Estructura De Un Proyecto En Android Studio

Como habíamos visto en el **artículo introductorio a la programación en Android**, el proyecto de una aplicación está contenido en una jerarquía donde se ubican todos los archivos de código fuente **Java**, los recursos, las configuraciones y los archivos de construcción de **Gradle**.

El artículo de hoy tiene como fin comprender la estructura de archivos de un proyecto en **Android Studio**. Esto nos permitirá investigar a fondo la lógica de construcción de una aplicación.

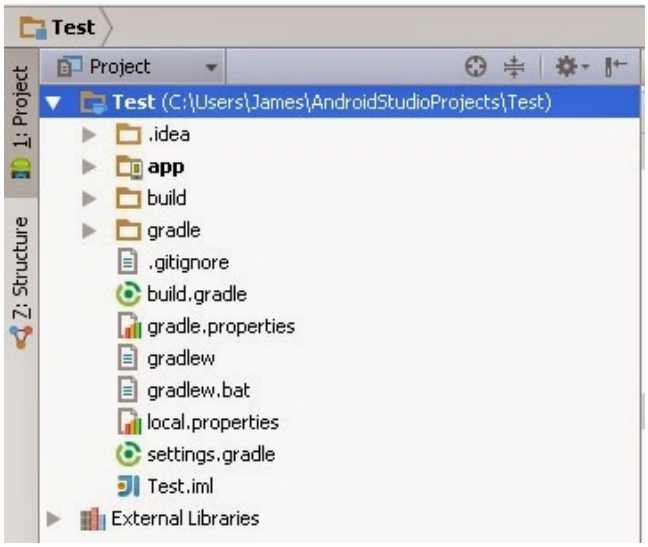
También estudiaremos como está estructurado el archivo AndroidManifest.xml, que contiene la carpeta «res», veremos la utilidad de los archivos de diseño, los archivos R.java. Y finalmente daremos un vistazo al **código fuente**.

CONTENIDO

Directorios de un Proyecto en Android Studio
¿Qué es el ANDROID MANIFEST?
¿Qué utilidad tiene el archivo strings.xml?
¿Qué hay dentro de la carpeta layout?
La carpeta drawable en Android Studio
Comprendiendo el propósito de la clase R.java
La carpeta java de un proyecto en Android Studio

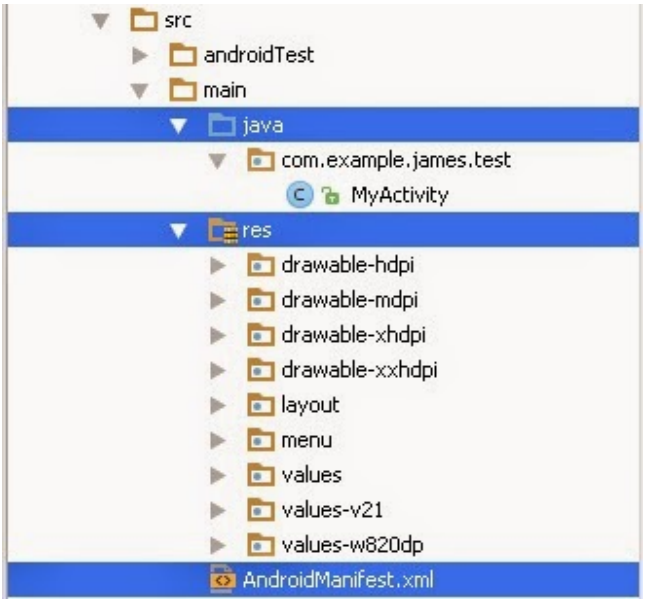
Directorios de un Proyecto en Android Studio

Para comenzar la explicación, ejecuta **Android Studio** y selecciona el proyecto «Test» que creamos anteriormente. Ahora ubícate en el panel de la parte izquierda llamado «Project», el cual representa la estructura de tu proyecto. Como notas, están las carpetas .idea, app, build, gradle y otros archivos de configuración.



La carpeta app es la que contiene todo lo relacionado con tu proyecto, es la carpeta que nos interesa por el momento y donde incluiremos los archivos necesarios para que nuestra aplicación sea empaquetada. Si despliegas su contenido veras tres carpetas: build, libs y src. Por ahora ignoraremos los demás archivos.

Normalmente la mayor parte del tiempo y fuerzas las usaremos en la carpeta src(Source). Dentro de ella se ubica la carpeta main, la cual contiene todos los **archivos fuente Java** para nuestra aplicación. La carpeta res (Resources) que contiene los recursos del proyecto (iconos, sonido, diseños, etc.) y el archivo AndroidManifest.xml.



