Programación multimedia y dispositivos móviles

UT-2. Conceptos de Diseño

https://developer.android.com/guide/

Introducción a las actividades

La clase Activity es un componente clave de una app para Android, y la forma en que se inician y se crean las actividades es una parte fundamental del modelo de aplicación de la plataforma. A diferencia de los paradigmas de programación en los que las apps se inician con un método main(), el sistema Android inicia el código en una instancia de Activity invocando métodos de devolución de llamada específicos que corresponden a etapas específicas de su ciclo de vida.

Una actividad proporciona la ventana en la que la app dibuja su IU. Por lo general, esta ventana llena la pantalla, pero puede ser más pequeña y flotar sobre otras ventanas. Generalmente, una actividad implementa una pantalla en una app. Por ejemplo, una actividad de una app puede implementar una pantalla Preferencias mientras otra implementa una pantalla Seleccionar foto.

Introducción a las actividades

La clase Activity está diseñada para facilitar este paradigma. Cuando una app invoca a otra, la app que realiza la llamada invoca una actividad en la otra, en lugar de a la app en sí. De esta manera, la actividad sirve como el punto de entrada para la interacción de una app con el usuario. Implementas una actividad como una subclase de la clase Activity.

La mayoría de las apps contienen varias pantallas, lo cual significa que incluyen varias actividades. Por lo general, una actividad en una app se especifica como la actividad principal, que es la primera pantalla que aparece cuando el usuario inicia la app. Luego, cada actividad puede iniciar otra actividad a fin de realizar diferentes acciones.

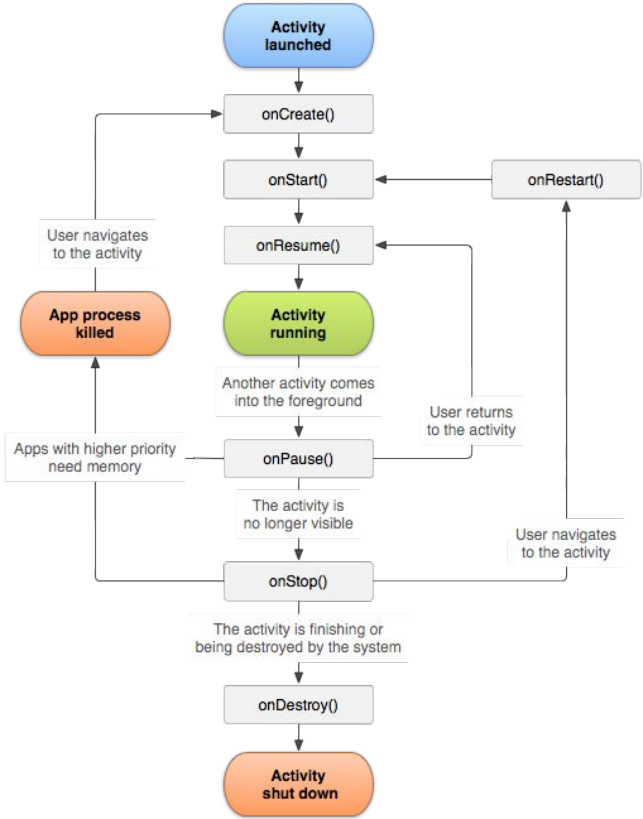
Ciclo de vida de una actividad

Una característica poco común y fundamental de Android es que la duración del proceso de una aplicación no se controla directamente desde la aplicación. En su lugar, el sistema la determina mediante una combinación de las partes de la aplicación que el sistema sabe que se están ejecutando, la importancia de estas para el usuario y la cantidad total de memoria disponible en el sistema.

Cuando un usuario navega por tu app, sale de ella y vuelve a entrar, las instancias de Activity de tu app pasan por diferentes estados de su ciclo de vida. La clase Activity proporciona una serie de devoluciones de llamada que permiten a la actividad saber que cambió un estado, es decir, que el sistema está creando, deteniendo o reanudando una actividad, o finalizando el proceso en el que se encuentra.

