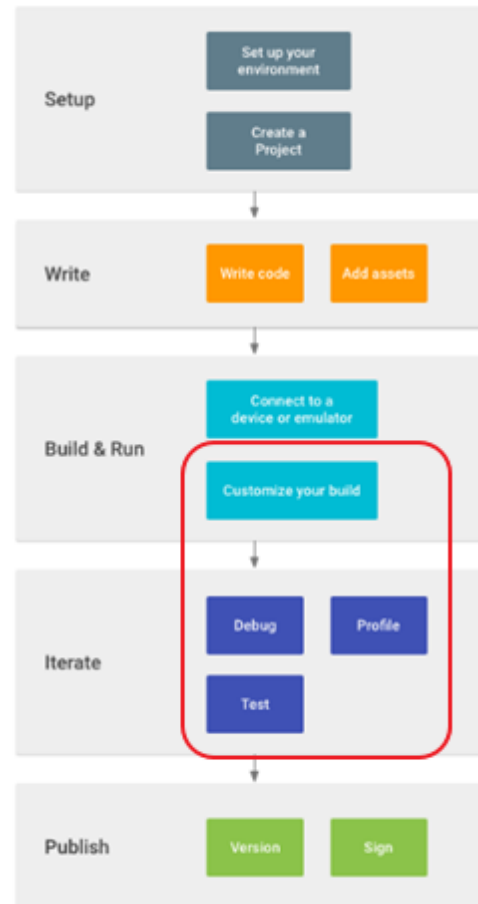
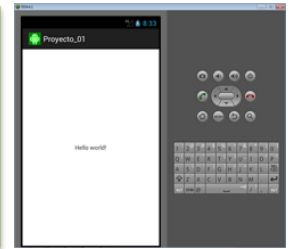
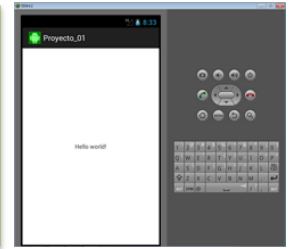


# Construcción, depuración y *testing*

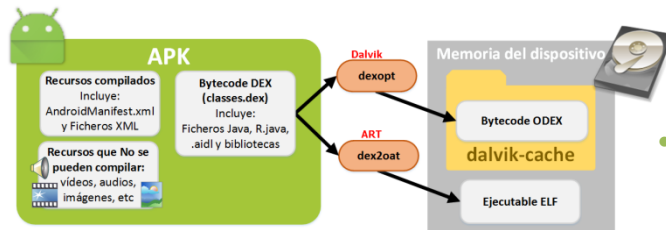
# Fase del flujo de trabajo



# Instalación y ejecución de la aplicación

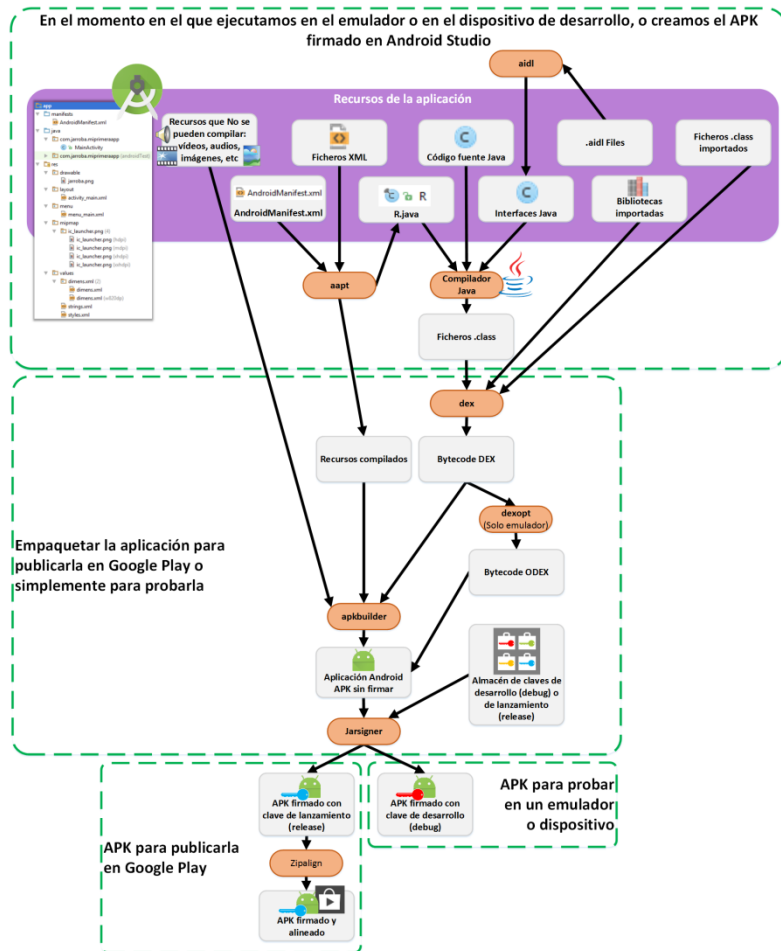
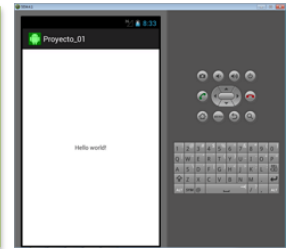


- Sabemos que todas las aplicaciones están escritas en Java/Kotlin y comprimidas en formato **APK** para que se puedan instalar sin dificultad.
- Dependiendo de la MV con que cuente el dispositivo, el proceso de instalación obtiene distintos resultados:



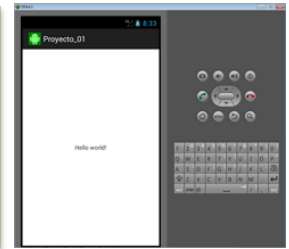
- **ODEX** (Optimized DEX, fichero DEX Optimizado): Este fichero contiene el conjunto de instrucciones utilizado por la máquina virtual **Dalvik**. Cada vez que la aplicación se ejecuta en el dispositivo la máquina virtual traduce las partes utilizadas del fichero DEX dinámicamente y lo guarda en el fichero ODEX; a medida que se va traduciendo más, se va guardando en el fichero ODEX.
- **ELF** (Executable and Linkable Format, Formato Ejecutable y Enlazable): El fichero ELF es un ejecutable para Linux; es un formato de archivo para ejecutables. El fichero bytecode DEX se traduce a código máquina del hardware donde se va a ejecutar; de ahí que se compile en el mismo dispositivo en el momento de la instalación. Es necesario para **ART**. (MV desde Android 4.4)
- Diferencias: ELF es la traducción completa de todo el fichero DEX en una sola vez durante la instalación; ODEX es la suma de las partes que ha ido traduciendo la máquina virtual Dalvik cada vez que se ejecuta la aplicación.

