1 - INTRODUCCIÓN ANDROID

SDK Android

- Software Development KIT, es, de forma análoga al JDK, el conjunto básico de librerías que forman el núcleo del entorno que permite desarrollar aplicaciones Android (compilador, debug, librerías básicas, etc…)

SDK Android

- Debemos descargar el API / versión con la que vayamos a trabajar

- El SDK Manager, integrado en el IDE, nos permite descargarnos todas APIS, herramientas y complementos

Herramientas de Desarrollo

- Android Studio, ha pasado en el segundo trimestre de 2015 a ser la herramienta oficial de desarrollo.
- El plug-in para Eclipse ADT ha dejado de mantenerse
- En Abril de 2016, se ha publicado la versión 2.1 de Android Studio

Instalación entorno

Instalamos el JDK 8

Android Studio (con SDK incluído)

Versiones

Desde su lanzamiento, hay una mejora constante en funcionalidades y vistosidad, que se traduce en nuevas clases, nuevas APIs y nuevas versiones

HISTORIAL

Versiones app/build.gadle

```
android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "edu.val.idel.rivas.turismorivas"
        minSdkVersion 15
        targetSdkVersion 28
        versionCode 3
        versionName "Mojito II" ...
```

Depuración

Para probar nuestros programas, es más cómodo hacerlo sobre el propio dispositivo, El emulador es lentísimo*. Debemos:

- Habilitar el modo desarrollador (7 veces sobre nº de compilación)
- Permitir instalar apps de origen desconocido
- Que A Studio reconozca nuestro dispositivo

Depuración

Y además:

- Build Variant DEBUG (Release APK)
- Dar desde nuestro móvil, dar permisos al equipo y habilitar la depuración USB

Enlaces de interés

Canal Youtube

https://www.youtube.com/channel/UC_x5XG1OV2P6uZZ5FSM9Ttw

https://www.youtube.com/user/androiddevelopers

Cuenta GitHub

https://github.com/googlesamples

Enlaces de interés

Documentación Oficial

http://developer.android.com/intl/es/index.html

Herramienta Generar Iconos

http://romannurik.github.io/AndroidAssetStudio/

Estructura de un Project

PROYECTO VS MODULO

- En teoría, puedo tener varias apps o módulos
- dentro de un mismo project. Android Studio
- además considera módulos a librerías
- externas, tests, etc
-
- En la práctica 1 Proyecto → 1 APP

Estructura de un Project

/app/src/main/java .- Código fuente

/app/src/main/res .- Imágenes, estilos, literales

- /drawable (recursos gráficos -project view)
- /drawable-ldpi (densidad baja)
- /drawable-mdpi (densidad media)
- /drawable-hdpi (densidad alta)
- /drawable-xhdpi (densidad muy alta)
- /drawable-xxhdpi (densidad muy muy alta)

Estructura de un Project

/app/src/main/res .- Imágenes, estilos, literales

- /layout (vertical / layaout-land)
  - interfaz usuario xml
- /anim/ - /animator/
- animaciones utilizadas por la aplicación.
- /menu/
  - los menús de la aplicación
- /mimap el icono de la aplicación

Estructura de un Project

/app/src/main/res .- Imágenes, estilos, literales

- /res/xml/
  - otros xml usados por la app
- /res/raw/
  - Recursos adicionales (no xml)
- /res/values/
- Literales texto (strings.xml),
- Estilos y temas (styles.xml)

Estructura de un Project

/app/src/main/res .- Imágenes, estilos, literales

- /res/values/
- Colores (colors.xml),
- Arrays de valores (arrays.xml),
- Tamaños (dimens.xml), etc.
-
- Las referencias a recursos en xml aparecen precedidas de una arroba @

Estructura de un Project

/app/libs .- Librerías

/app/build .- Los .class

/app/assets .- Porpiedades (Solo lectura)

/src/test .- Clases de TEST (Junit Mockito)

Estructura de un Project

/app/src/main/AndroidManifest.xml.- Fichero de Manifiesto o configuración que contiene toda la información relevante de la aplicación

<manifest package="com.val.ebtm.myiconnapp">

Estructura de un Project

Package .- Atributo importántisimo, ya que dará el nombre de nuestra app en la tienda (ID)

Contiene este fichero además una descripción textual de las clases relevantes para el motor de android: las Actividades, los Servicios y los Receivers (componentes)

MANIFEST

COMPONENETES

- <activity
- android:name="com.val.ebtm.view.AjustesActivity"
- android:label="Ajustes">
- </activity>

PERMISOS

- <uses-permission android:name="android.permission.INTERNET"/>
- <uses-permission android:name="android.permission.VIBRATE"/>
-

Estructura de un Project

ES IMPORTANTE AÑADIR LAS CLASES CON EL ASISTENTE, PUESTO QUE IMPLICAN NUEVOS ELEMENTOS XML EN EL FICHERO DE MANIFIESTO

Importar Biblioteca

- Puede resultar necesario o útil añadir clases
- de terceros. Para ello:

Seleccionamos en el menú de A Studio Build → Edit libraries and dependencies

O Editamos directamete el fichero app/build.gradle

Importar Biblioteca

- Para buscar librerías en JCENTER y MAVEN
- emplear las webs respectivas
-
- https://bintray.com/bintray/jcenter
- (busca en https://jcenter.bintray.com/)
- http://mvnrepository.com/

Importar Biblioteca

- Y buscar la pestaña de Gradle en la web
-
- compile group: 'com.google.code.gson',
- name: 'gson', version: '2.3.1'
- compile 'com.android.volley:volley:1.0.0'
-
-

Importar un GIT

- Es muy útil tomar ejemplos de aplicaciones
- que existen y jugar con ellos y depurarlos en
- nuestro entorno.
-
- Al estar github formado por repositorios
- públicos, puedo descargarlos directamente

Importar un GIT

- VCS → Checkout from version control → GIT
- Indicamos:
  - 1. URL repositorio remoto
  - 2. Directorio padre
  - 3. Subcarpeta proyecto
  - Y seguimos las instrucciones (posiblemente sea necesario
  - instalar librerías adicionales – el IDE nos avisa)

La clase R

Esta clase contiene referencia a todos los recursos de la aplicación definidos en /app/src/main/res/, de forma que podamos acceder fácilmente a estos recursos desde nuestro código java a través de dicho dato

La clase R

Se genera automáticamente y da acceso

- Layaout
- Strings
- Styles
- Iconos etc.

GRADLE

Lenguaje de script sobre el que se sustenta la compilación y gestión de nuestro app, introducido con A Studio

- Solo usuarios con necesidades
- avanzadas tendrán que editarlo
- manualmente

GRADLE

Usos:

- - Importación librerías
- - Generación de distintas versiones (“flavours”)
  - Para distintos dispositivos
  - Versión gratuita / de pago
  -

Callback

- O llamada del sistema ocurre cuando el SO u otra clase, ante un determinado evento, llama a un método mío
- Ocurre una “llamada por detrás”
- La llamada no la hago yo, pero la recibo y me toca gestionarla

Listener

- Concepto ligado al anterior. Una clase listener estará “escuchando” lo que pasa y ante un determinado evento, entrará en ejecución (por un callback)

Activity

- Al crear una actividad, se dibuja un menú con el método inflate (setContentView hace lo mismo)

  “INFLAR” es llevar la descripción de un xml a su equivalente clase Java.

Inflar

- LayoutInflater es el objeto encargado de esa función: obtener un XML sus equivalentes objetos JAVA

- Dibujar la vista inflada en la Actividad, puede ser o no inmediato, según los métodos de inflater que use

