



T1. INTRODUCCIÓN AL DESARROLLO EN ANDROID STUDIO

Santiago Rodenas Herráiz

2º DAM (PMAM) 2022/2023








Información general

- Tecnologías móviles
 - Comienzos de Android
 - Algunos conceptos
 - Componentes App Android
 - Fichero manifest
 - Entorno de desarrollo
 - AVD
 - Estructura de un proyecto
 - Activity y ciclo de vida
 - Primer contacto con eventos
 - Intent
-

Tecnologías móviles



Resumen

1G	2G	3G	4G	5G
				
1981	1992	2001	2010	2020
2 Kbps	64 Kbps	2 Mbps	100 Mbps	10 Gbps
Servicio básico de telefonía analógico	Servicio básico de telefonía digital (GSM) + mensajes de texto	Llega Internet al móvil	Banda ancha real (video HD)	Internet de las cosas

Tecnologías móviles



- **Primera Generación**

- Tecnologías analógicas. Sólo se permitían llamadas y surgieron entre los años 70 y 80.
- Eran grandes con baterías que duraban muy poco.
- Demasiado caros para cualquier persona.
- Surgieron en EEUU.
- Movilidad reducida.



Tecnologías móviles



- **Segunda Generación**

- Surge la tecnología digital.
- Aparecen los SMS y el concepto de Roaming en el extranjero.
- Se conoce como la tecnología GSM
- Los precios son más asequibles.
- Principios de la navegación por internet.
- Intercambio de imágenes.
- Nace la blackberry. Ya desaparecida.



Tecnologías móviles



- **Tercera Generación**

- Se mejora la transmisión de datos mediante la incorporación de contenido multimedia.
- Podemos hacer videoconferencias.
- Aparecen aplicaciones como youtube.
- Se mejora el potencial de sus antenas, mejorando el roaming.
- Da igual en qué país estés, podrás navegar por internet.
- Mayor calidad en voz y datos.



Tecnologías móviles



- **Cuarta Generación**

- Es la era del smartphone
- Mejoran las antenas
- Podemos ver una película en tiempo real.
- Velocidad comparada con la fibra óptica



Tecnologías móviles



- **Quinta Generación**

- Estamos hablando de velocidades 100 veces la anterior 4G.
- Mayor cantidad de dispositivos conectados. Internet de las cosas.
- Se incluyen las casas inteligentes, los coches autónomos.
- Se pueden realizar operaciones con robots en tiempo real.
- Es la era de la inteligencia artificial.



Resumen



- Con más de 3000 millones de dispositivos, 10 años y 15 nombres en clave, tenemos a este sistema como el más utilizado en la actualidad.
- Todo empezó en el 2005 cuando Google compró a Android. Se tomó la decisión de utilizar linux como base del S.O.
- A finales del 2008 se realizó el primer lanzamiento de Android. Durante el 2009 se lanzaron cuatro versiones. Cupcake y Donut.
- En octubre del 2009 se lanzó Android 2.0 y hasta la fecha de hoy con Android 11.0
- Como datos curiosos, por cada liberación conocida por un número de versión y un nivel de API, casualmente su nombre representa un postre.

Resumen

Nombre código ↕	Número de versión ↕	Fecha de lanzamiento ↕	Nivel de API ↕
Apple Pie ¹	1.0	23 de septiembre de 2008	1
Banana Bread ¹	1.1	9 de febrero de 2009	2
Cupcake	1.5	25 de abril de 2009	3
Donut	1.6	15 de septiembre de 2009	4
Eclair	2.0 – 2.1	26 de octubre de 2009	5 – 7
Froyo	2.2 – 2.2.3	20 de mayo de 2010	8
Gingerbread	2.3 – 2.3.7	6 de diciembre de 2010	9 – 10
Honeycomb ²	3.0 – 3.2.6	22 de febrero de 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.5	18 de octubre de 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	9 de julio de 2012	16 – 18
KitKat	4.4 – 4.4.4	31 de octubre de 2013	19 – 20
Lollipop	5.0 – 5.1.1	12 de noviembre de 2014	21 – 22
Marshmallow	6.0 – 6.0.1	5 de octubre de 2015	23
Nougat	7.0 – 7.1.2	15 de junio de 2016	24 – 25
Oreo	8.0 – 8.1	21 de agosto de 2017	26 – 27
Pie	9.0	6 de agosto de 2018	28
Android 10 ³	10.0	3 de septiembre de 2019	29

Versiones y números de versión de Android

Algunos conceptos necesarios

- **API (Application Programming Interface)** Es un conjunto de funciones y procedimientos que permiten a los desarrolladores interactuar con el S.O.
- ● **App (Aplicación informática)** —
- **Framework** es un conjunto de módulos
- **Kernel** es la parte principal y fundamental del S.O.
- **Runtime** es el intervalo de tiempo en el que una app se ejecuta en el S.O.
- **UI** es la interfaz de usuario.
- **Bundle** es la forma que tendremos de pasar información entre las diferentes pantallas de la app.
- **Activiti** es cada una de las pantallas de nuestra app.
- **SDK** es el conjunto de herramientas que necesitamos en el desarrollo de una app. Herramientas para el programador.
- **Gradle** es el sistema de compilación para Android. Incluye un conjunto de dependencias que hace crecer nuestro proyecto. Integra a terceros y ofrece integración en varios lenguajes.
- **Manifest** es nuestro fichero de proyecto, donde se deben declarar todas las actividades y componentes entre otras.

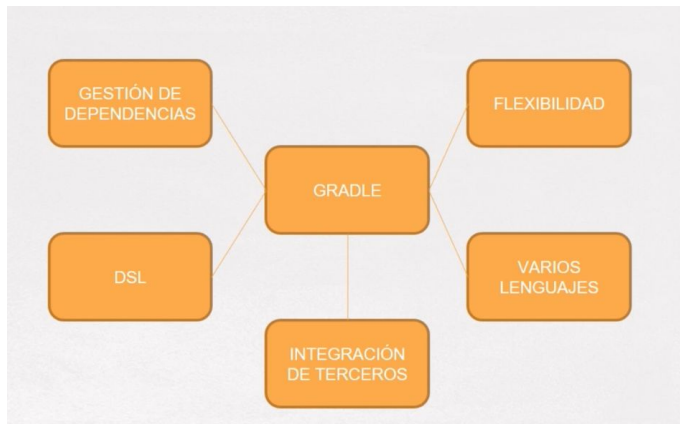
Algunos conceptos necesarios

- Gradle (Application Programming Interface)

- Se basa en JVM.
- Es una tecnología que sirve para la administración de dependencias.
- Nos encontramos el fichero gradle en proyecto y módulo en diferente directorio.
- Grandle en proyecto
 - Indicamos los repositorios que son google y jcenter.
 - Indicamos la versión de gradle.
- Grandle en módulo
 - Podemos tener varios módulos dentro del mismo proyecto.
 - Indicamos la versión mínima para compilar.
 - Indicamos el id o nombre del paquete con el que se identificará en el store.
 - La versión de nuestro módulo/app.
 - dependencias. Traemos lo que nos interesa. Por ejemplo
 - `implementation 'androidx.constraintlayout:constraintlayout:2.1.4'`
 - `implementation 'androidx.appcompat:appcompat:1.5.0'`

Algunos conceptos necesarios

- Gradle (Application Programming Interface)
 - Importante. Siempre que hay un cambio en el fichero gradle, hay que hacer una sincronización.
 - Si detecta un error, no podremos pasar el proceso de compilación.
 - DSL. Gradle se escribe en lenguaje Groovy o Kotlin.

