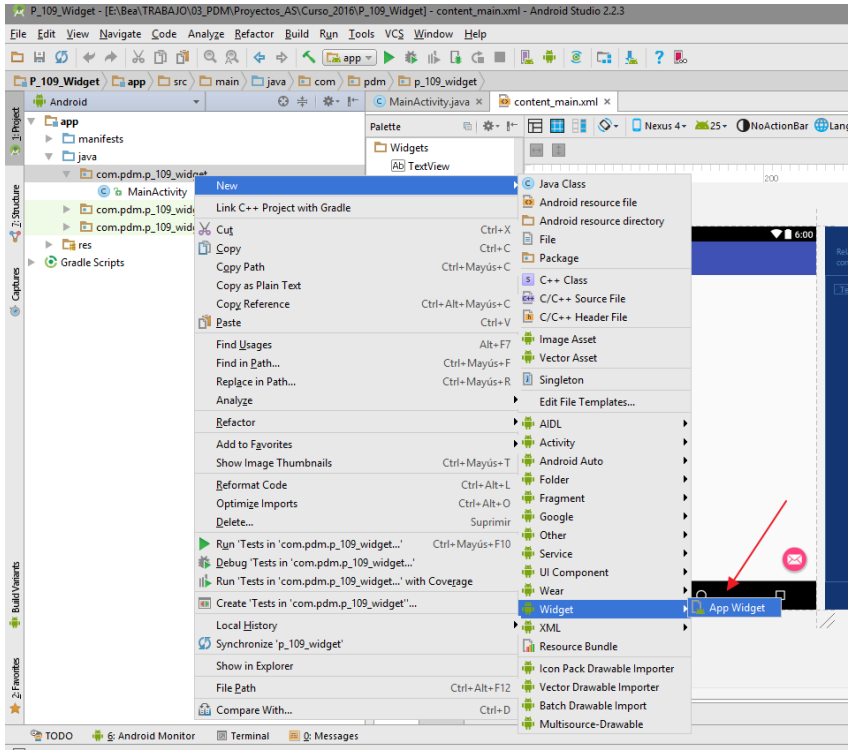
# Widgets

# App Widgets

- App Widgets son vistas en miniatura de las aplicaciones que se pueden incrustar en otras aplicaciones (como Home) y recibir actualizaciones periódicas.
- Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.
- Los App Widgets se suelen mostrar en la pantalla principal del dispositivo.
- Crea en AS el proyecto P_83_Widget_1 para practicar con este componente.