Ciclo de vida de una actividad

Dentro de los métodos de devolución de llamada de ciclo de vida, puedes declarar el comportamiento que tendrá tu actividad cuando el usuario la abandone y la reanude. Por ejemplo, si creas un reproductor de video en streaming, puedes pausar el video y cancelar la conexión de red cuando el usuario cambia a otra app. Cuando el usuario vuelve, puedes volver a establecer la conexión con la red y permitir que el usuario reanude el video desde el mismo punto.



Ciclo de vida de una actividad

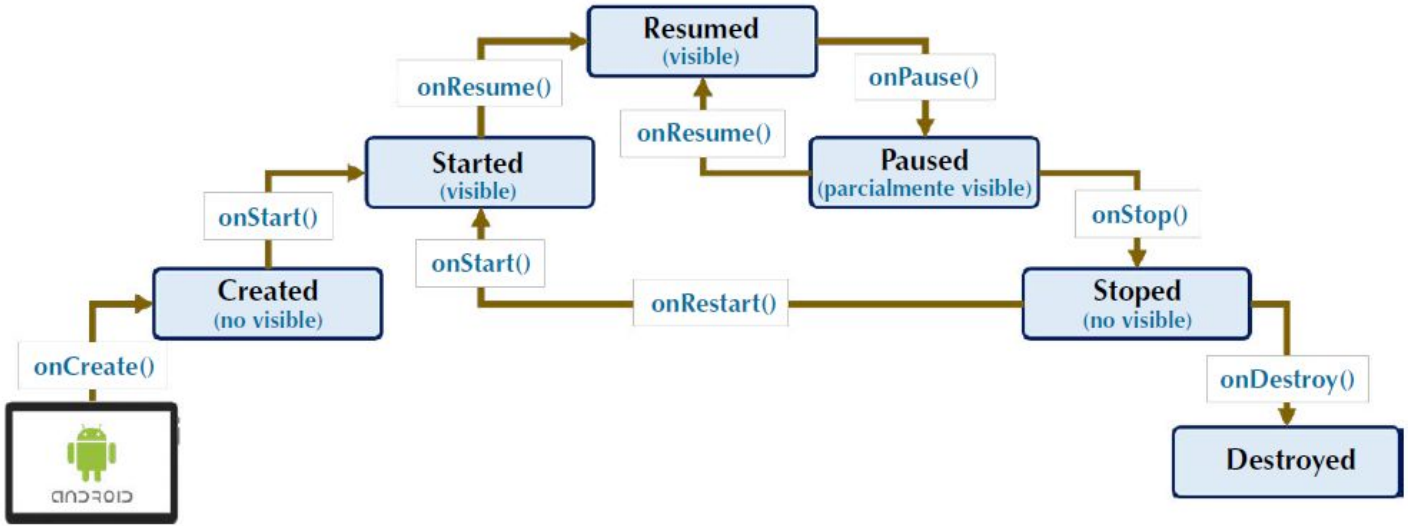


Figura 3.3
Ciclo de vida de una actividad en Android.

Componentes de una aplicación Android

Repasemos brevemente, a modo de glosario, los componentes principales que pueden formar parte de una aplicación Android o tener relevancia en su funcionamiento:

- **Activity:** Las actividades (activity) representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana o pantalla en cualquier otro lenguaje visual.
- **View:** Las vistas (view) son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, análogo por ejemplo a los controles de Java o .NET. De inicio, Android pone a nuestra disposición una gran cantidad de controles básicos, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear nuestros propios controles personalizados.

Componentes de una aplicación Android

- **Fragment:** Los fragmentos (fragment) se pueden entender como secciones o partes (habitualmente reutilizables) de la interfaz de usuario de una aplicación. De esta forma, una actividad podría contener varios fragmentos para formar la interfaz completa de la aplicación, y adicionalmente estos fragmentos se podrían reutilizar en distintas actividades o partes de la aplicación. No es obligatorio utilizar fragmentos en una aplicación, pero sí nos serán de mucha ayuda en ciertas ocasiones, por ejemplo para adaptar la interfaz de nuestra aplicación a distintos dispositivos, tamaños de pantalla, orientación, etc.
- **Service:** Los servicios (service) son componentes sin interfaz gráfica que se ejecutan en segundo plano. Conceptualmente, son similares a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acción, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (p.ej. actividades) si se necesita en algún momento la interacción con del usuario.