# Construcción de la APK

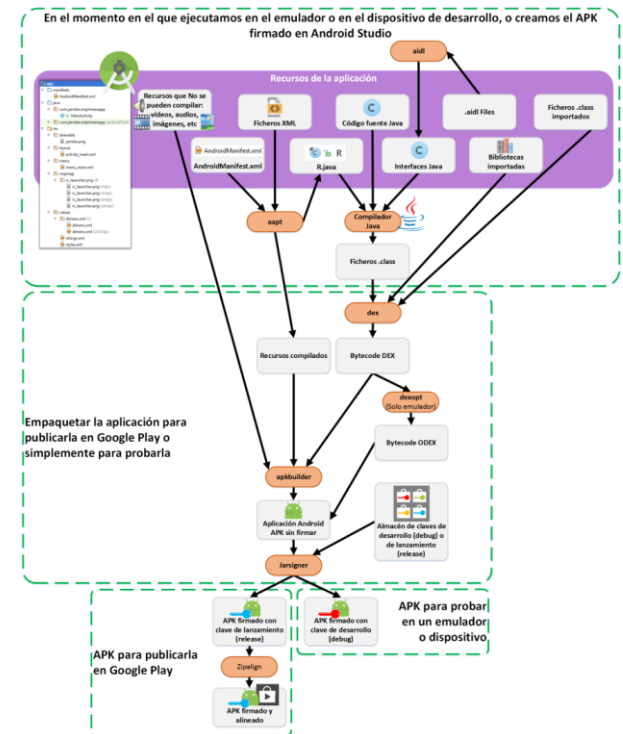


- **Android Studio utiliza Gradle para la obtención de la APK.**
- El Sistema de Construcción (Build System) combina el **JDK** de Java y las herramientas del **SDK** de Android.
- Con **Gradle** en Android Studio se compilará, firmará y optimizará la aplicación a la vez.
- Cuando se termine de preparar, se obtendrá una aplicación **APK** firmada que contendrá: código fuente compilado, recursos, el fichero de manifiesto, etc.
- Muchas veces nos parece un proceso muy lento, pero es que las tareas son muchas.

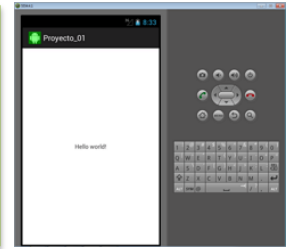
# Herramientas



- La mayoría de estas herramientas las podemos encontrar como ejecutables dentro de la carpeta del SDK de Android. Gradle hace que todo ese proceso sea automático y no tiene que preocuparnos más que por su lentitud ☺:
- aapt** (Herramienta de empaquetado de activos Android, The Android Asset Packaging Tool): Compila el fichero "AndroidManifest.xml" y los ficheros XML de las Activities. Además, genera el fichero "R.java" para referenciar los recursos desde el código Java.
- aidl** (Lenguaje para definir interfaces Android, Android Interface Definition Language): Convierte los interfaces ".aidl" (interfaces para comunicar servicios mediante la comunicación entre procesos, IPC).
- Compilador Java**: toma todo el código Java, incluidos los ficheros "R.java", los interfaces de ".aidl", y los compila en ficheros ".class".
- dex**: Convierte los ficheros ".class" en ficheros bytecode DEX. Además, cualquier biblioteca importada, así como otros ficheros ".class" que hayas incluido serán convertidos a ficheros DEX, incluyendo todas las clases de nuestro código en un único fichero DEX. Para optimizar el espacio, los Strings duplicados y otras constantes repetidas se introducen una única vez en el fichero DEX.
- dexopt**: Solo se aplica si se ejecuta en el emulador desde Android Studio. Convierte el fichero DEX al fichero compilado ODEX mediante una compilación AOT (Ahead Of Time, Antes de Tiempo) para que funcione única y exclusivamente en el emulador. En este caso el ODEX no se empaqueta dentro del APK, sino que se pone al junto al APK.
- apkbuilder**: imágenes, vídeos, y otros que no se pueden compilar, junto con los ficheros DEX son empaquetados en el fichero APK.
- Jarsigner**: Firmamos el ficheros APK con una clave de desarrollo (debug) o de lanzamiento (release). El fichero APK resultante ya se puede instalar en cualquier dispositivo Android.
- Zipalign**: Si el fichero APK se ha firmado con la clave de lanzamiento (release), el fichero se debe alinear (la alineación de la estructura de datos consiste en desplazar los bits un múltiplo del tamaño de la palabra que pueda controlar el procesador, y rellenar lo que falte; en Android normalmente son 4 bytes). El fichero APK alineado resultante aumenta el rendimiento del sistema de memoria debido a la forma en que el procesador gestiona la memoria en un dispositivo.



# Gradle



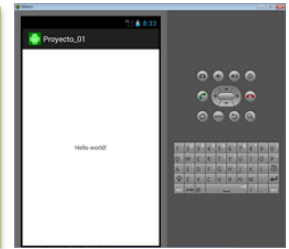
- Es una herramienta para automatizar la construcción de proyectos, por ejemplo las tareas de compilación, testing, empaquetado y el despliegue de los mismos (o lo que comúnmente se llama, *hacer una **build***).
- Usa un “Domain Specific Language” (DSL) basado en “Groovy” para declarar la formas de construir el proyecto (las tareas).
- Maneja compilaciones incrementales (básicamente verifica si hubo algún cambio en el código fuente después de la última compilación, si es así re-compila todo, si no se ahorra la tarea).
- Es Open Source licenciado bajo Apache Software License (ASL)