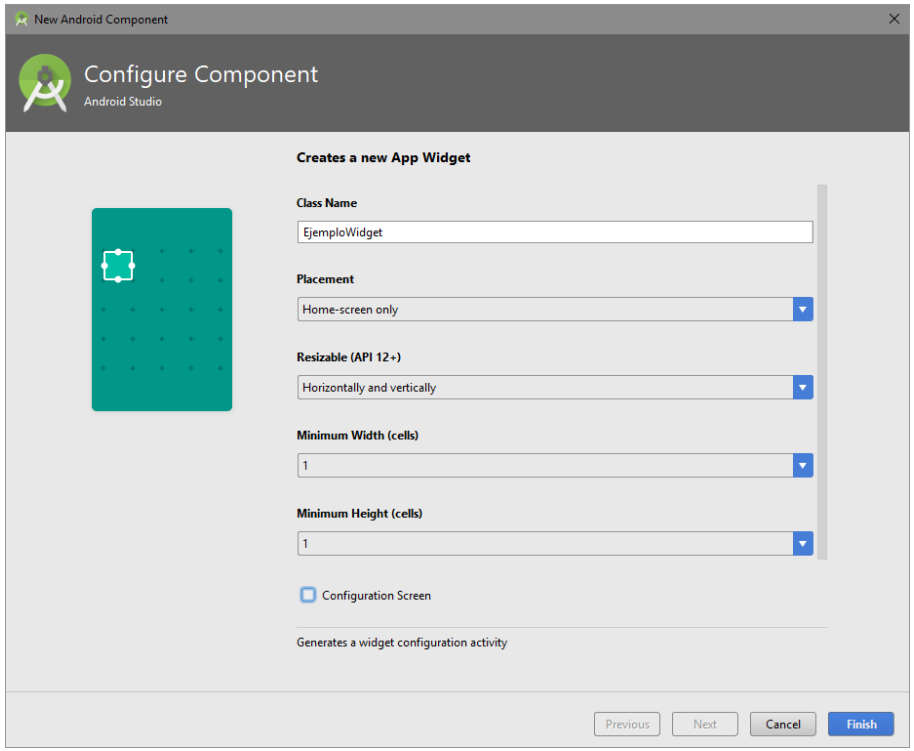
# Pasos de creación

- Declaración del widget dentro del *Android Manifest* de la aplicación
- Configuración del widget (AppWidgetProviderInfo) mediante fichero xml en /res/xml/
- Definir su *layout* mediante un archivo XML en /res/layout
- Implementación de la funcionalidad del widget (AppWidgetProvider) en la clase java correspondiente
- Implementación de la Actividad de configuración del widget (Opcional)
- AS nos proporciona un asistente que crea los pasos obligatorios anteriores e incluso el opcional



# Personalización del widget

- Nombre de la clase
- Si widget es para pantalla Home/pantalla Bloqueo/para ambas
- Si  puede cambiar de tamaño
- Tamaño mínimo en celdas
- Actividad de configuración (opcional)



# Trabajo realizado por el asistente

P_93_Widget_1 › app › src › main › AndroidManifest.xml

Android

- app
  - manifests
    - AndroidManifest.xml
  - java
    - com.pdm.p_93_widget_1
      - EjemploWidget
      - MainActivity
    - com.pdm.p_93_widget_1 (androidTest)
    - com.pdm.p_93_widget_1 (test)
  - res
    - drawable
      - example_appwidget_preview.png (nodpi)
      - ic_launcher_background.xml
      - ic_launcher_foreground.xml (v24)
    - layout
      - activity_main.xml
      - ejemplo_widget.xml
    - mipmap
    - values
    - xml
      - ejemplo_widget_info.xml
- Gradle Scripts

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.pdm.p_93_widget_1">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="P_93_Widget_1"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/AppTheme">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19
20         <receiver android:name=".EjemploWidget">
21             <intent-filter>
22                 <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
23             </intent-filter>
24
25             <meta-data
26                 android:name="android.appwidget.provider"
27                 android:resource="@xml/ejemplo_widget_info" />
28         </receiver>
29     </application>
30
31 </manifest>
```

# Declaración del widget en el Android Manifest

- El widget se declarará como un elemento <receiver> y deberemos aportar la siguiente información:

```
<receiver android:name=".EjemploWidget" >
        <intent-filter>
          <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>

        <meta-data
          android:name="android.appwidget.provider"
          android:resource="@xml/ejemplo_widget_info" />
</receiver>
```

Referencia a la clase java de nuestro widget

Indicaremos los "eventos" a los que responderá nuestro widget, normalmente al evento APPWIDGET_UPDATE, para detectar la acción de actualización.

Para enlazar el archivo de recursos xml "AppWidgetProviderInfo" con el widget que estamos creando

# Archivo de recursos AppWidgetProviderInfo

- Define las cualidades esenciales de un widget como las dimensiones mínimas del layout , el layout inicial, el periodo de actualización y opcionalmente una actividad de configuración que será lanzada al instalarse el widget.
- Se implementa mediante un archivo XML el cual debe estar en /res/xml/ y en el que se usa la etiqueta raíz <appwidget-provider>

ejemplo_widget_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialKeyguardLayout="@layout/ejemplo_widget"
    android:initialLayout="@layout/ejemplo_widget"
    android:minHeight="40dp"
    android:minWidth="40dp"
    android:previewImage="@drawable/example_appwidget_preview"
    android:resizeMode="horizontal|vertical"
    android:updatePeriodMillis="86400000"
    android:widgetCategory="home_screen"></appwidget-provider>
```