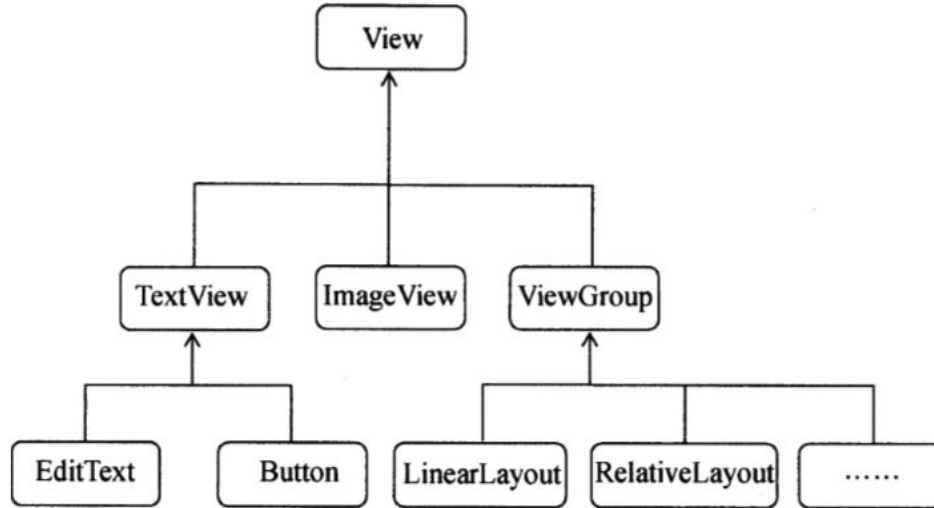


Aplicación en Android (Componentes)

- **View** Son las partes fundamentales de las que se componen las Interfaces de Usuario.
- **Activity** Podemos decir que son cada una de las pantallas de nuestra app
- **Layout** Se puede decir que son los contenedores de cada uno de los componentes que conforman nuestra Interfaz gráfica.
- **Service** Son notificaciones o aplicaciones en segundo plano sin interfaz de usuario, que interaccionan con nuestras app. Por ejem., cargar una lista de canciones.
- **Intent** Objetos que permite la comunicación entre diferentes activities.
- **Proveedores de contenido** Permite el acceso a los datos entre una app y una fuente de datos.
- **Broadcast receive** Controla la forma de recibir anuncios del sistema. Ejem., llamadas entrantes, mensajes, batería baja.
- **Fragments** Parte de una activity. Permite adaptar nuestras app a diferentes pantallas y ofrece mucha más eficiencia.

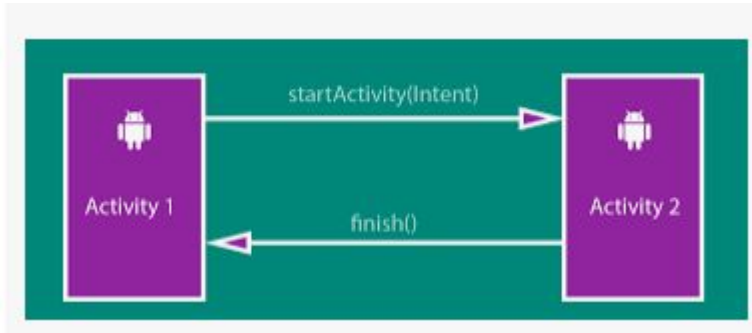
Aplicación en Android (Componentes)

- **View** Son las partes fundamentales de las que se componen las Interfaces de Usuario.



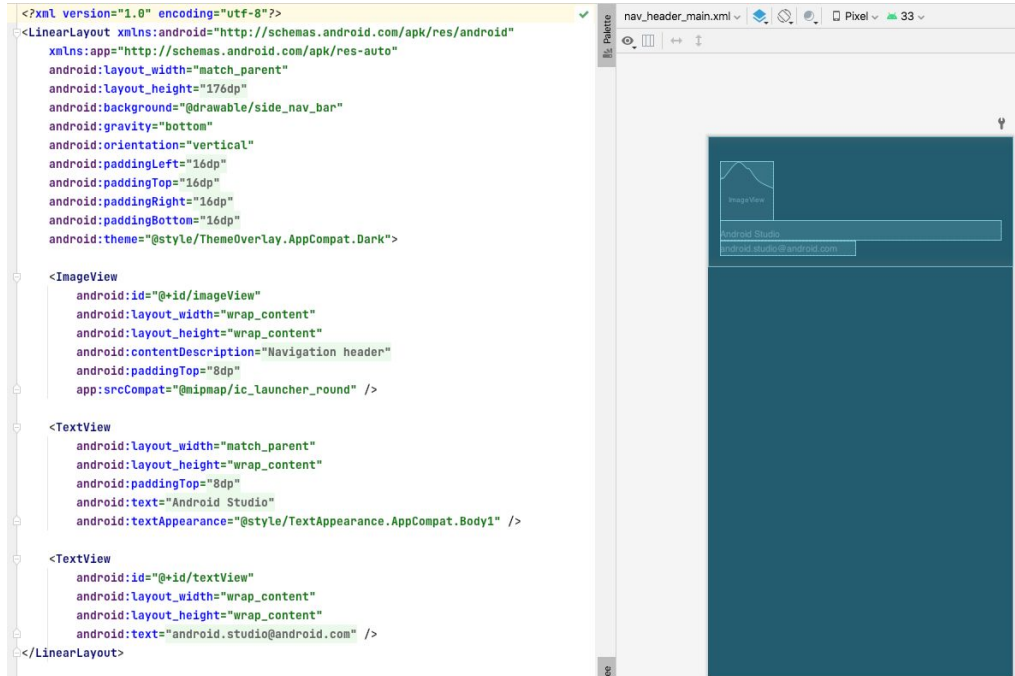
Aplicación en Android (Componentes)

- **Activity** Podemos decir que son cada una de las pantallas de nuestra app



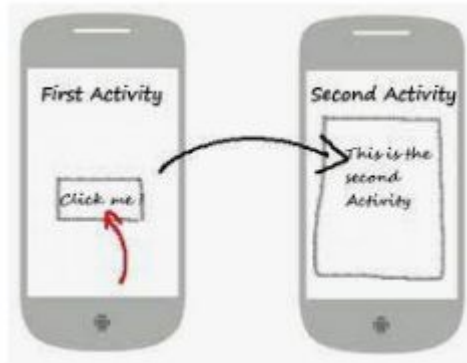
Aplicación en Android (Componentes)

- **Layout** Se puede decir que son los contenedores de cada uno de los componentes que conforman nuestra Interfaz gráfica.