## #WHYGRADLE

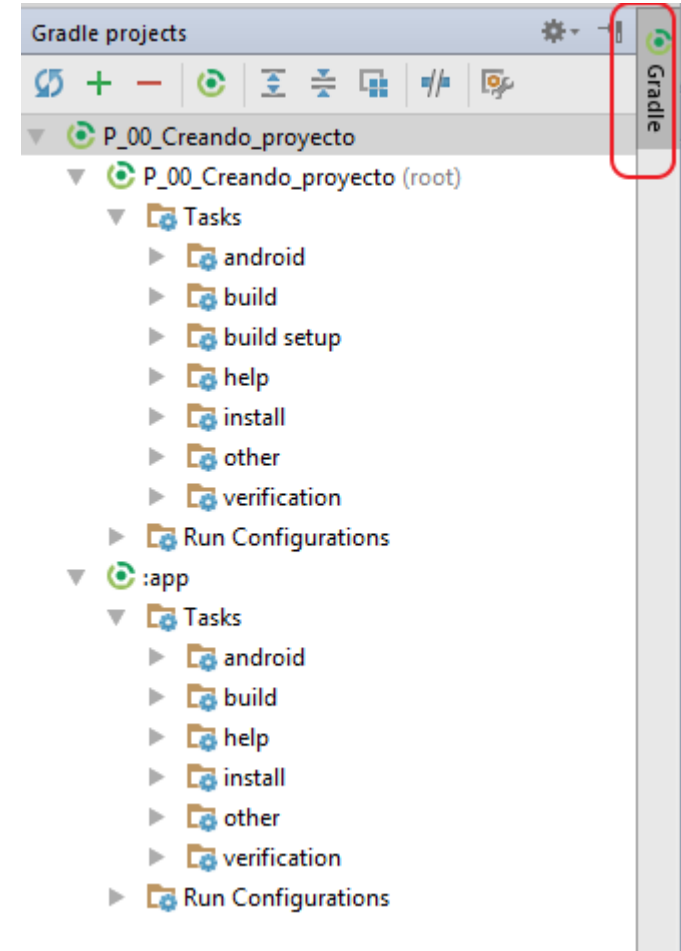
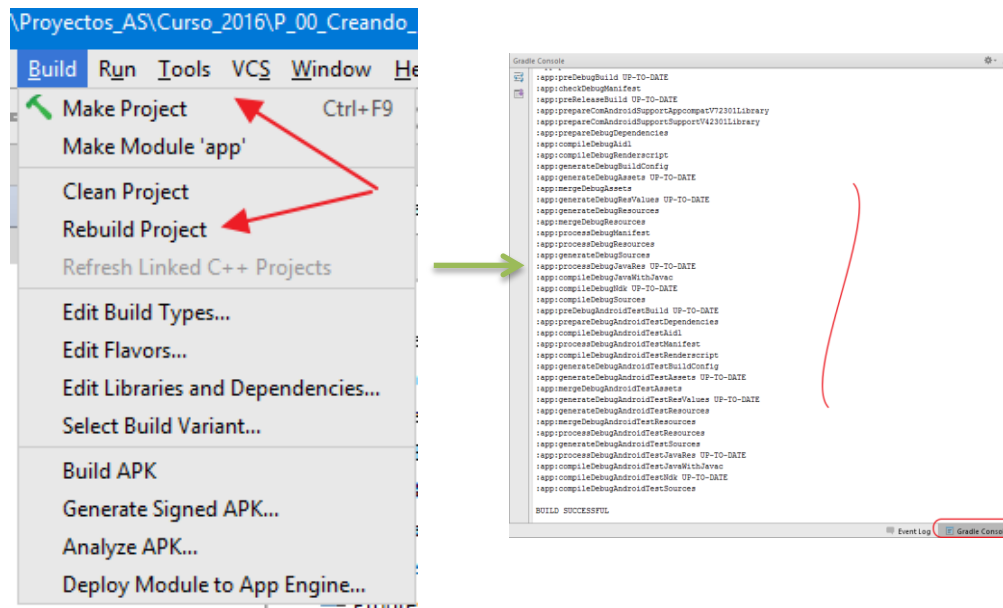
Gradle hace posible lo imposible, la posible fácil y la facilidad elegante.



# Gradle en Android

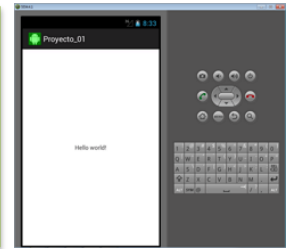


- Gradle es el sistema de compilación y construcción que ha adoptado Google para Android Studio.



- En nuestro proyecto hay dos ficheros build.gradle, uno a nivel de app (módulo) y otro a nivel de proyecto

# build.gradle (de app)

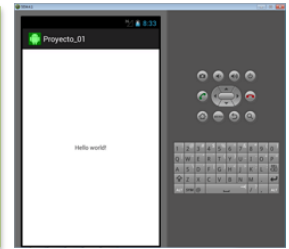


- Gradle no es un sistema específico de Android, sino que puede utilizarse con otros lenguajes/plataformas. Por tanto, lo primero que indicamos en este fichero es que utilizaremos el plugin para Android mediante la sentencia “**apply plugin**”.
- A continuación definiremos varias opciones específicas de Android, como las versiones de la API mínima (**minSdkVersion**), API objetivo (**targetSdkVersion**), y API de compilación (**compileSdkVersion**) que utilizaremos en el proyecto, la versión de las *build tools* (**buildToolsVersion**) que queremos utilizar, la versión tanto interna (**versionCode**) como visible (**versionName**) de la aplicación o el tipo de construcción para release o para debug (**buildTypes**).
- El ultimo elemento llamado “**dependencies**” también es importante y nos servirá entre otras cosas para definir las librerías externas que utilizaremos en la aplicación. Por defecto vemos que se añade la librería de compatibilidad appcompat que nos permite utilizar la *Action Bar* en la mayoría de versiones de Android, y todos los fichero .jar que incluyamos en la carpeta /libs.

```
1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 28
5      defaultConfig {
6          applicationId "com.pdm.p_00_creacion"
7          minSdkVersion 16
8          targetSdkVersion 28
9          versionCode 1
10         versionName "1.0"
11         testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
12     }
13     buildTypes {
14         release {
15             minifyEnabled false
16             proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
17         }
18     }
19 }
20
21 dependencies {
22     implementation fileTree(dir: 'libs', include: ['*.jar'])
23     implementation 'com.android.support:appcompat-v7:28.0.0-rc02'
24     implementation 'com.android.support.constraint:constraint-layout:1.1.3'
25     testImplementation 'junit:junit:4.12'
26     androidTestImplementation 'com.android.support.test:runner:1.0.2'
27     androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
28 }
```