# Atributos

- No necesariamente hay que declarar todos los atributos, todo dependerá de nuestras necesidades.
  - minWidth y minHeight - indican la cantidad mínima de espacio que utilizara el widget.
  - minResizeWidth y minResizeHeight - indican el tamaño mínimo que deberá utilizar el widget.
  - **updatePeriodMillis** - define la frecuencia de actualización, el tiempo que debe de pasar para volver a llamar al método "onUpdate()". No se garantiza con exactitud el valor introducido y es recomendable no actualizar continuamente (como mucho una vez por hora). Además para ahorrar batería el sistema no actualizará con menos de 30 minutos. Si es necesaria una frecuencia menor, hay que programar alarma cíclica que lance servicio que actualice widget.
  - initialLayout - recurso "layout" con el diseño del widget.
  - configure - define una activity por defecto para configurar el widget la primera vez que el usuario lo arrastra al escritorio.
  - previewImage - define un archivo de recursos "drawable" con la imagen de la vista previa en la pestaña widgets del dispositivo. Si no se establece, el usuario verá por defecto el icono de la aplicación como vista previa.
  - autoAdvanceViewId - especifica la id del widget que debe auto-avanzar.
  - resizeMode - especifica como se debe ajustar su tamaño.
  - widgetCategory - indica si el widget sera de pantalla de inicio, de bloqueo o ambos.
  - initialKeyguardLayout - recurso "layout" con el diseño del widget para la pantalla de bloqueo.
- Documentación Android Developers

# Determinando tamaño del widget

- Debe definirse siempre minWidth and minHeight que son los tamaños mínimos que ocupará en la pantalla por defecto.
- La pantalla "home" de android está diagramada en forma de rejilla, la cantidad de filas y columnas depende del tamaño del display que posea el dispositivo que estemos usando. Esta rejilla puede variar según el dispositivo, por ejemplo, muchos teléfonos móviles ofrecen una cuadrícula de 4x4, y las tabletas pueden ofrecer una más grande, 8x7.
- Teniendo en cuenta las diferentes dimensiones de estas celdas según la orientación de la pantalla, existe una fórmula sencilla para ajustar las dimensiones de nuestro widget para que ocupe un número determinado de celdas sea cual sea la orientación:

  ancho_mínimo = (num_celdas_a_ocupar * 70) – 30

  alto_mínimo = (num_celdas_a_ocupar * 70) – 30
- Para una correcta creación del tamaño y dimensiones del widget podemos visitar el siguiente enlace: Instrucciones de diseño para widgets

# Layout inicial del widget

- La GUI de un Widget se definen mediante fichero en res/layout y deben estar basadas en un tipo especial de componentes llamados **RemoteViews** (comunica nuestra app con el widget)
- Un RemoteViews soporta las siguientes clases de ViewsGroup:
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - GridLayout (desde Android 4)
- y las siguientes clases de View:

AnalogClock
Button
Chronometer
ImageButton
ImageView
ProgressBar
TextView
ViewFlipper
ListView (desde Android3)
GridView (desde Android3)
StackView (desde Android3)
AdapterViewFlipper (desde And3)