Aplicación en Android (Componentes)

- **Intent** Forma que tenemos de llamar a un activiti u otro componente desde otro .
 - Por supuesto, podemos pasarle datos, a partir de un objeto de tipo Bundle.
 - Veremos dos tipos de Intent (Implícitos y Explícitos)



Aplicación en Android (Componentes)

- **Service** Son notificaciones o aplicaciones en segundo plano sin interfaz de usuario, que interaccionan con nuestras app.
 - En el ejemplo de una lista dinámica con canciones MP3.
 - ¿Nuestra APP debe esperar a que carguen dinámicamente todas las canciones MP3?
 - Lo más eficiente es crear un servicio que sea un thread y se ejecute en segundo plano.
 - Nuestra app no tiene porqué bloquearse a la espera de la carga.
 - Deben declararse en nuestro fichero manifest.

```
<manifest ... >
  ...
  <application ... >
    <service android:name=".ExampleService" />
    ...
  </application>
</manifest>
```

Aplicación en Android (Componentes)

- **Service** en segundo plano. El usuario no lo nota.
- Debe heredar de Service o IntentService (ya antiguo). Documentación
- Debe sobrescribir onCreate y onDestroy entre otros.

```
public class ServicioMusica extends Service {

    MediaPlayer miReproductor;

    public void onCreate(){

        super.onCreate();

        miReproductor=MediaPlayer.create(this,R.raw.cancion2);

        miReproductor.setLooping(true);

        miReproductor.setVolume(100,100);

    }

    public int onStartCommand(Intent intent, int flags, int startId){

        miReproductor.start();

        return START_STICKY;

    }

}
```

```
public void onDestroy(){

    super.onDestroy();

    if(miReproductor.isPlaying()) miReproductor.stop();

    miReproductor.release();

}
```

```
public void enciendeMusica(){

    botonMusica.setImageResource(R.drawable.musica2);

    Intent miReproductor=new Intent(getActivity(), ServicioMusica.class);

    getActivity().startService(miReproductor);

    public void apagaMusica(){

        botonMusica.setImageResource(R.drawable.musica);

        Intent miReproductor=new Intent(getActivity(), ServicioMusica.class);

        getActivity().stopService(miReproductor);

    }

}
```

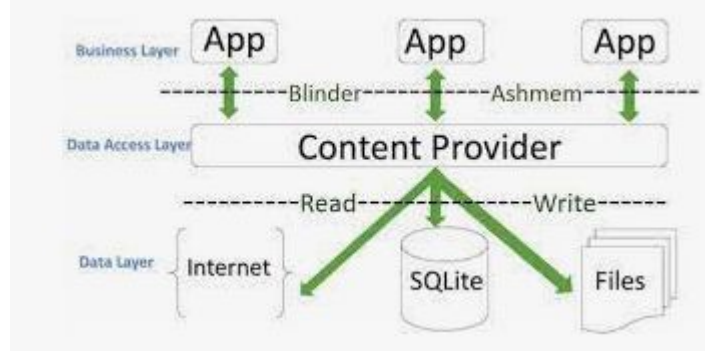
Desde el Activity

- El activity inicia el servicio a partir de un objeto Intent.
- Al crearse el Servicio, llamará al método onStartCommand.
- En cualquier momento, podemos destruirlo y se llamará a onDestroy

Aplicación en Android (Componentes)

- Proveedores de contenido

- Es tremendamente importante disponer de APIs que permitan encapsular la forma que tienen las aplicaciones del acceso a los datos tanto en local como en remoto.
- Trabajaremos con los proveedores de contenido más adelante, sobre todo en nuestro objetivo de trabajar con REST.



Aplicación en Android (Componentes)

- Broadcast receive

- El sistema emite diferentes mensajes o avisos como batería baja, final de la carga, llamada entrante, etc.
- Nuestras app, pueden registrarse en ciertas emisiones para que cuando ocurran, éstas sean notificadas.
- Al igual que en los servicios, también deben añadirse en el manifest.
- Un ejemplo.
 - Crear una alarma y en su configuración, notificará a nuestra app mediante una clase que herede de BroadcastReceiver, después de un cierto tiempo..
 - Cuando la alarma se active, nuestra app se dará por enterada y ejecutará lo programado.



Aplicación en Android (Componentes)

- Fragments



Fichero manifest

- Fichero Manifest

- Las actividades deben registrarse dentro de este fichero.
- Los servicios y BroadcastReceiver también deben de registrarse.
- La activity principal, posee un atributo de tipo <intent-filter> en el que indica que es la activity que se mostrará al ejecutarse nuestra app y que es capaz de recibir eventos.
- Se deben de registrar los permisos que nuestra app debe tener, por ejemplo.
 - `<uses-permission android:name="android.permission.INTERNET" />`
- Contiene la API mínima que requiere nuestra app.
- La propiedad Application, contiene metadatos de nuestra App.
 - El icono de nuestra aplicación.
 - El título de nuestra aplicación.
 - El tema.
- Nombre del paquete, versión y nombre.

Fichero manifest

- Fichero Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Ejemplo sencillo de manifest con distintos activities
... PMDM 2023
... santi
-->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.primerintent">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="PrimerIntent"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PrimerIntent">
        <activity
            android:name=".DatosActivity2"
            android:exported="false" />
        <activity
            android:name=".DatosActivity"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

Entorno de desarrollo (IDE)

- Android Studio
 - [Descarga.](#)
 - [Documentación.](#)

- 1 La **barra de herramientas** te permite realizar una gran variedad de acciones, como ejecutar tu app e iniciar las herramientas de Android.
- 2 La **barra de navegación** te ayuda a explorar tu proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana **Project**.
- 3 La **ventana del editor** es el área en la que puedes crear y modificar código. Según el tipo de actividad actual, el editor puede cambiar. Por ejemplo, cuando ves un archivo de diseño, el editor muestra el Editor de diseño.
- 4 La **barra de la ventana de herramientas** se encuentra afuera de la ventana del IDE y contiene los botones que te permiten expandir o contraer ventanas de herramientas individuales.
- 5 Las **ventanas de herramientas** te brindan acceso a tareas específicas, como la administración de proyectos, la búsqueda, el control de versiones, entre otras. Puedes expandirlas y contraerlas.
- 6 En la **barra de estado**, se muestra el estado de tu proyecto y el IDE, además de advertencias o mensajes.