Inflar

1 Inflate (ID)

2 Inflate (ID, ViewGroup)

3 Inflate (ID, ViewGroup, boolean)

1 De xml a objeto, sin más

Inflar

1 Inflate (ID)

2 Inflate (ID, ViewGroup)

3 Inflate (ID, ViewGroup, boolean)

2 El ID inflado, pasa a ser un hijo de ViewGroup, heredando de éste sus atributos de tamaño horizontal y vertical

Inflar

1 Inflate (ID)

2 Inflate (ID, ViewGroup)

3 Inflate (ID, ViewGroup, boolean)

2 En este caso, el tercer parámetro se asume a true

Inflar

3 Inflate (ID, ViewGroup, boolean)

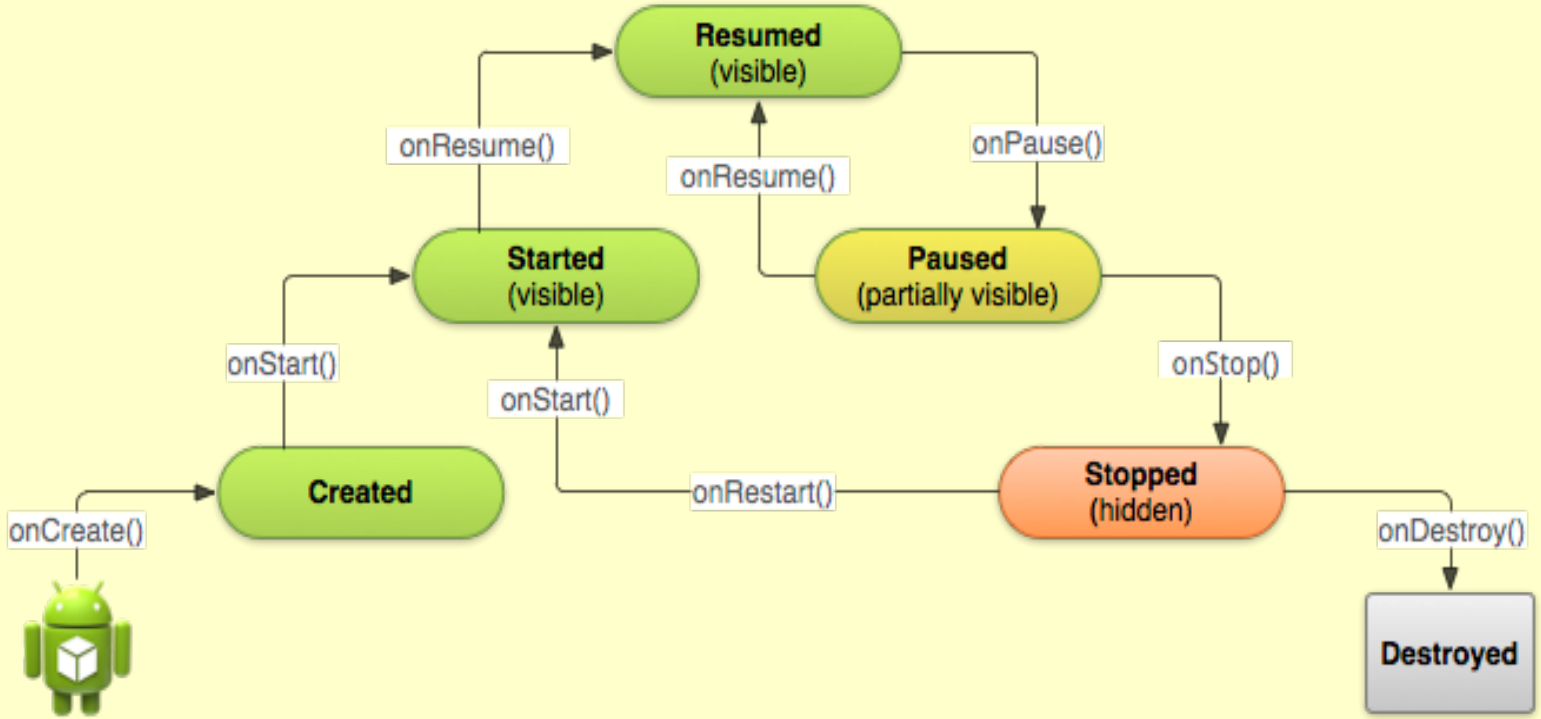
3 Se infla la vista (ID) con los atributos Layout del padre (ViewGroup) y si el parámtetro booleano indica si el recurso ID pasa a ser hijo o no en este momento (Sí por defecto)

Inflar

3 Inflate (ID, ViewGroup, boolean)

3 Si boolean es false, la vista generada, la puedo añadir posteriormente al ViewGroup con el método addView de ViewGroup.

Ciclo de Vida Activity



Ciclo de Vida

onCreate Inicializa variables y la UI

onStart La actividad se hace visible

onResume Primer plano (foreground)

onPause Parcialmente visible /apilada

onStopped Actividad No visible

onDestroy Actividad finalizada

Parada y reinicio

La parada pueden darse en diferentes escenarios. Como venir de:

- Aplicaciones Recientes
- Nueva actividad / back button
- Llamada / Alarma

Parada y reinicio

- OnPause es raro sobreescribir este método cuando la actividad es parcilamente visible. Podría usarse para liberar recursos temporalmente, como la cámara

Parada y reinicio

- OnStop, al contrario que OnDestroy, siempre es invocado. Luego es el sitio ideal para guardar datos que queremos conservar de manera persistente.

Fin inesperado (Recrear)

- El SO puede decidir acabar con una actividad o incluso la APP. Si ello ocurre, tenemos ocasión de guardar la información relavante e un objeto Bundle

  ¡De hecho, al cambiar la orientación, la activdad se destruye y se recrea!

Fin inesperado

- Los objetos visuales se autoguardan, pero si queremos guardar la **información de progreso** debemos usar

  @Override

  public void onSaveInstanceState

  (Bundle savedInstanceState)

Fin inesperado

Y RECUPERARLOS en onRestoreInstanceState (Bundle savedInstanceState)

Ó

onCreate (Bundle) //preguntar si !=null

Fin inesperado vs Parada

OnStop → Para guardar info No volátil

onSaveInstance → Para info volátil

Bundle

Clase que es una especie de saco donde puedo guardar cosas con un nombre y recuperarlas, por ese nombre.

```
Bundle.putString("mensaje", "Hola");
String c = Bundle.getString("mensaje");
```

Cambios orientación

Podemos adoptar 2 estrategias:

- INSENSIBLE
- SENSIBLE

Cambios orientación

Estrategia INSENSIBLE más fácil y más restrictiva. ¿Adecuada? Depende.

<activity
android:name="com.val.Actividad"
android:configChanges="keyboardHidden|orientation|screenSize"

Cambios orientación

Estrategia SENSIBLE. Más laboriosa. Defino un estilo para landescape o apasiado

Simplemente **creo un directorio layout-land** en /res donde pongo todos los xml adecuados a esta vista

Cambios orientación

Puedo además crear una subcarpeta para cada tipo de resolución (tipo media-query)

Por ejemplo /layout-sw720dp/ valdría para pantallas con esa resolución (10") o mayor

Depuración LOG

Para mostrar nuestros mensajes, sólo debemos emplear la clase del SDK android.util.Log, la cual nos ofrece los métodos estáticos para cada nivel de log, al estilo LOG4J

Log.e (getLocalClassName(), “error”);

Log.i (getLocalClassName(), “info”);

Log.d (getLocalClassName(), “debug”);

Log.v (getLocalClassName(), “verbose”);