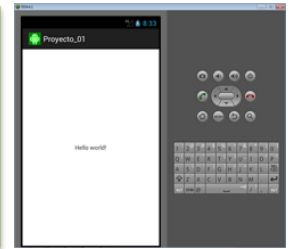
# build.gradle (de proyecto)



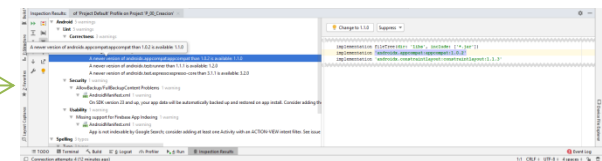
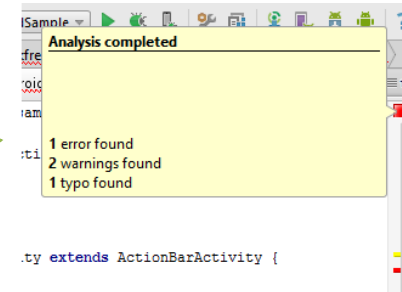
- Permite definir las características de construcción comunes a todos los módulos.
- El archivo Gradle a nivel de proyecto utiliza *buildscript* para definir dependencias de librerías remotas y los repositorios donde se alojan.
- Los repositorios soportados son JCenter, Maven Central o Ivy.
- En nuestro ejemplo se declara que se utiliza el repositorio JCenter y la dependencia a la ruta de clases que contiene el plugin de Android para Gradle versión 3.5.0.

```
activity_main.xml × app × P_00_Creacion × MainActivity.java ×
1 // Top-level build file where you can add configuration options common to all sub-projects/modules.
2
3 buildscript {
4
5     repositories {
6         google()
7         jcenter()
8     }
9     dependencies {
10         classpath 'com.android.tools.build:gradle:3.1.4'
11
12         // NOTE: Do not place your application dependencies here; they belong
13         // in the individual module build.gradle files
14     }
15 }
16
17 allprojects {
18     repositories {
19         google()
20         jcenter()
21     }
22 }
23
24 task clean(type: Delete) {
25     delete rootProject.buildDir
26 }
27
```

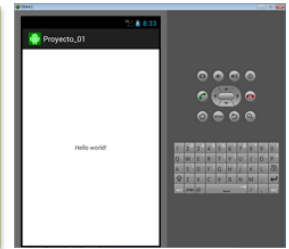
# Análisis




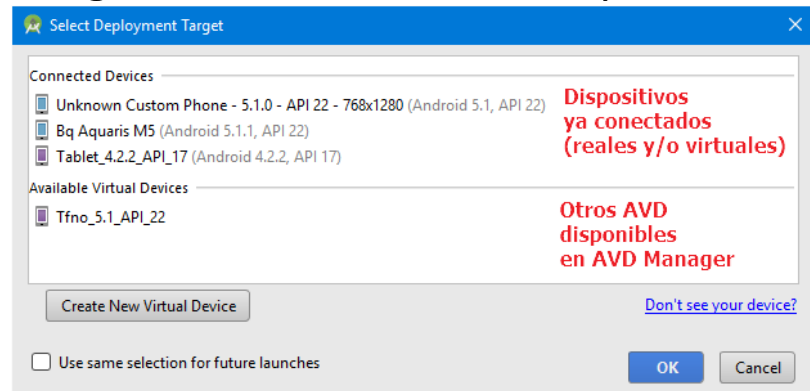
- A veces no es posible la compilación porque hay errores en el código.
- Android Studio incorpora una función denominada "análisis de código en la marcha".
- Además incluye **Android Lint**, un analizador estático que analizará el código fuente de nuestro proyecto. Esta herramienta puede detectar errores potenciales y otros problemas en el código que el compilador puede pasar por alto.
- La imagen siguiente, por ejemplo, nos dice que el **LinearLayout** del diseño no se usa.
- Es recomendable ejecutar la herramienta lint de Android Studio de vez en cuando para chequear los problemas potenciales de nuestro proyecto. La herramienta lint incluso nos dirá si tenemos imágenes o traducciones duplicadas.
- Para ejecutar la herramienta lint, seleccionamos **Inspect Code...** del menú **Analyze**. Cuando Android Studio finalice de inspeccionar nuestro proyecto nos mostrará el resultado en la parte baja de la ventana. Debemos tener en cuenta que además de **Android Lint**, Android Studio realiza uno cuantos chequeos adicionales también. Simplemente pulsando doble click sobre uno de los problemas el sistema nos llevará al fichero donde se encuentra localizado. Muchos errores pueden ser ignorados (la ortografía o el nombre del proyecto, etc.).



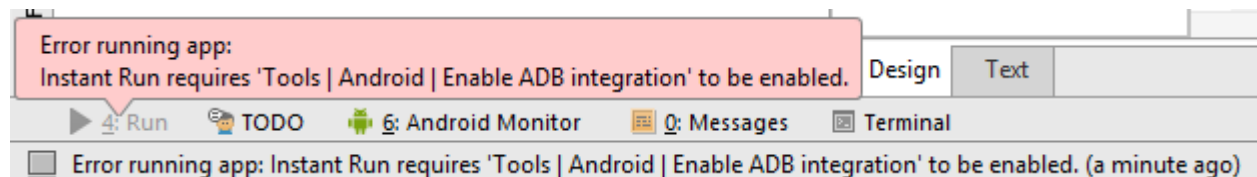
# Ejecutar aplicación



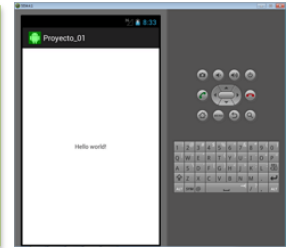
- Pulsaremos simplemente el menú Run / Run 'app' (o la tecla rápida Mayús+F10) o click en 
- Android Studio nos preguntará en qué dispositivo queremos ejecutar la aplicación de entre todos los que tengamos conectados o disponibles en el ADB, ya sean reales o virtuales.