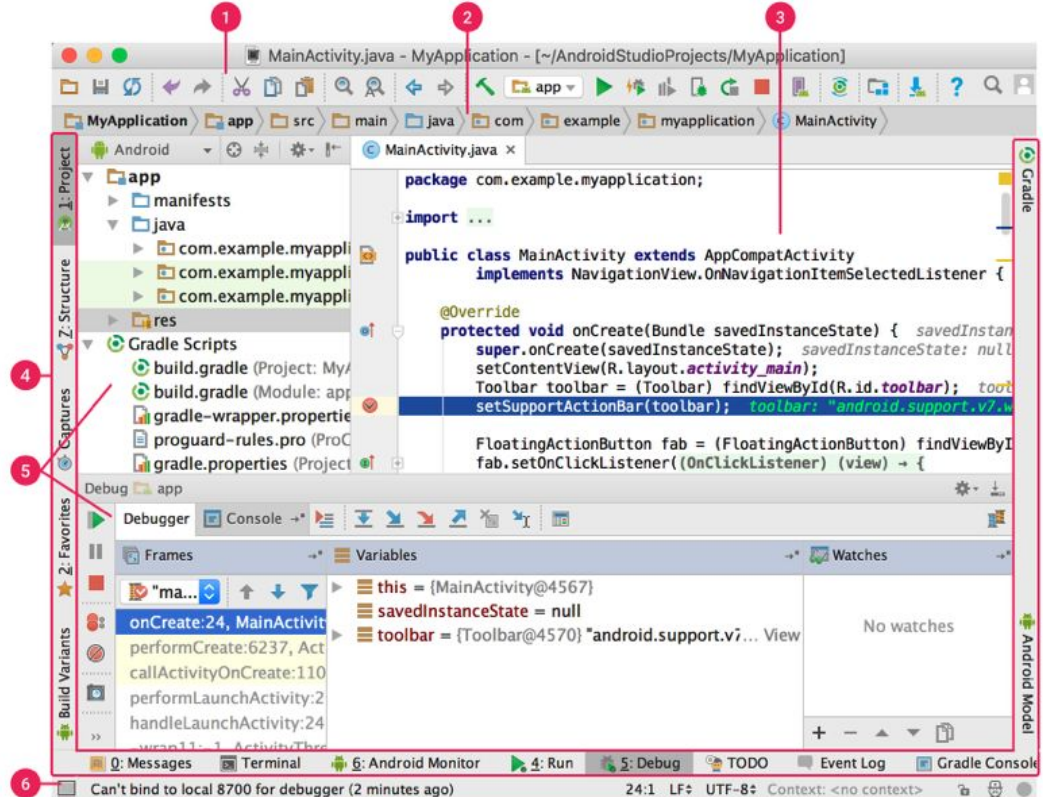


Figura 3: Ventana principal de Android Studio

Entorno de desarrollo (IDE)

- Actividades de clase

- Instalación de Android Studio.
- Configurar el color y tema principal del IDE.
- Modificar el tamaño del texto.
- Configurar el tamaño máximo de ram.
- Crear un primer proyecto y configurar con API 30
- Visualiza el fichero manifest y examinar las dependencias de módulo.
- Realiza un recorrido en vista proyecto, y localiza el código de la activity.
- Examina el código del manifest y explica cada una de las líneas que contiene.
- Sobre el código de la activity, posiciona el ratón sobre el recurso `R.layout.activity_main`.
Abrir dicho fichero y examinar lo que contiene.
- Explicar la diferencia entre vista y lógica.

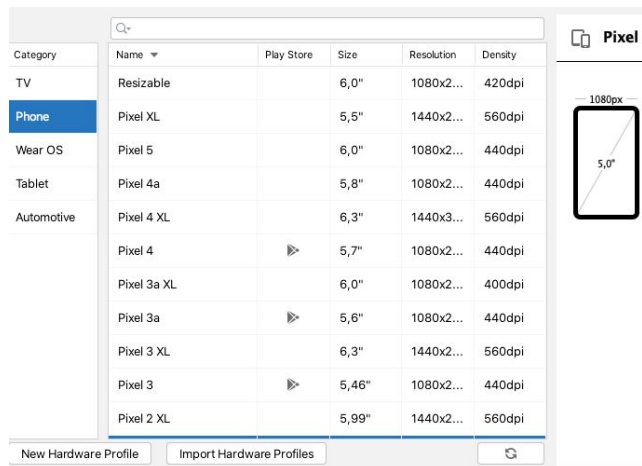
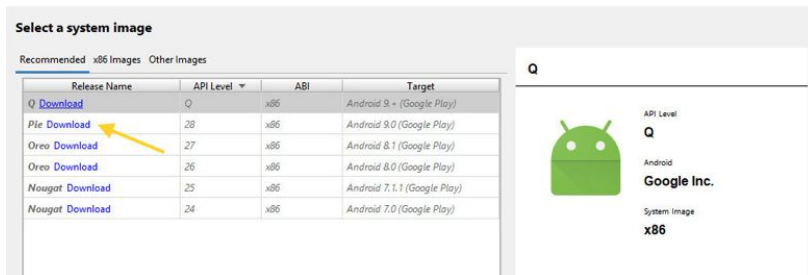
Entorno de desarrollo

- **Android Studio**, es un sistema nativo porque, lo que genera a partir de código java/kotlin, son app compiladas específicamente para este sistema.
- Otras tecnologías como **flutter**, són híbridas porque pueden ser ejecutadas tanto en Android como IOS, mediante la utilización de APIS específicas para una y otra.
- Al final, podemos desarrollar app móviles utilizando una tecnología y otra, pero si queremos sistemas para Android, lo razonable es utilizar Android Studio.
- Si nuestro objetivo es llegar a diferentes tecnologías como IOS o Android, podemos plantearnos otras alternativas.

AVD

- Avd

- Android studio permite crear diferentes emuladores dependiendo del dispositivo y de la API.
- Se gestiona a partir del Virtual Devices.
- Existen un conjunto de imágenes del sistema. Android studio te recomendará dependiendo de la arquitectura hard que tengas.
- Como mínimo, debes tener 8GB de RAM y un i5.
- Se puede acelerar el emulador.



AVD

- Actividades de clase
 - Crear vuestro primer emulador con API 30
 - Buscar información de cómo acelerarlo.

AVD

Conectar vuestro móvil a Android Studio

1. Ir al SDK Manager > SDK Tools > verificar si tenemos el Google USB Driver. Si no lo tenemos, hay que instalarlo.
2. En vuestro dispositivo móvil, hay que ir a Ajustes > Acerca del teléfono > ir a la información del software o versión > Pulsar al menos 5 toques sobre Numero de compilación. Debemos de ver el mensaje de “Se activo modo desarrollador”.
3. Ir a opciones del desarrollador y activar la depuración por USB. Nos preguntará si permitimos y pulsamos Aceptar.
4. Desde el SO de windows, nos preguntará si Permitimos desde esta computadora. Decimos que permitir.
5. Ya nos aparecerá nuestro dispositivo dentro del Android Studio.
6. Ahora podemos ejecutar nuestra App en nuestro dispositivo móvil físico.
7. Es aconsejable eliminar versiones a la vez que nos instala después de múltiples ejecuciones. Hay que evitar basura.
8. Información de la [página oficial](#).

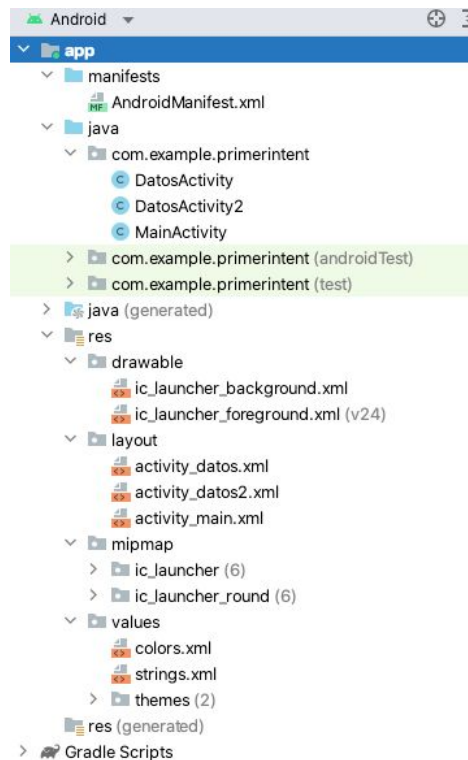
Estructura de un proyecto Android

Estructura sencilla

- manifest
- Directorio java con su paquete
- Directorio res (recursos)
 - Directorio drawable
 - Directorio layout
 - Directorio mipmap
 - Directorio values
- Ficheros gradle

build.gradle (Project: PrimerIntent)

build.gradle (Module: PrimerIntent.app)