Componentes de una aplicación Android

- **Content Provider:** Un proveedor de contenidos (content provider) es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los content provider que ésta última haya definido.
- **Broadcast Receiver:** Un broadcast receiver es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: “Batería baja”, “SMS recibido”, “Tarjeta SD insertada”, …) o por otras aplicaciones (cualquier aplicación puede generar mensajes (intents, en terminología Android) de tipo broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).

Componentes de una aplicación Android

- **Widget:** Los widgets son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (home screen) del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal.
- **Intent:** Un intent es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

Información general de la Interfaz de usuario

La interfaz de usuario de tu app es todo aquello que el usuario puede ver y todo aquello con lo que este puede interactuar. Android ofrece una variedad de componentes de IU previamente compilados, como objetos de diseño estructurados y controles de IU que te permiten compilar la interfaz gráfica de usuario para tu app. Android también ofrece otros módulos de IU para interfaces especiales, como diálogos, notificaciones y menús.

Según Android Developers, todos los elementos de la interfaz de usuario de una app para Android están desarrollados con objetos **View** y **ViewGroup**. Un objeto View dibuja algo en la pantalla con lo que el usuario puede interactuar. Un objeto ViewGroup agrupa otros objetos View (y ViewGroup) para definir el diseño de la interfaz.

Android proporciona una colección de subclases View y ViewGroup que te ofrecen controles de entrada comunes (como los botones y los campos de texto) y varios modelos de diseño (como un diseño lineal o relativo).

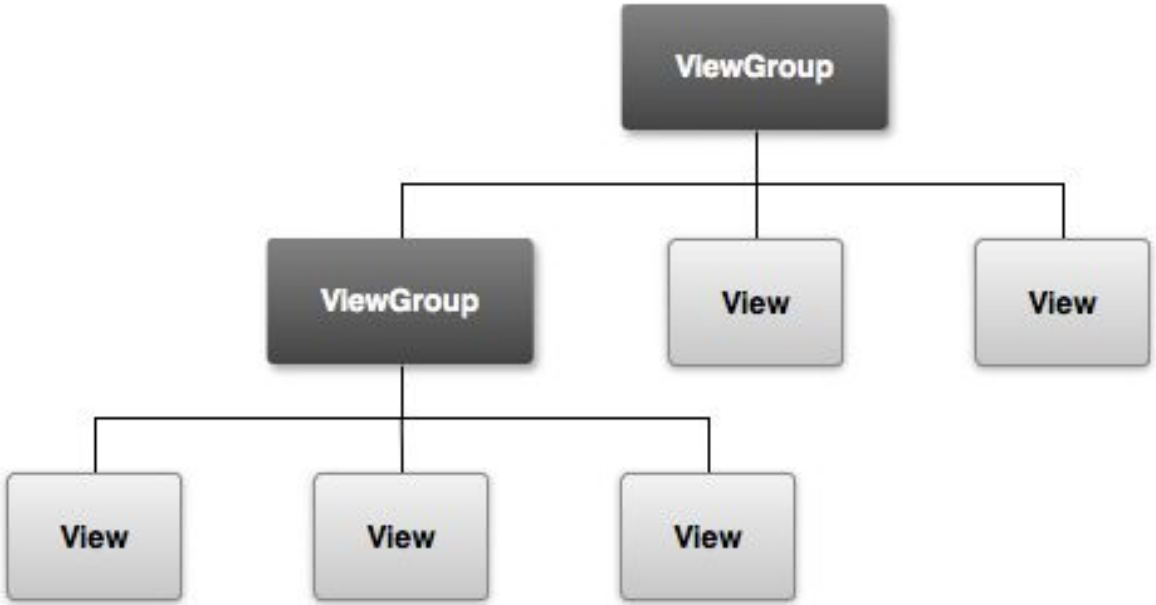
Diseño de la interfaz de usuario

El diseño (layout) define la estructura de la interfaz de usuario de tu app. Todos los elementos en el diseño (layout) se construyen usando una jerarquía de objetos View y ViewGroup.