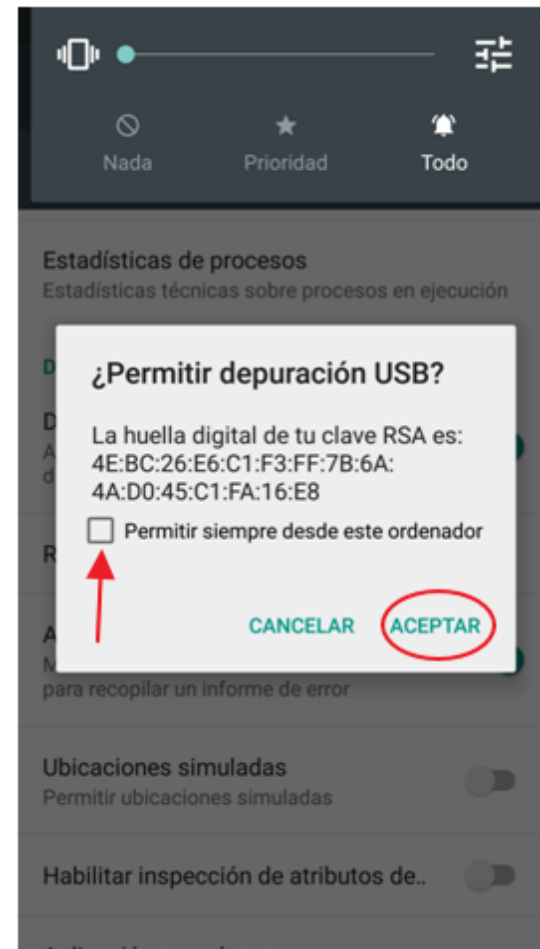
- Puede que a veces para que ejecutemos una aplicación debamos configurar AS:



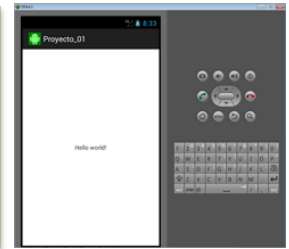
# Ejecución en dispositivo real



- Al intentar probar la aplicación en un dispositivo con Android 4.2.2 o superior desde un ordenador, se muestra un cuadro de diálogo en el que se solicita aceptar una clave RSA para realizar la depuración a través de este equipo.
- Este mecanismo de seguridad protege los dispositivos porque garantiza que la depuración USB y otros comandos adb no se ejecuten a menos que se desbloquee el dispositivo y se acepte el contenido del diálogo.
- Te aconsejo que habilites "Permitir siempre desde este ordenador" para todos tus equipos de desarrollo!



# App instalada

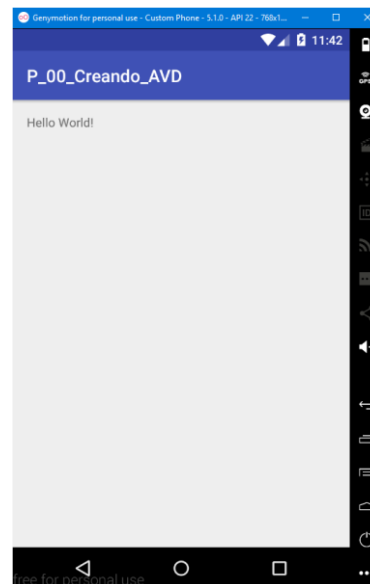


```
Run: AVD: Tfno_5.1_API_22 app
09/25 18:23:19: Launching app
$ adb push E:\Bea\TRABAJO\03_PDM\Proyectos_AS\Curso_2016\P_00_Creando_AVD\app\build\outputs\apk\app-debug.apk /data/local/tmp/com.pdm.p_00_creando_avd
$ adb shell pm install -r "/data/local/tmp/com.pdm.p_00_creando_avd"
    pkg: /data/local/tmp/com.pdm.p_00_creando_avd
Success

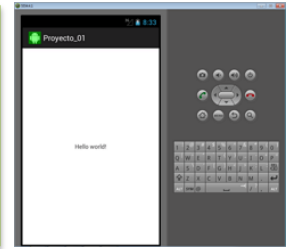
$ adb shell am start -n "com.pdm.p_00_creando_avd/com.pdm.p_00_creando_avd.MainActivity" -a android.intent.action.MAIN -c android.intent.category.LAUNCHER
Client not ready yet..Waiting for process to come online
Connected to process 5549 on device unknown-custom_phone__5_1_0__api_22__768x1280-192.168.56.101:5555
```


Instalación

Ejecución

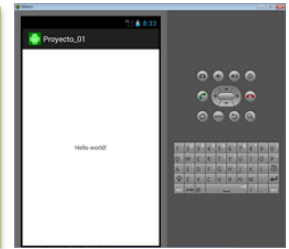


# Instant Run

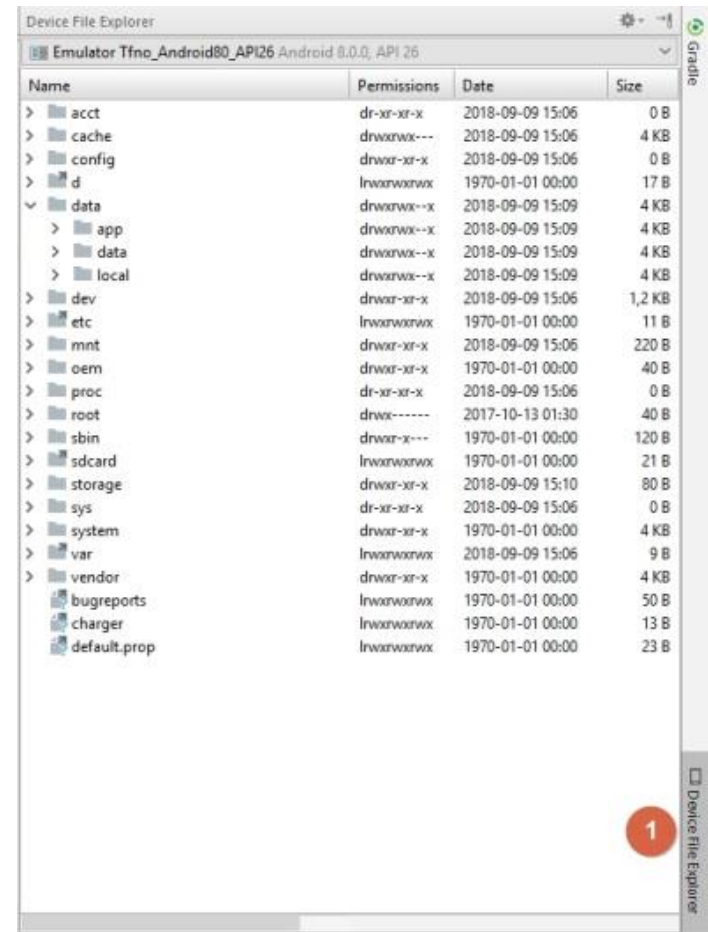


- Desde AS 2.0, con "Instant Run", pueden realizarse cambios en los métodos y recursos en una aplicación en ejecución sin necesidad de construir un nuevo APK, por lo que dichos cambios son visibles casi al instante.
- Sólo se admite para minSdkVersion 15 o más y el mejor rendimiento es para minSdkVersion 21 o superior.
- Después de ejecutar una aplicación, un icono con una flecha aparece detrás del botón **Run** , lo que indica que Instant Run está listo para transferir las actualizaciones la próxima vez que haga clic en el botón.