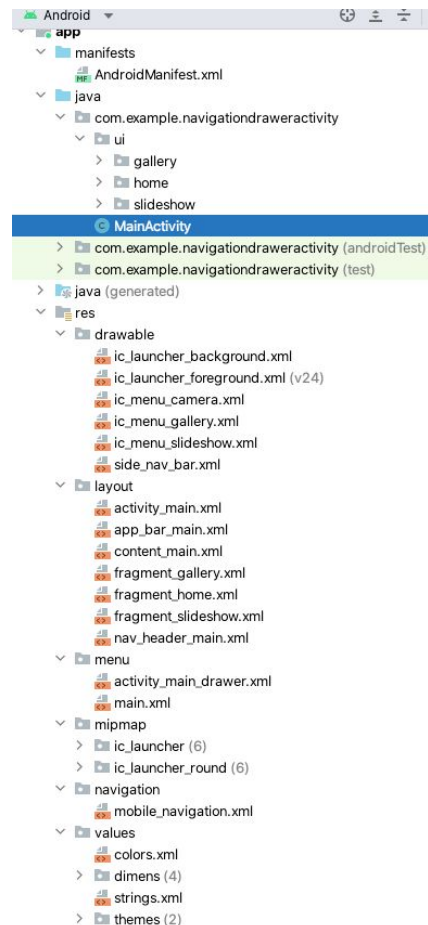
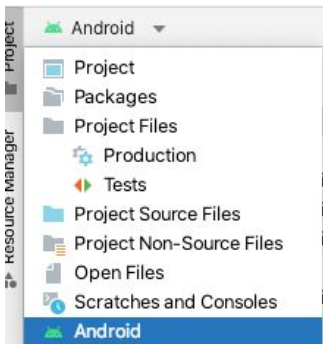


Estructura de un proyecto Android

Otra estructura más completa

- Directorio menu
- Directorio navigation

Diferentes vistas de nuestros ficheros



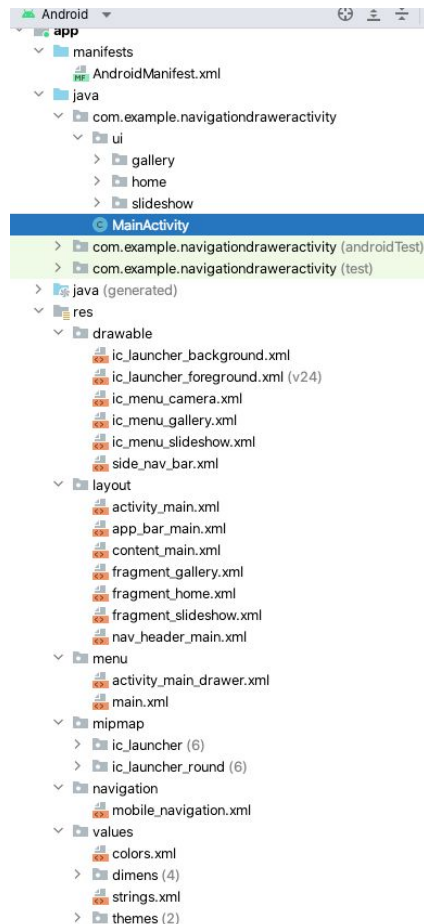
Estructura de un proyecto Android

Directorio java

- Contiene todas las clases y Activitis organizadas previamente en sus correspondientes paquetes. Es aconsejable no mezclar.
- Consejo. Crear paquetes diferentes dependiendo de la arquitectura de vuestra aplicación.

Directorio res

- Son nuestros recursos de la app. Se acceden a partir de la clase **R** (conex entre Interfaz y lógica)
- **drawable**. Tenemos nuestros iconos vectoriales en formato xml. Son diseños vectoriales como **VectorDrawable** y **AnimatedVectorDrawable**
 - VectorDrawable. Cada path, contiene la geografía del esquema del objeto.
 - Podemos utilizar la herramienta Vector Asset Stdudio. New > Vector Asset.