¿Qué es el ANDROID MANIFEST?

Es un **archivo XML** que contiene nodos descriptivos sobre las características de una aplicación Android. Características como los **building blocks** existentes, la **versión de SDK** usada, los permisos necesarios para ejecutar algunos servicios y muchas más. En pocas palabras el Android Manifest es un panorama de toda nuestra aplicación.

Si abres el archivo en el editor verás un código similar a este:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/andr
    package="com.TUPAQUETE.test" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MyActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN

                <category android:name="android.intent.category.
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Todas las aplicaciones deben contener este archivo por convención. El nombre debe permanecer intacto, ya que se usa como referencia

para el **parsing** de nuestra aplicación. El **nodo raíz** de este documento se representa con la etiqueta <manifest> y por obligación debe contener un hijo de tipo <application>.

En el código anterior podemos observar que manifest posee dos atributos, xmlsn:android y package. El primero no debemos cambiarlo nunca, ya que es el **namespace** del archivo.

El atributo package indica el nombre del **paquete Java**que soporta a nuestra aplicación. El nombre del paquete debe ser único y un diferenciador a largo plazo.

La etiqueta <application> representa como estará construida nuestra aplicación. Dentro de ella definiremos nodos referentes a las **actividades** que contiene, las **librerías incluidas**, los **Intents**, **Providers**, y demás componentes.

Algunos atributos de la etiqueta <application> son:

- allowBackup: Este atributo puede tomar los valores de true o false. Indica si la aplicación será persistente al cerrar nuestro **AVD**.

- icon: Indica donde está ubicado el icono que se usará en la aplicación. Debemos indicar la ruta y el nombre del archivo que representa el icono. En este caso apuntamos a las carpetas drawable donde se encuentra ic_launcher.png. Icono por defecto que nos proporciona **Android Studio**.

- label: Es el nombre de la aplicación que verá el usuario en su teléfono. Normalmente apunta a la cadena «app_name» que se encuentra en el recurso strings.xml (Más adelante lo veremos).

- theme: Este atributo apunta al archivo de recursos styles.xml, donde se define la personalización del **estilo visual** de nuestra aplicación.

Dentro de <application> encontraremos expresada la actividad principal que definimos al crear el **proyecto Test**. Usaremos <activity> para representar un nodo tipo actividad.

<activity> tiene dos atributos: name, el cual se refiere a la **clase Java** que hace referencia a esta actividad (Comienza con un punto «.»). Y el atributo **label** que hace referencia al texto que se mostrará en la cabecera de la actividad. En este caso es el mismo string «app_name».

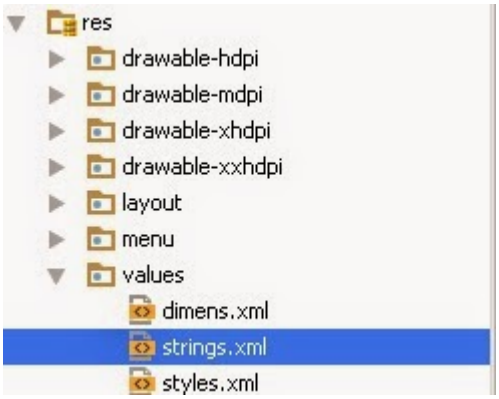
Por el momento estudiaremos hasta este punto. En próximos artículos veremos el proposito de los nodos <intent-filter>, <action> y <category>.

¿Qué utilidad tiene el archivo strings.xml?

Dentro de la carpeta»res» encontraremos todos aquellos recursos tercerizados para nuestra aplicación. Esta práctica de excluir los atributos de la aplicación a través de archivos externos, permite reducir la complejidad de diseño en las interfaces.

Uno de los recursos más relevantes es el archivo strings.xml que se encuentra dentro de la subcarpeta values. Este fichero almacena todas las cadenas que se muestran en los **widgets** (controles, formas, botones, vistas, etc.) de nuestras actividades.

Por ejemplo, si tuvieses un botón cuyo título es *«Presiona aquí»*, es recomendable incluir dicha cadena en tu archivo strings.xml.



Para declarar nuestro recurso de strings usaremos el nodo raíz <resources>. Para declarar las cadenas usaremos la etiqueta <string> y estableceremos el atributo name como identificador. Dentro de esta etiqueta pondremos el texto que se visualizará en el **componente de interfaz**. Esta declaración es similar al uso de la etiqueta <h1>Texto</h1> en **HTML**.

Si abres el archivo, verás que se encuentran tres nodos del tipo <string> : «app_name», «action_settings» y «hello_world».