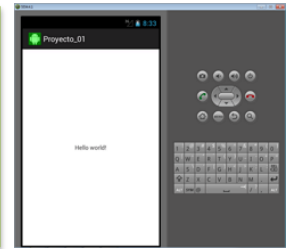
# Ver archivos del dispositivo



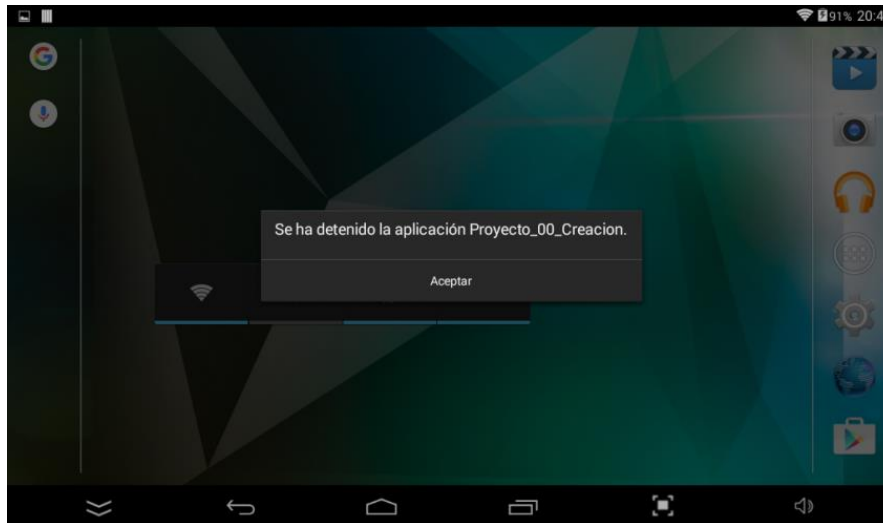
- El Explorador de archivos de dispositivos permite ver, copiar y borrar archivos en un dispositivo Android. Esto es útil para ver archivos creados por la app o si se desea transferir archivos hacia y desde un dispositivo.
- Para abrir el explorador de archivos de dispositivos basta con clicar en View > Tool Windows > Device File Explorer o en el botón Device File Explorer de la ventana de herramientas y seleccionar el dispositivo.
- **No todos los archivos de un dispositivo de hardware pueden visualizarse en el explorador de archivos de dispositivos. Por ejemplo, en el directorio data/data/, las entradas que corresponden a apps del dispositivo que no son depurables no se pueden expandir en el explorador de archivos de dispositivos.**
- Al explorar los archivos de un dispositivo, los siguientes directorios resultan particularmente útiles:
  - data/data/app\_name/ Contiene archivos de datos para tu app guardados en el almacenamiento interno
  - sdcard/ Contiene archivos del usuario guardados en el almacenamiento externo del usuario (fotos, etc.)
- Se puede interactuar con el contenido del dispositivo en la ventana del explorador de archivos haciendo clic con el botón secundario en un archivo o directorio para crear un archivo o directorio nuevo, guardar el archivo o directorio seleccionado en tu equipo y realizar cargas, eliminaciones o sincronizaciones. Haciendo doble clic en un archivo se abre en AS (Nota: AS guarda los archivos que se abren de esta manera en un directorio temporal fuera del proyecto. Si se realizan modificaciones en un archivo que se abrió usando el explorador de archivos de dispositivos y se desean guardar los cambios en el dispositivo, hay que cargar manualmente la versión modificada del archivo en el dispositivo).

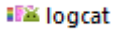



# Errores de ejecución



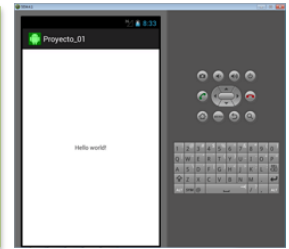
- El diseño lógico de la aplicación ha fallado!



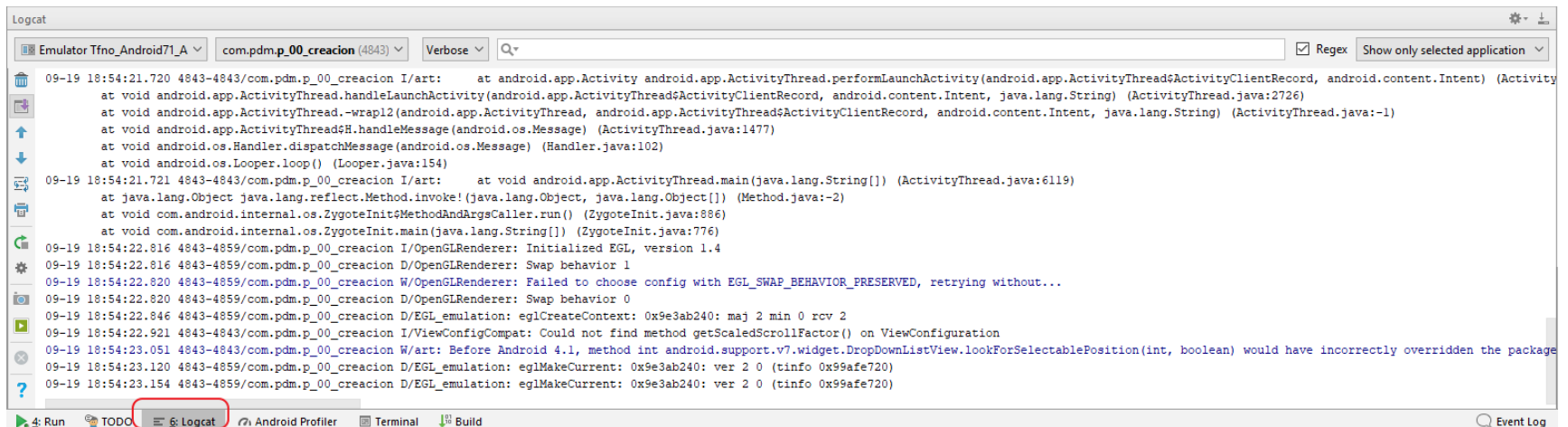
- Depurar:
  - Con  logcat
  - Con Android Profiler
  - Con depurador 