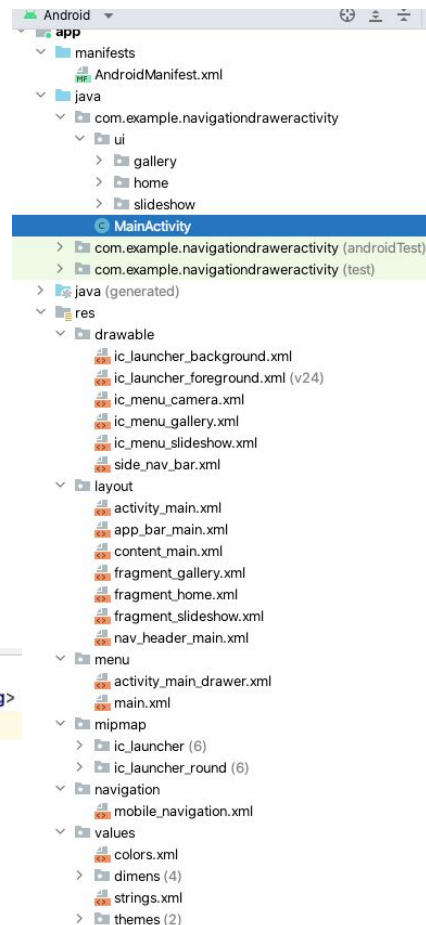
Estructura de un proyecto Android

Directorio res

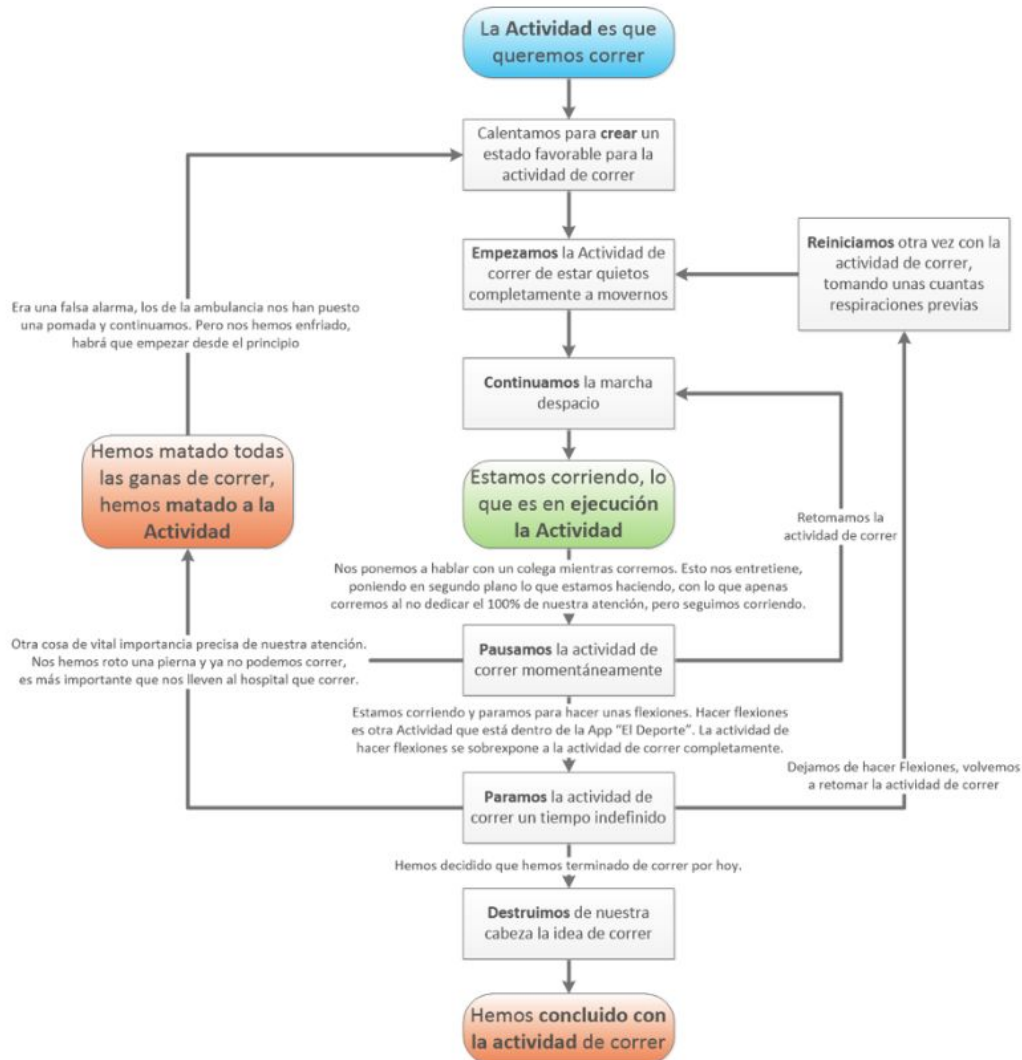
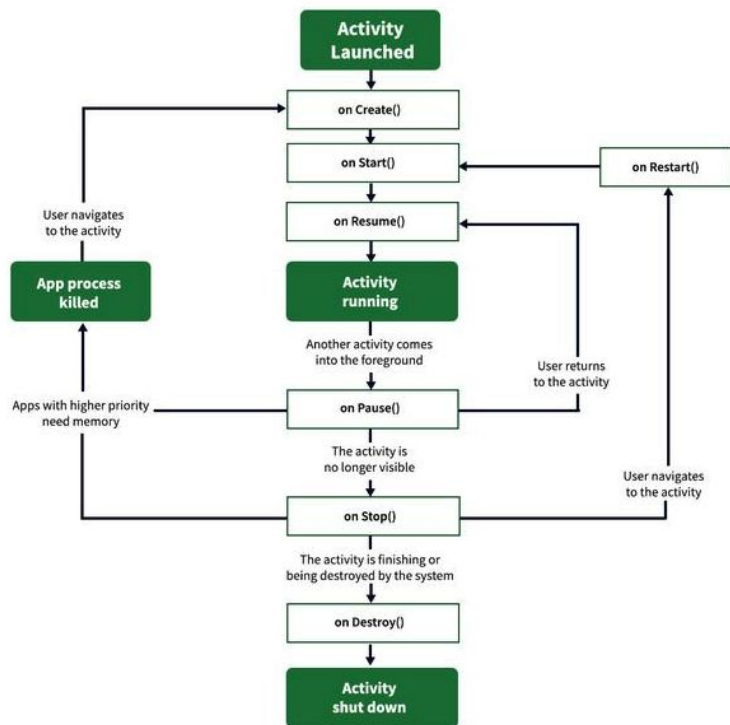
- **layout.** Son nuestros diseños en formato xml. Representan las vistas.
- **values.** Recursos como la definición de colores, definición de cadenas, diferentes temas definiendo estilos (por ejemplo para menú de Opciones)

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
  <style name="Theme.PrimerIntent" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/purple_500</item>
    <item name="colorPrimaryVariant">@color/purple_700</item>
    <item name="colorOnPrimary">@color/white</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/teal_200</item>
    <item name="colorSecondaryVariant">@color/teal_700</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
  </style>
</resources>
```