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Test</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>
```

app_name contiene el nombre de la aplicación. En este caso es «Test», action_settings se refiere al título del menú acciones y hello_world es el string para nuestro mensaje en pantalla la actividad.

El archivo strings.xml es muy útil para los desarrolladores. Una de sus grandes utilidades es facilitar el uso de **múltiples idiomas** en tu aplicación. Esto se debe a que puedes externalizar las cadenas del código java y seleccionar la **versión** del archivo strings.xml con el lenguaje necesitado.

Personalmente te recomiendo que uses este archivo siempre, incluye todo tu texto y no dejes nada por fuera, tu aplicación te lo agradecerá en el futuro.

¿Qué hay dentro de la carpeta layout?

En la carpeta layout encontrarás los archivos de diseño de todas tus actividades. En mi caso existe el archivo activiy_my.xml. Este archivo representa el diseño de la interfaz de mi actividad principal. En el se establecerán todos los widgets que vaya a agregar a la actividad.

Construir la interfaz a través de nodos XML es mucho más sencillo que la creación a través de código Java. Adicionalmente **Android Studio** nos ha dotado de un panel de diseño estilo **Drag and Drop**, lo cual es una bendición para los desarrolladores, ya que facilita demasiado la creación de una interfaz de usuario.

Este archivo de diseño comienza con un nodo raíz llamado <RelativeLayout>. Un Layout es el contenedor principal que define el orden y secuencia en que se organizarán los widgets en nuestra actividad. Existen varios tipos de **Layouts,** como por ejemplo el LinearLayout, GridLayout, FrameLayout, etc.

Android Studio crea por defecto un RelativeLayout porque permite crear un grupo de componentes con ubicaciones relativas.

Quiere decir que se ubicaran por referencias y no por valores absolutos. Esto permite ajustar nuestras aplicaciones a cualquier tipo de pantalla para dispositivos móviles.

Si abres el archivo de diseño de tu actividad verás algo como esto:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/re
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MyActivity">
    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</RelativeLayout>
```

Veamos la utilidad de los atributos para <RelativeLayout>:

- layout_width: Es el ancho que tendrá el layout dentro de la actividad. Aunque se puede especificar con unidades personalizadas, es recomendable usar match_parent para ajustarlo al ancho del dispositivo.

- layout_height: Representa la dimensión vertical del layout. Usa match_parent para ajustarla al dispositivo.

- paddingLeft, paddingRight: Es el espacio lateral existente entre el contorno del Layout y los widgets. Su valor apunta al archivo de recursos dimens.xml ubicado en la carpeta «values». Este archivo contiene nodos de tipo <dimen> con valores density-indepent pixels (**dp**). Para estos paddings se usa el nodo «activity-horizontal-margin», cuyo valor estándar son **16dp**.

- paddingTop, paddingBottom: Es el espacio vertical existente entre el contorno del Layout y los widgets. Su valor es igual a el nodo «activity-vertical-margin» de dimens.xml.

- context: Define el nombre del archivo Java que contiene la actividad donde el Layout será acogido.

Con este diseñador solo ubicamos los componentes donde nos plazca y automáticamente son generados como nodos en nuestro archivo.

Los lectores que han creado páginas web a través de frameworks como **DreamWeaver** podrán entender rápidamente el concepto, debido a que diseñar la interfaz de una aplicación Android es muy similar a crear aplicaciones **HTML**.

La carpeta drawable en Android Studio

Si has sido atento también podemos encontrar el icono de la aplicación en una serie de carpetas que comienzan por el cualificador drawable. Estas carpetas se relacionan directamente con el tipo de **densidad de pantalla** donde se ejecutará nuestra aplicación.

La mayoría de dispositivos móviles actuales tienen uno de estos 4 tipos de densidades:

- **Mediun Dots per Inch**(**mdpi**): Este tipo de pantallas tienen una densidad de 160 puntos por pulgada.

- **High Dots per Inch**(**hdpi**): En esta clasificación encontraremos teléfonos cuya resolución es de 240 puntos por pulgada.

- **Extra high dots per inch**(**xhdpi**): Resoluciones mayores a 340 puntos por pulgada

- **Extra Extra high dots per inch**(**xxhdpi**): Rangos de resoluciones mayores a 480 puntos por pulgada.