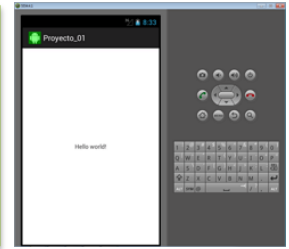
# LogCat



- La vista LogCat es la encargada de **mostrar el sistema de registros de Android.**
- **Tenemos todos los mensajes del dispositivo**, la mayoría de los cuales serán incomprensibles, pero también podremos ver mensajes de excepciones y mensajes de "señuelo" que hayamos dejado en el código de nuestra aplicación para ver hasta dónde ejecuta o por qué se rompe.



# Mensajes de LogCat

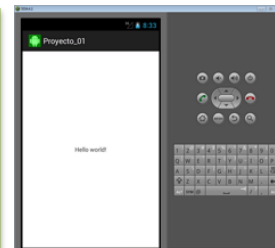


- El sistema Android utiliza el fichero *LogCat* para registrar todos los problemas y eventos principales que ocurren en el sistema. Ante cualquier error resulta muy interesante consultarlo para tratar de encontrar su origen.
- En Android, todos los mensajes de log llevarán asociada la siguiente información:
  - Fecha/Hora del mensaje.
  - Criticidad. Nivel de gravedad del mensaje.
  - PID. Código interno del proceso que ha introducido el mensaje.
  - Tag. Etiqueta identificativa del mensaje (se detalla más adelante).
  - Mensaje. El texto completo del mensaje.



```
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.pdm.practicas.proyecto_00_creacion/com.pdm.practicas.proyecto_00_creacion.MainActivity}: java.lang.ArithmeticException: divide by zero
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2195)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2245)
    at android.app.ActivityThread.access$800(ActivityThread.java:135)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5017) <1 internal calls>
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:779)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:595)
Caused by: java.lang.ArithmeticException: divide by zero
    at com.pdm.practicas.proyecto_00_creacion.MainActivity.onCreate(MainActivity.java:12)
```

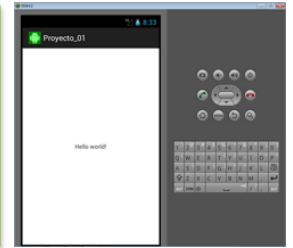
# Filtrar mensajes



- La ventana de LogCat ofrece una serie de funcionalidades para facilitar la visualización y búsqueda de determinados mensajes.
- Por ejemplo, podemos restringir la lista para que sólo muestre mensajes con una determinada criticidad mínima.
- O buscar los mensajes que se han producido en el "runtime" de la aplicación.
- Otro método de filtrado más interesante es la definición de filtros personalizados, donde podemos filtrar la lista para mostrar únicamente los mensajes con un PID o Tag determinado. Si hemos utilizado como etiqueta de los mensajes el nombre de nuestra aplicación o de nuestras actividades esto nos proporcionará una forma sencilla de visualizar sólo los mensajes generados por nuestra aplicación.



# Logging



- Una de las técnicas más útiles a la hora de depurar y/o realizar el seguimiento de aplicaciones sobre cualquier plataforma es la **creación** de logs de ejecución
- Android nos proporciona *logging* a través de la clase **android.util.Log**.
- Cada uno de estos métodos recibe como parámetros la etiqueta (tag) y el texto en sí del mensaje. Como etiqueta de los mensajes, aunque es un campo al que podemos pasar cualquier valor, suele utilizarse el nombre de la aplicación o de la actividad concreta que genera el mensaje.

## The Log class

`Log` is a logging class that you can utilize in your code to print out messages to the LogCat. Common logging methods include:

- `v(String, String)` (verbose)
- `d(String, String)` (debug)
- `i(String, String)` (information)
- `w(String, String)` (warning)
- `e(String, String)` (error)

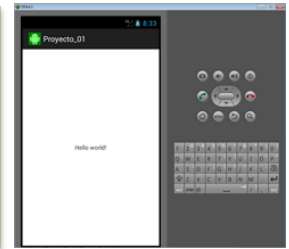
For example:

```
Log.i("MyActivity", "MyClass.getView() - get item number " + position);
```

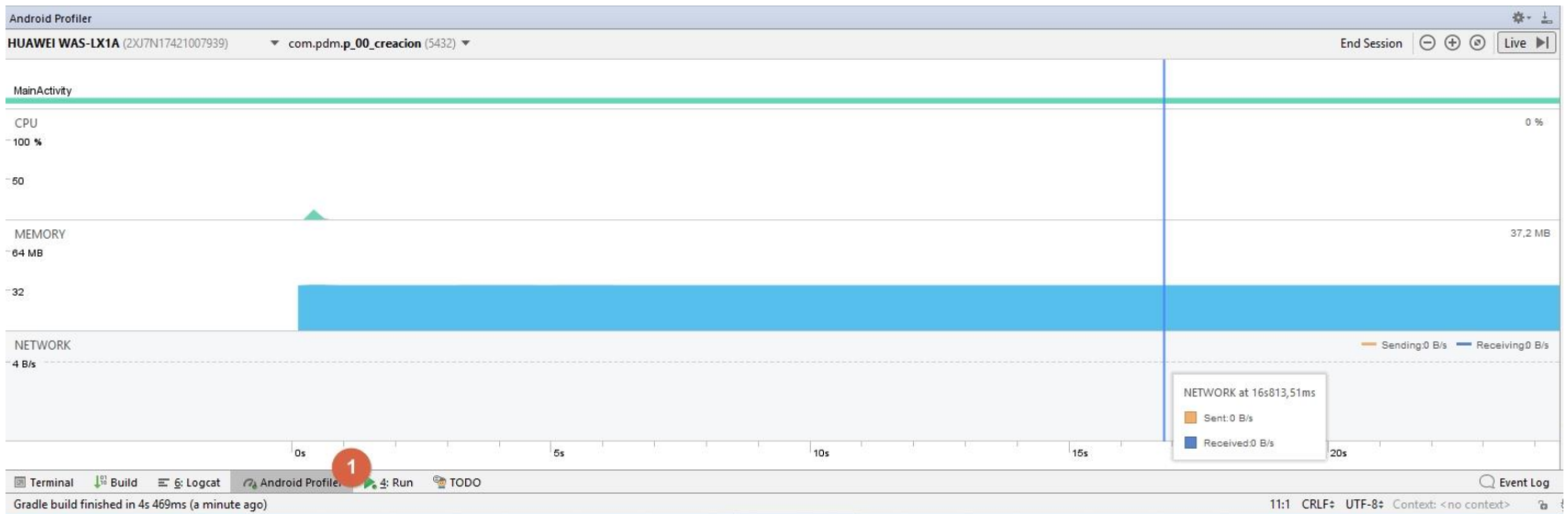
The LogCat will then output something like:

```
I/MyActivity( 1557): MyClass.getView() - get item number 1
```