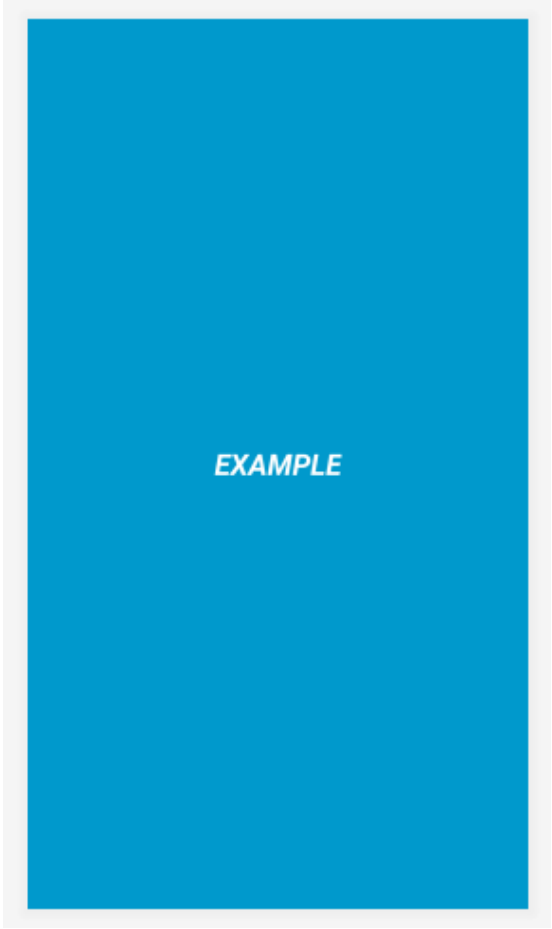
# Layout ejemplo_widget.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="#09C"
   android:padding="@dimen/widget_margin">

   <TextView
      android:id="@+id/appwidget_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:layout_margin="8dp"
      android:background="#09C"
      android:contentDescription="@string/appwidget_text"
      android:text="@string/appwidget_text"
      android:textColor="#ffffff"
      android:textSize="24sp"
      android:textStyle="bold|italic" />

</RelativeLayout>
```

Feo, eh?



# Implementación de la funcionalidad del widget (AppWidgetProvider)

- La clase EjemploWidget extiende la clase **AppWidgetProvider** (que a su vez no es más que una clase auxiliar derivada de BroadcastReceiver, ya que los widgets de Android no son más que un caso particular de este tipo de componentes, ya que responden al "paso del tiempo").
- Es la clase que maneja todos los eventos y configuración del widget.
- Recibe los eventos mas relevantes del widget como puede ser una actualización, eliminación, habilitación y deshabilitación y responde con los correspondientes métodos.
- En la mayoría de los casos, tendremos que implementar como mínimo el evento **onUpdate()**. El resto de métodos dependerán de la funcionalidad de nuestro widget.

EjemploWidget.java

```
package com.pdm.p_109_widget1;

import ...

/**
 * Implementation of App Widget functionality.
 */
public class EjemploWidget extends AppWidgetProvider {

    static void updateAppWidget(Context context, AppWidgetManager appWidgetManager,
                                int appWidgetId) {

        CharSequence widgetText = "EXAMPLE";
        // Construct the RemoteViews object
        RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.ejemplo_widget);
        views.setTextViewText(R.id.appwidget_text, widgetText);

        // Instruct the widget manager to update the widget
        appWidgetManager.updateAppWidget(appWidgetId, views);
    }

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        // There may be multiple widgets active, so update all of them
        for (int appWidgetId : appWidgetIds) {
            updateAppWidget(context, appWidgetManager, appWidgetId);
        }
    }

    @Override
    public void onEnabled(Context context) {
        // Enter relevant functionality for when the first widget is created
    }

    @Override
    public void onDisabled(Context context) {
        // Enter relevant functionality for when the last widget is disabled
    }
}
```

# Métodos de la clase AppWidgetProvider

- En esta clase deberemos sobreescribir los métodos a los que vamos a responder desde nuestro widget:
  - **onUpdate()** - llamado cada vez que el widget se actualice a intervalos definidos en el atributo "updatePeriodMillis". También es llamado cada vez que el usuario añada el widget al escritorio por lo que es el lugar indicado para configurar el widget y añadir un servicio si es necesario. Deberemos tener en cuenta que si creamos una activity de configuración, al añadir el widget al escritorio no se llamará al método, aunque si lo hará en las siguientes actualizaciones. Por lo tanto deberemos configurar el widget en la activity de configuración.
  - **onAppWidgetOptionsChanged()** - llamado la primera vez que se coloque el widget en el escritorio y/o cuando el widget cambie su tamaño.
  - **onDeleted()** - llamado cada vez que un widget es eliminado.
  - **onEnabled()** - llamado cuando el usuario añada la 1ª instancia del widget al escritorio. Si posteriormente añade mas instancias del mismo widget al escritorio no se llamará a este método.
  - **onDisabled()** - llamado cuando se elimine la ultima instancia de nuestro widget. Es un lugar indicado para limpiar los recursos generados en la clase del widget.
  - **onReceive()** - llamado para manejar cualquier estado anterior. Por lo tanto podremos anular todos los métodos anteriores y manejar desde aquí todos los eventos del widget. O responder a eventos ("clicks") sobre el widget.

# Usando RemoteViews

- Ya hemos comentado que en un widget el layout esta basado en un tipo especial de componente llamados RemoteViews
- Según la documentación oficial es una clase que describe una jerarquía de vistas que se pueden mostrar en otro proceso.
- La jerarquía es inflada desde un archivo layout, esta clase proporciona algunas operaciones básicas para modificar el contenido de dicha jerarquía.
- Básicamente permite acceder a una vista y realizar modificaciones.
- Observa el fichero java:

```
// Construct the RemoteViews object
RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.ejemplo_widget);
```

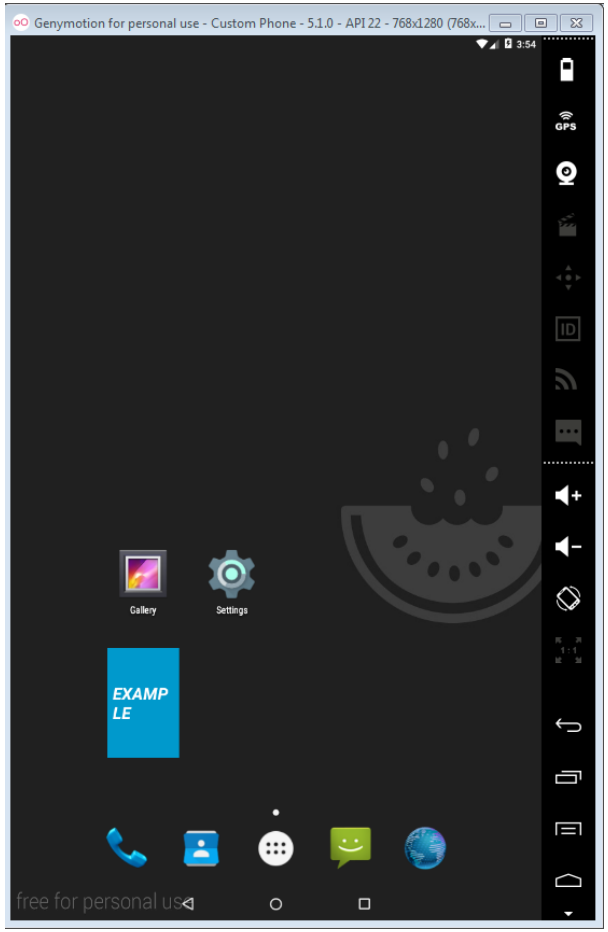
  Se 'infla' con el paquete context.getPackageName() y el layout a mostrar.

- También posee métodos para modificar contenidos de componentes como:

  views.setTextViewText(R.id.appwidget_text, widgetText);

  modifica el texto del TextView del layout del widget.

# Puesta en marcha




- Para probarlo, podemos ejecutar el proyecto en el emulador, esperar a que se ejecute la aplicación principal (en la que, si quieres, puedes cambiar el mensaje "Hello, word!" por "Widget instalado!"), ir a la pantalla principal del emulador y añadir nuestro widget al escritorio tal cómo lo haríamos en nuestro teléfono.
- El ejemplo es tan básico que cada 24 horas se actualiza el texto con el mismo texto!
- Las actualizaciones consumen batería! Android recomienda no menores a 1 hora e incluso que el usuario las pueda fijar. Si son necesarias más continuas hay que utilizar Alarmas.

# Más información

- La documentación de Android ofrece la siguiente:
  - Recomendaciones para el diseño (y posibilidad de conseguir templates)
  - Guía (Es interesante la recomendación que hacen en cuanto a los márgenes, ya que a partir de Android 4.0 los widgets incorporan un margen por defecto)

# Eventos en controles de RemoteViews

- A los controles utilizados en los widgets de Android, que ya sabemos que son del tipo RemoteView, **no podemos asociar eventos de la forma tradicional** (setOnClickListener …).
- Sin embargo, en su lugar, tenemos la posibilidad de **asociar a un evento** (por ejemplo, el click sobre un view) **un determinado mensaje** (**Pending Intent**) de tipo broadcast que será lanzado cada vez que se produzca dicho evento.
- Además, podremos configurar el widget para que capture esos mensajes, e implementar en el evento **onReceive()** del widget las acciones necesarias a ejecutar tras capturar el mensaje.
- Con estas tres acciones simularemos la captura de eventos sobre controles de un widget.

# Repaso de PendingIntent

- El propósito del objeto *PendingIntent* es conceder permiso a una aplicación externa para utilizar la intención contenida, como si se hubiera ejecutado desde el propio proceso de nuestra aplicación.
- Los principales casos en los que usamos un *PendingIntent* son:
  - Declarar un *Intent* para ser ejecutado cuando el usuario realice una acción en una notificación (NotificationManager del sistema Android ejecuta el *Intent*).
  - Declarar un *Intent* para ser ejecutado cuando el usuario realice una acción en un Widget (la aplicación de la pantalla de inicio ejecuta el *Intent*). **NUESTRO CASO.**
