del fitxer del Layout, modificar la propietat **android:onClick** al botó per changeText.