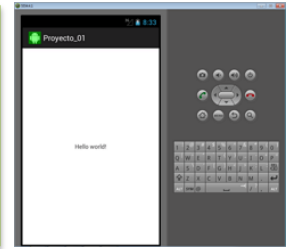
# Android Profiler




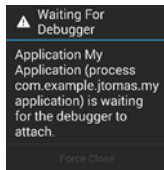
- Las herramientas de generación de perfiles proporcionan datos en tiempo real relacionados con la CPU, la memoria y la actividad de red de tu app.
- La mayoría de las veces solo tiene sentido en dispositivos reales.



# Ejecutar en modo depuración



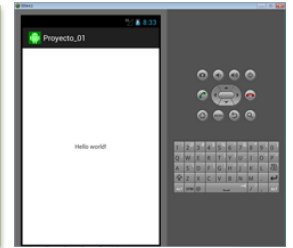
- Para averiguar más sobre el error, inserta un punto de ruptura en el código fuente en la línea deseada (el *breakpoint* se introduce haciendo clic en la barra de la izquierda).
- Selecciona con **Android Studio** *Run > Debug 'app'* (Mayús+F9) para ejecutarlo en modo *Debug* o pulsa 



- Tu aplicación se reiniciará en el dispositivo mostrando un mensaje de advertencia y quedará suspendida cuando alcance el punto de ruptura que has introducido.
- Entonces puedes recorrer el código en modo *Debug*, igual que se haría en cualquier otro entorno de programación. Pulsa en *Run > Step Over* (F8) para ir ejecutando las líneas una a una.

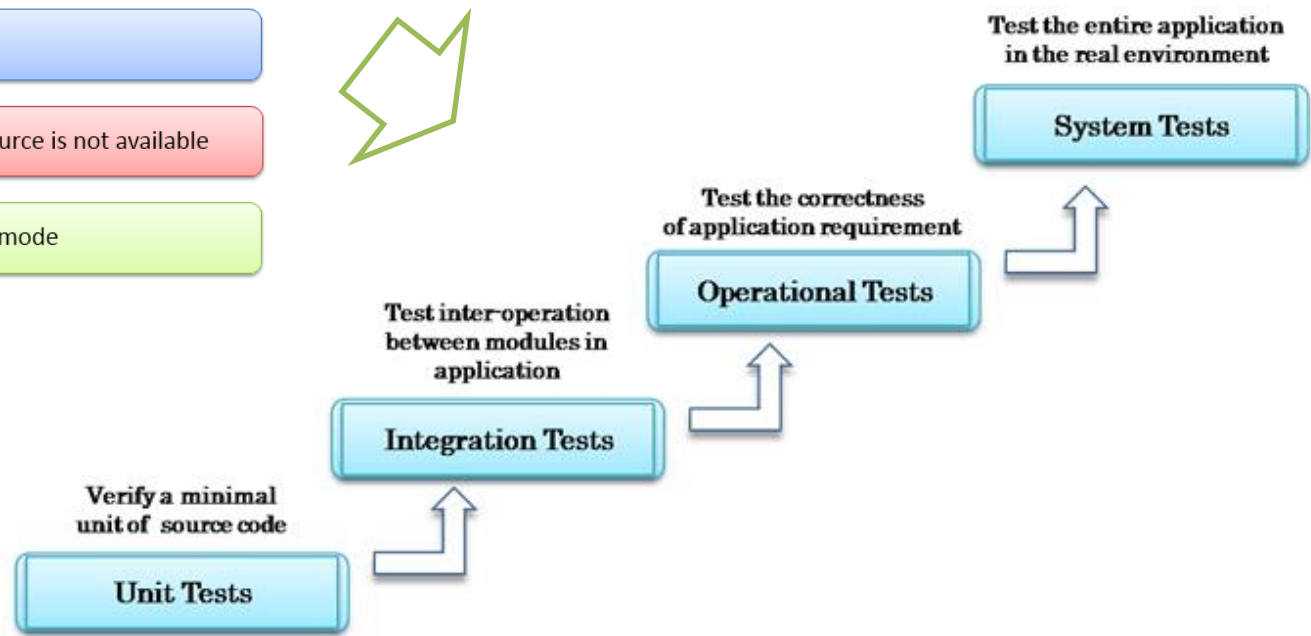


# Testing

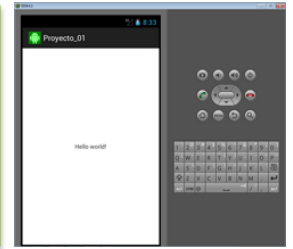


## Typical Failures in Android Application

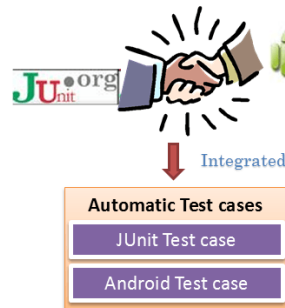
1. Application Installation Failure
2. Application crash during execution
3. Scaling/Layout Problems
4. Application hangs if some resource is not available
5. Problem in landscape/portrait mode



# Pruebas unitarias



- Las pruebas unitarias en Android están basadas en JUnit y Android Studio ya prepara los proyectos con la carpeta androidTest y su plantillas de casos de pruebas para cada clase java que creamos.



- [Información de Android Developers](#)
- [Otro buen manual de Testing con JUnit en Android](#)