  - Declarar un *Intent* para ser ejecutado en un momento determinado en el futuro (AlarmManager del sistema Android ejecuta el *Intent*).
- Debido a que cada objeto *Intent* está diseñado para ser manejado por un tipo específico de componente (ya sea una Actividad, un Servicio o un *BroadCastReceiver*()), también debe ser creado un *PendingIntent* con la misma consideración. Cuando utilizamos un *PendingIntent*, nuestra aplicación no ejecutará el *Intent* con una llamada a *startActivity()*, debemos declarar el tipo de componente de destino al crear el *PendingIntent* con la llamada a su método creador respectivo:
  - *PendingIntent.getActivity()* para un *Intent* que inicia una *Actividad*.
  - *PendingIntent.getService()* para un *Intent* que inicia un *Servicio*.
  - *PendingIntent.getBroadcast()* para un *Intent* que inicia un *BroadcastReceiver*.
- En resumen, y para aclarar de una forma más sencilla lo que hace un *PendingIntent* podemos decir que es un *Intent* que se manda al sistema, pendiente de una ejecución futura, que desconocemos cuando se producirá.

# Clase AppWidgetManager

- Permite actualizar el estado de un widget, obtener información acerca de los AppWidget providers instalados y el estado de otros widgets.

  AppWidgetManager appWidgetManager;
- La clase AppWidgetManager tiene el método updateAppWidget() que necesita 2 parámetros: la referencia al widget dada por ComponentName y el RemoteView a mostrar

  appWidgetManager.updateAppWidget(thisWidget, remoteViews);
- La clase ComponentName identifica a un componente especifico de una aplicación disponible (Activity, Service, BroadcastReceiver, or ContentProvider). Son necesarios dos parámetros para identificar a un componente: el paquete y el nombre de la clase:

  ComponentName thisWidget = new ComponentName(context, nombre_clase.class);
- Veremos como utilizar la clase para actualizar el widget con click en los ejercicios.

# Prácticas propuestas

- Realiza la hoja de ejercicios Ejercicios_19_Widgets (para añadir funcionalidades a los widgets: configuración inicial, actualizaciones periódicas y respuesta a eventos)
- Ojo!: A veces, aunque corrijamos los errores en el proyecto, parece que en el emulador siguen sin arreglarse por mucho que desinstalemos el widget y/o reiniciemos el emulador. La mejor solución es borrar la caché y dejar el emulador como nuevo antes de volver a reiniciarlo (aunque habremos perdido las aplicaciones anteriormente instaladas!).
- AVD del SDK: Wipe data desde AVD Manager
- AVD GenyMotion: Ver enlace