Una vista (objeto View) normalmente dibuja algo que el usuario puede ver y con lo que puede interactuar, mientras que un **grupo de vista (objeto ViewGroup**) es un contenedor invisible que define la estructura de diseño de otros objetos View y ViewGroup. Este árbol de jerarquía puede ser tan simple o complejo como lo necesites (pero la simplicidad es lo mejor para el rendimiento).

Los objetos View se denominan controles (**widgets**) y podemos crear instancias de muchas subclases como por ejemplo **Button** o **TextView**. Los objetos **ViewGroup** se denominan estructuras de diseño (**layouts**) y pueden ser de distintos tipos los cuales proporcionan diferentes estructuras de diseño como **LinearLayout** o **ConstraintLayout**.

Diseño de la interfaz de usuario



Otros componentes de la interfaz de usuario

Además de los objetos View y ViewGroup. Android proporciona varios componentes de app que ofrecen un diseño de IU estándar, en los cuales solo tienes que definir el contenido.

Cada uno de estos componentes de IU tienen un conjunto único de clases en la API y se describen en los apartados de la documentación correspondientes.

Diseños (layouts)

Un diseño (layout) define la estructura visual para una interfaz de usuario, como la IU para una actividad o control (widget) de una app. Puedes definir un diseño (layout) de dos maneras:

- Definir elementos de la IU en XML. Android proporciona un vocabulario XML simple que coincide con las clases y subclases de vistas, como las que se usan para controles (widgets) y diseños.
- Dibujando los elementos de la IU con la “paleta”. Android proporciona una herramienta con la dibujar los controles y ver más o menos cómo van a quedar y que puedes cambiar en cualquier momento a ver el XML y viceversa..

También podrás instanciar elementos de diseño en tiempo de ejecución. Tu aplicación puede crear objetos View y ViewGroup y manipular sus propiedades programáticamente.

Diseños (layouts)

La ventaja de definir tu IU en XML es que te permite separar mejor la presentación de tu aplicación del código que controla su comportamiento. Tus descripciones de la IU son externas al código de tu aplicación, lo que significa que puedes modificarlo o adaptarlo sin tener que modificar el código fuente y volver a compilar. Por ejemplo, puedes crear diseños XML para diferente orientación de pantalla, diferentes tamaños de pantalla de dispositivos y diferentes idiomas.

En general, el vocabulario XML para definir elementos de la IU sigue de cerca la estructura y la denominación de las clases y los métodos, en los que los nombres de los elementos coinciden con los nombres de las clases y los nombres de los atributos coinciden con los métodos. De hecho, la correspondencia generalmente es tan directa que puedes deducir qué atributo XML corresponde a un método de clase, o deducir qué clase corresponde a un elemento XML determinado. No obstante, ten en cuenta que no todo el vocabulario es idéntico. En algunos casos, hay pequeñas diferencias de denominación.

Atributos

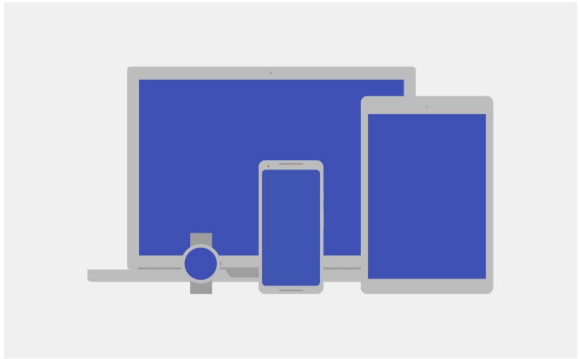
Todos los objetos **View** y **ViewGroup** admiten su propia variedad de atributos XML. Algunos atributos son específicos para un objeto View (por ejemplo, **TextView** admite el atributo **textSize**), pero a esos atributos también los heredan otros objetos View que podrían extender esta clase. Algunos son comunes para todos los objetos View ya que se heredan desde la clase View raíz (como el atributo **id**).