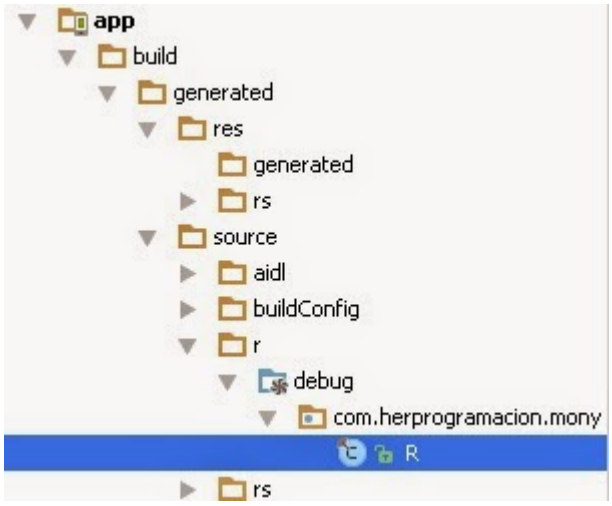
Miremos una pequeña imagen ilustrativa:

La imagen muestra algunos ejemplos de dispositivos móviles cuya densidad se encuentra dentro de los rangos establecidos. Al final podemos ver una categoría extra llamada **xxxhdpi**. Esta denominación describe a la densidad de televisores de última generación que ejecutan Android.

Sería excelente que definieras todos tus recursos gráficos en los distintos tipos de densidades. Esta diversidad permitirá mayor compatibilidad con distintos dispositivos móviles.

Comprendiendo el propósito de la clase R.java

El archivo R.java es una archivo que se autogenera dentro de la carpeta build, para linkear todos los recursos que tenemos en nuestro proyecto al **código Java**.



Si abres el archivo podremos ver un código similar a este:

```
/*AUTO-GENERATED FILE. DO NOT MODIFY. *
* This class was automatically generated bythe
* aapt tool from the resource data itfound. It
* should not be modified by hand. */

package com.TUPAQUETE.test;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int activity_horizontal_margin=0x7f0
        publicstaticfinalint activity_vertical_margin=0x7f040001
    }
    public static final class drawable {
        publicstaticfinalint ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080000;
    }
    public static final class layout {
        publicstaticfinalint activity_my=0x7f030000;
    }
    public static final class menu {
        publicstaticfinalint my=0x7f070000;
    }
    public static final class string {
        public static final int action_settings=0x7f050000;
        public static final int app_name=0x7f050001;
        public static final int hello_world=0x7f050002;
    }
    public static final class style {
        /** Customize your theme here.
         */
        public static final int AppTheme=0x7f060000;
    }
}
```

Como ves, la clase R contiene clases anidadas que representan todos los recursos de nuestro proyecto. Cada atributo tiene un dirección de memoria asociada referenciada a un recurso en específico. Por ejemplo, la clase string posee el atributo hello_world, el cual representa nuestro TextView en la actividad principal. Este recurso está ubicado en la posición0x7f050002.

No modifiques el archivo R.java, el se actualiza automáticamente al añadir un nuevo elemento al proyecto.

La carpeta java de un proyecto en Android Studio

Justamente era lo que iba a decir. Finalmente llegamos a la carpeta «java». Aquí se alojarán todos los archivos relacionados con nuestras actividades y otros archivos fuente auxiliares.

Al abrir nuestro archivo MyActivity.java veremos toda la lógica necesaria para que la actividad interactúe de manera correcta con el usuario.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MyActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu){
// Inflate the menu;this adds items to the actionbar if it is pr
        getMenuInflater().inflate(R.menu.my, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item){
// Handle action bar item clicks here. The actionbar will
// automatically handle clicks on the Home/Upbutton, so long
// as you specify a parent activity inAndroidManifest.xml.
        int id = item.getItemId();
        if(id == R.id.action_settings){
            return true;
        }
```

```
        return super.onOptionsItemSelected(item);
    }
}
```

Para toda actividad creada debemos **extender** una subclase de la superclase Activity. Más adelante veremos que una actividad tiene un **ciclo de vida** y lo único que esta bajo nuestro control es la manipulación de cada estado.

Si ya notaste, las actividades no tiene un punto de entrada main(). Esto se debe a que las aplicaciones **Android**parten de múltiples puntos de ejecución. Pero esa característica será objeto de estudio del próximo artículo.

Conclusión

¿Habías tenido problemas al entender la jerarquía de un proyecto en Android Studio?

Es de gran importancia tener claro las funciones de cada directorio si deseas comenzar a desarrollar en esta herramienta.

Es inevitable que Android Studio se actualice constantemente para mejorar su calidad como producto, sin embargo la jerarquía interna de la composición del proyecto es poco probable que cambie.

Ahora es el turno de tomar un tutorial básico del lenguaje Xml para comprender el manejo de recursos.