```
<resources>
  <string name="app_name">PrimerIntent</string>
</resources>
```



Activity y su ciclo de vida



Activity y su ciclo de vida

- **onCreate()**

- Se llama a este método, inmediatamente de crearse el activity. Se inflan las vistas, obtenemos las referencias de los elementos, asignar listeners. La activity no es visible aún.

- **onStart()**

- Es llamado después del método onCreate. Es justamente cuando está apunto de verse visible para el usuario.

- **onResume()**

- Es llamado después de onStart(). La activity está en primer plano (ponerlo a la pila de las Activitys) y el usuario ya puede interactuar con los elementos de la interfaz.

<EJECUCIÓN DEL ACTIVITY>

Activity y su ciclo de vida

- **onPause()**

- Es llamado cuando la activity pierde el foco o deja de estar en primer plano. Puede ser por:
 - El usuario presiona el botón de HOME
 - El usuario presiona el botón de aplicaciones recientes.
- La activity se vuelve parcialmente visible y podemos tomar dos caminos:
 - onResume() cuando la actividad vuelve a estar en primer plano.
 - onStop() cuando la actividad deja de ser visible al usuario.

- **onStop()**

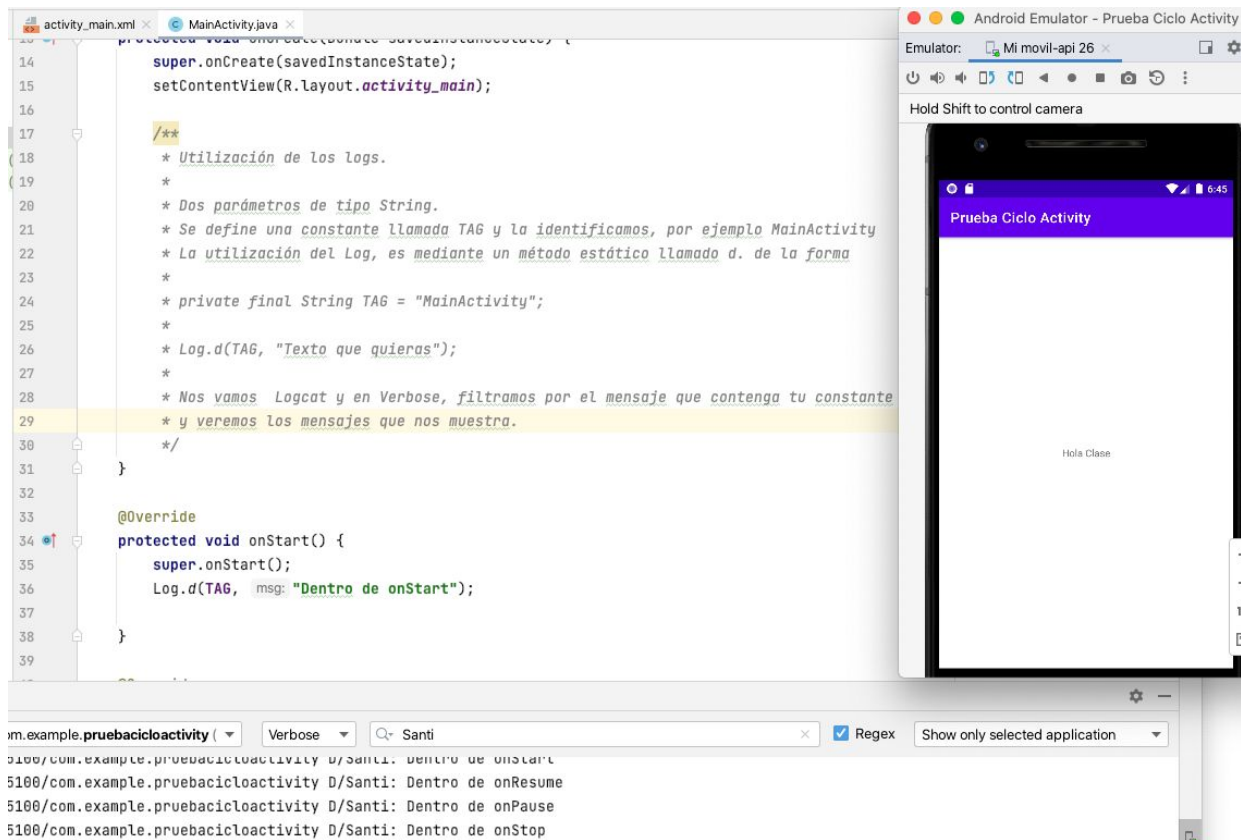
- Es llamado cuando el activity ya no es visible por algún motivo del onPause(). Puede tomar tres caminos:
 - onStart() si la actividad vuelve a estar visible para el usuario
 - onDestroy() si la actividad muere.
 - El S.O. puede bloquear la activity por cualquier motivo pero debe volver a crearse.

Activity y su ciclo de vida

- **onDestroy()**

- La actividad es destruída bien por el usuario o por el S.O.
- Permite liberar recursos.

Estado	¿En memoria?	¿Visible para el usuario?	¿En primer plano?
Inexistente	No	No	No
Detenida	Sí	No	No
Pausada	Sí	Parcialmente	No
En ejecución	Sí	Sí	Sí



Activity y su ciclo de vida

- **MainActivity** hereda de **AppCompatActivity** (`androidx.appcompat.appcompat:1.4.1`)
 - Forma parte de la biblioteca que da soporte de compatibilidad para que la app se **ejecute en la versión inferior del sistema Android**.
 - **Appcompat** apareció por primera vez en la biblioteca de compatibilidad de V7 y ahora veremos que se ha migrado a **AndroidX**.
 - ¿Qué sucede si sale una API 33 (que lo hay) con muchas mejoras y sin embargo llega a un 20% de dispositivos? Lo que se hacía antes, era diseñar nuestras App con versiones antiguas, aun pudiendo tener mejoras con la nueva API 33.
 - En la versión 3.0 aparecieron importantes novedades como los (fragment). Como no era viable incluir la 3.0, sacaron una librería de compatibilidad para que pudieran ser soportados con versiones antiguas (**con límites**).

Activity y su ciclo de vida

- Las librerías de compatibilidad, nos permiten aprovechar las últimas novedades con (LIMITACIONES, NO TODAS PUEDEN), en dispositivos más antiguos.
- AppCompatActivity hereda de Activity con funcionalidad añadida.
 - Por ejemplo, proporciona funciones de ActionBar
- **setContentView** hace que se infle o cargue en memoria la IU para agregarlo al contexto de la actividad (pantalla).
- La biblioteca de compatibilidad de Android ha dejado de ser operativa y ahora tenemos la AndroidX. Esta biblioteca reemplaza completamente la biblioteca de Android.

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

Activity y su ciclo de vida

- **¿Cual es el motivo por el cual es recomendable actualizar una app frente a una nueva actualización de nuestro sistema Android?**
 - Mejorar el rendimiento de nuestra aplicación.
 - Mejorar su interfaz con algunas novedades que permita una API más actual.
 - Producto inacabado.
 - Incluir cambios en la app.
 - Incluir nueva funcionalidad.
- **¿Qué sucede si ponemos una restricción de instalación de API para un emulador inferior?**
 - Podemos crear un proyecto para una API determinada, indicará que si ese dispositivo **no tiene esa versión, no podrá ejecutarse (lógico, no tiene esa funcionalidad)**. Por ejemplo, creemos una app con fragmentos y nuestro emulador tiene una versión inferior a esa API. No contendrá esa funcionalidad. Ahora, si incluimos en nuestro proyecto una librería de compatibilidad, nuestra app hará compatible los fragmentos de alguna manera, con nuestro emulador cuya versión de API es inferior. Al final, lo que hace es incorporar una capa intermedia que engordará el proyecto.

Primer contacto con los eventos

- Los eventos son sucesos que reciben ciertos objetos o elementos de la interfaz, cuyo objetivo es realizar una acción.
- • Trabajaremos en primer lugar con los botones. —
- ¿Qué haremos?
 - Diseñar nuestra primera vista con un simple TextView y un Button.
 - Asociar los elementos de la interfaz con su lógica.
 - Crear el método que capture el evento asociado a pulsar sobre el botón.

Primer contacto con los eventos

- Asociar el método dentro de la vista.

```
<Button
    android:id="@+id/btn_pulsar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:text="Pulsar"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/txtSaludo"
    android:onClick="saludarASanti"/>
```

Create 'saludarASanti(View)' in 'MainActivity' >

Press `⌘Space` to open preview

paintLayout>

```
public void saludarASanti(View view) {
    Toast.makeText(context, this, text: "Hola Santi, que tal", Toast.LENGTH_SHORT).show();
}
```

Primer contacto con los eventos

- Dentro del método onCreate, asociamos vista y lógica de ambos view.
- Modificamos el método de saludo, escribiendo en el textView el saludo.

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

    txtSaludo = findViewById(R.id.txt_saludo);
    bntPulsar = (Button) findViewById(R.id.btn_pulsar);
}
```

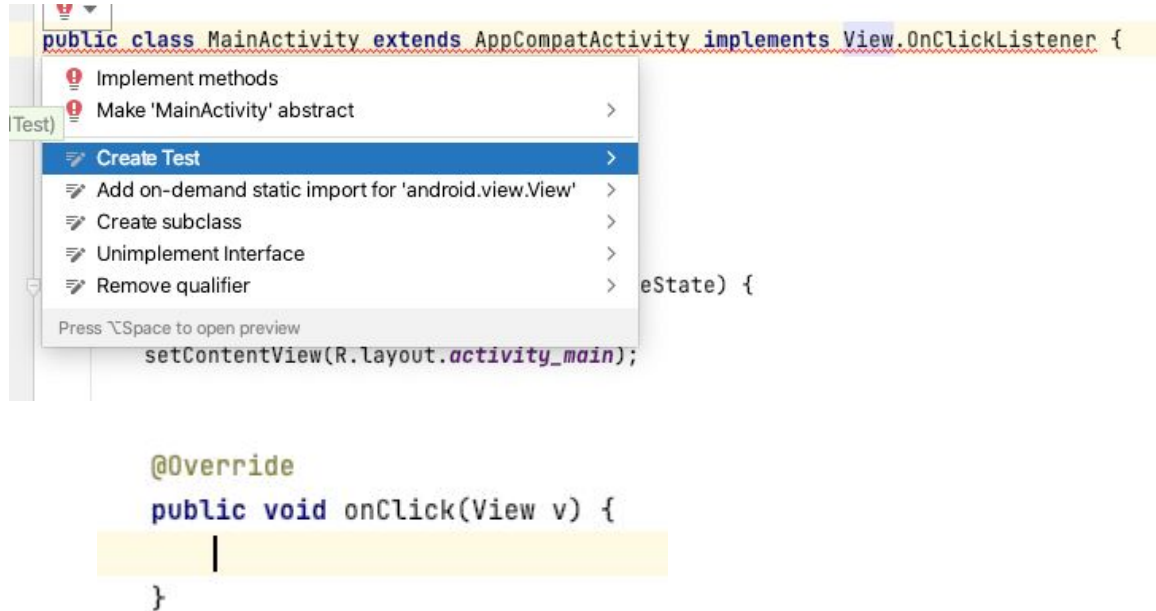
findViewById, conecta el elemento gráfico con su lógica a partir de la clase R. Más adelante, veremos otra forma con **binding**.

```
public void saludarASanti(View view) {
    String saludo="Hola santi, que tal";
    Toast.makeText( context: this, saludo,Toast.LENGTH_SHORT).show();
    txtSaludo.setText(saludo);
}
```



Primer contacto con los eventos

- Haremos lo mismo, pero implementando directamente de la interfaz.