Todos los grupos de vistas incluyen un ancho y una altura (layout_width y layout_height). Puedes especificar el ancho y la altura con medidas exactas, aunque probablemente no quieras hacerlo con mucha frecuencia.

Compatibilidad de pantalla. Tamaños y densidades

Android se ejecuta en muchos tipos de dispositivos los cuales tienen diferentes tamaños de pantalla y densidades de píxeles.

El sistema realiza un escalado y redimensionamiento básico para adaptar la interfaz de usuario a las distintas pantallas, pero hay un trabajo adicional que debemos hacer para asegurar que nuestra interfaz de usuario se adapta correctamente a cada tipo de pantalla.



Compatibilidad de pantalla. Tamaños y densidades

Por defecto, Android redimensiona el diseño de nuestra app para que se ajuste a la pantalla actual. Para asegurar que nuestro diseño se redimensiona correctamente incluso para pequeñas variaciones en el tamaño de pantalla, necesitamos implementar nuestro diseño con la flexibilidad en mente.

El principio fundamental que debemos seguir es evitar codificar de forma fija (hard-coding) la posición y el tamaño de nuestros componentes de la interfaz de usuario. En vez de eso, debemos permitir que las vistas puedan cambiar de tamaño y especificar posiciones de visualización relativas a la vista padre (contenedor)

Compatibilidad de pantalla. Tamaños y densidades

Para asegurar un diseño flexible deberíamos usar los valores:

- **wrap_content**: indica a la vista que adapte su tamaño a su contenido
- **match_parent**: hace que la vista se expanda tanto como sea posible dentro de la vista padre (contenedor)

En general, no se recomienda especificar el ancho y la altura de un diseño con unidades absolutas como píxeles.

En cambio, el uso de medidas relativas como unidades de píxeles independientes de la densidad (dp), wrap_content, o match_parent, es un mejor enfoque, ya que ayuda a garantizar que tu aplicación se muestre correctamente en dispositivos con pantallas de diferentes tamaños.

Creación de diseños alternativos

Aunque nuestro diseño debería siempre responder a los cambios de tamaños de pantalla ajustado el espacio dentro y alrededor de sus vistas, eso podría no proporcionar la mejor experiencia de usuario para cada tamaño de pantalla.

Por ejemplo, la interfaz de usuario que diseñamos para un teléfono móvil, probablemente no ofrece la mejor experiencia para una tableta. Por tanto, nuestra app debería proporcionar también recursos de diseño alternativos (alternative layout resources) para optimizar el diseño de la interfaz de usuario para distintos tamaños de pantalla.

Creación de diseños alternativos

De forma “nativa” o “natural”, Android Studio te permite generar los diseños alternativos más comunes:

Para crear simplemente debemos ir a la paleta de cuya actividad queramos crear un diseño alternativo y pulsar el botón:

Desde donde podremos elegir crear un layout para orientación apaisada o para tablet. Si lo hacemos vemos que se ha creado un nuevo directorio llamado layout-land que contiene un nuevo fichero de diseño activity_main.xml para modo orientación apaisada (landscape).

Creación de diseños alternativos

Normalmente, desearemos que la orientación (y su diseño asociado) obedezca al sensor de orientación, de tal forma, que al girar el móvil automáticamente se visualice el diseño vertical (portrait) o el apaisado (landscape). Es la opción por defecto en cada pantalla sin hacer nada especial

Pero también podemos hacer que la rotación de nuestro dispositivo no afecte a la apariencia de nuestro diseño, es decir, que se mantenga fija. Para ello podemos hacer cambios en nuestro AndroidManifest.xml. Para ello usaremos el atributo android:screenOrientation sobre la actividad (activity) en cuestión:

- android:screenOrientation="portrait" para el modo vertical.
- android:screenOrientation="landscape" para el modo apaisado.

Creación de diseños alternativos

Podemos hacer lo anterior directamente en el fichero kotlin (dentro del método onCreate():

- this.requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT;
- this.requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE;
- this.requestedOrientation =ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR;

https://developer.android.com/guide/topics/manifest/activity-element?hl=es-419#screen