Primer contacto con los eventos

- Haremos lo mismo, pero con clases anónimas. La forma más elegante.



```
bntPulsar.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // ...  
    }  
});
```


Primer contacto con los eventos

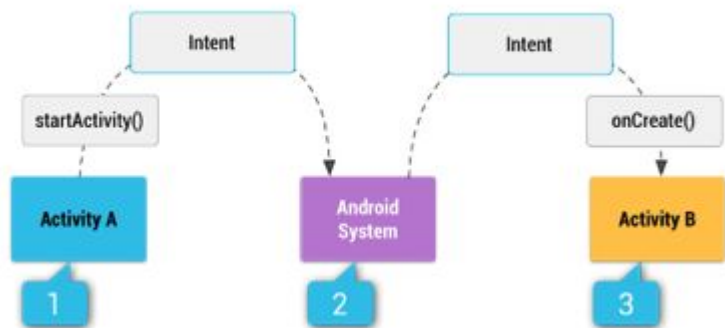
- Actividad.

— Crear una vista con tres botones que implemente las tres formas de capturar el evento de un botón.

Intent

Los intent son unos objetos que permiten llamar a un activity/componente, desde otra activity. Representan la intención de hacer algo.

- Iniciar una actividad. Iniciar otra pantalla.
- Iniciar un servicio. Realizar operaciones en segundo plano.
- Transmitir una emisión. Cualquier aviso que una aplicación puede recibir.



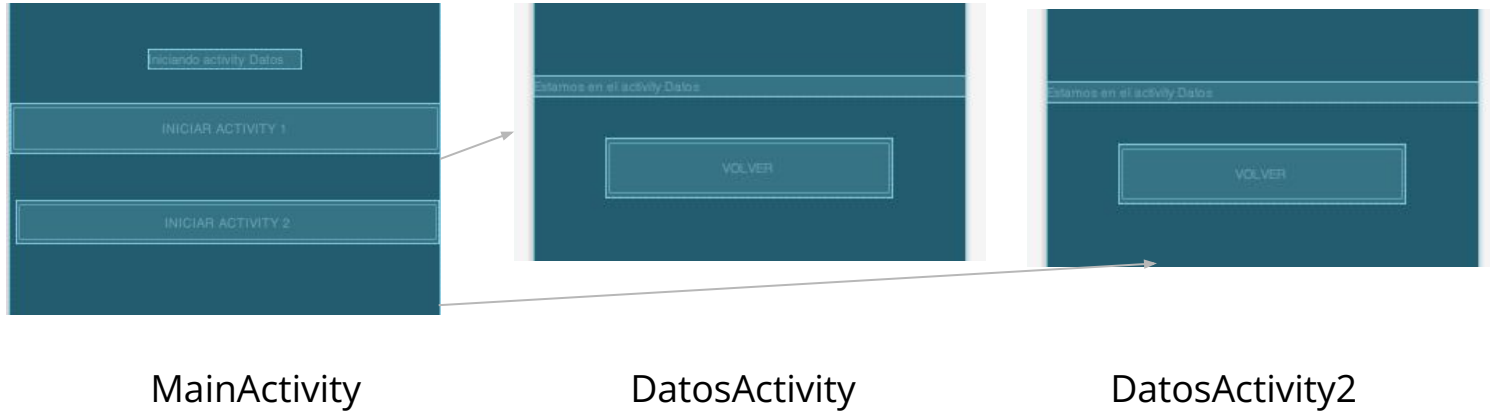
- La Activity B, se lanza mediante `startActivity` de la Activity A.
- El S.O. toma el control y crea el activity B.
- Este tipo de intent, se llaman implícitas.

Intent Implícitos

- Crearemos un objeto de tipo Intent, asignándole dos parámetros
 - El activity donde nos encontramos
 - El activity a donde queremos ir
 - Llamar a startActivity()
-
- **Ejemplo:** Crearemos un activity con dos botones. Cada uno iniciará un activity diferente, por tanto tenemos el ActivityMain, DatosActivity y AdatosActivity2.
 - Cuando accedamos a un DatosActivity, podremos volver al principal mediante un botón de volver.
 - También pasaremos parámetros o información desde un activity a otro.
 - El paso de parámetros vendrá por el método putExtra. Pasaremos una clave que es la identificación del parámetros y después el valor.
 - i.putExtra("nombre_parametro", valor_parametro);

Intent Implícitos

-



Intent Implícitos

- Desde el MainActivity

```
public void iniciarActividad(View view) {  
    /**  
     * Necesitamos el contexto (this)  
     * Necesitamos el activity que quiero iniciar.  
     * Pasaremos también parámetros al intent.  
     */  
  
    Intent intentDatos = new Intent( packageContext: this, DatosActividad.class);  
    intentDatos.putExtra( name: "numero", value: 10);  
    intentDatos.putExtra( name: "nombre", value: "Santiago");  
    startActivity(intentDatos);  
}  
  
public void iniciarActividad2(View view) {  
    Intent intentDatos = new Intent( packageContext: this, DatosActividad2.class);  
    startActivity(intentDatos);  
}
```

Intent Implícitos

- Desde cada uno de los otros dos Activities
- En este caso, elegimos como eventos de los botones las clases anónimas.

- Para recuperar nuestros parámetros, llamaremos al método getIntent() y dependiendo del tipo de parámetro
 - getStringExtra("clave");
 - getIntExtra("clave", default);

Existe otra forma de pasar parámetros que son con los objetos de tipo **Bundle**. Lo veremos más adelante.

```
/**
 * @author santi
 */
public class DatosActivity extends AppCompatActivity {

    private Button botonVolver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_datos);

        String nombre = getIntent().getStringExtra( name: "nombre");
        int numero = getIntent().getIntExtra( name: "numero", defaultValue: 0);
        Toast.makeText( context: this, text: "mi nombre es "+nombre + " y mi numero es " + numero, Toast.LENGTH_SHORT).show();

        botonVolver = (Button)findViewById(R.id.botonVolver);
        View.OnClickListener eventoBotonVolver = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Toast.makeText(DatosActivity.this, "Estoy dentro de Datos Activity ", Toast.LENGTH_SHORT).show();
                Intent intentVolver = new Intent( packageContext: DatosActivity.this, MainActivity.class);
                startActivity(intentVolver);
            }
        };
        botonVolver.setOnClickListener(eventoBotonVolver);
    }
}
```

FIN TEMA
