Desarrollo de aplicaciones móviles I

Tema Nº8:Componentes de Android (Actividades II).

Indicador de logro Nº8:Estructura una interfaz de usuario en Android mediante la integración de componentes que funcionan como contenedores de otros o que facilitan la navegación dentro de la aplicación.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº8:**

Componentes de Android (Actividades).

**Subtema 8.1:**

Toolbar

Desde la versión 3.0, se introdujo en Android un nuevo elemento en la interfaz de usuario: la barra de acciones. Esta se sitúa en la parte superior de la pantalla, fue creada para que el usuario tuviera una experiencia unificada a través de las distintas aplicaciones. La barra de acciones aglutina varios elementos; los más habituales son el nombre de la aplicación, el botón para abrir el Navigation Drawer y los botones de acciones frecuentes. Las acciones menos utilizadas se sitúan en un menú desplegable, que se abrirá desde el botón Overflow (se representa con tres puntos verticales). Si la aplicación dispone de pestañas (tabs), estas podrán situarse en la barra de acciones. También pueden añadirse otros elementos, como listas desplegables y otros tipos de widgets incrustados, como el widget de búsqueda que veremos más adelante.

Interfaz de usuario gráfica

Descripción generada automáticamente

Existen dos clases que nos permiten añadir la barra de acciones: ActionBar y ToolBar. La clase ActionBar aparece en la versión 3.0. Por defecto, la barra de acciones es incluida en todas las actividades. Si queremos que no aparezca tenemos que asignar un tema especial a la actividad. Por ejemplo. Theme.AppCompat.NoActionBar o cualquiera que acabe en .NoActionBar. La clase ToolBar aparece con la versión 5.0. Cambia el diseño de la barra de acciones para que siga las especificaciones de Material Design. Puede usarse en versiones anteriores dado que no se incorpora al API de la versión 5.0, si no a una la librería de compatibilidad appcompat. A diferencia de ActionBar, la barra de acciones no es incrustada de forma automática, si no que hay que incluirla en el layout con la etiqueta <ToolBar>. Esto nos permite situarla en la posición que queramos y nos da más opciones de configuración.

Añadir un ToolBar a la aplicación es muy sencillo. Normalmente no es necesario realizarlo si al crear un proyecto has seleccionado Basic Activity o Scrolling Activity, dado que en este caso ya se ha añadido un Toolbar. Se ha incluido el siguiente ejercicio para los casos en que partes de un proyecto donde se ha incluido un ActionBar (seleccionando Empty Activity) o en proyectos creados con versiones anteriores.

1. Quitamos el ActionBar.

Texto

Descripción generada automáticamente con confianza baja

1. Agregamos el ToolBar al Activity.

Captura de pantalla de un celular

Descripción generada automáticamente

1. Creamos un nuevo recurso Menu en el directorio res.

Texto

Descripción generada automáticamente

1. En la clase java del Activity agregamos las siguientes instrucciones.

Imagen que contiene nombre de la empresa

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Rectángulo

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

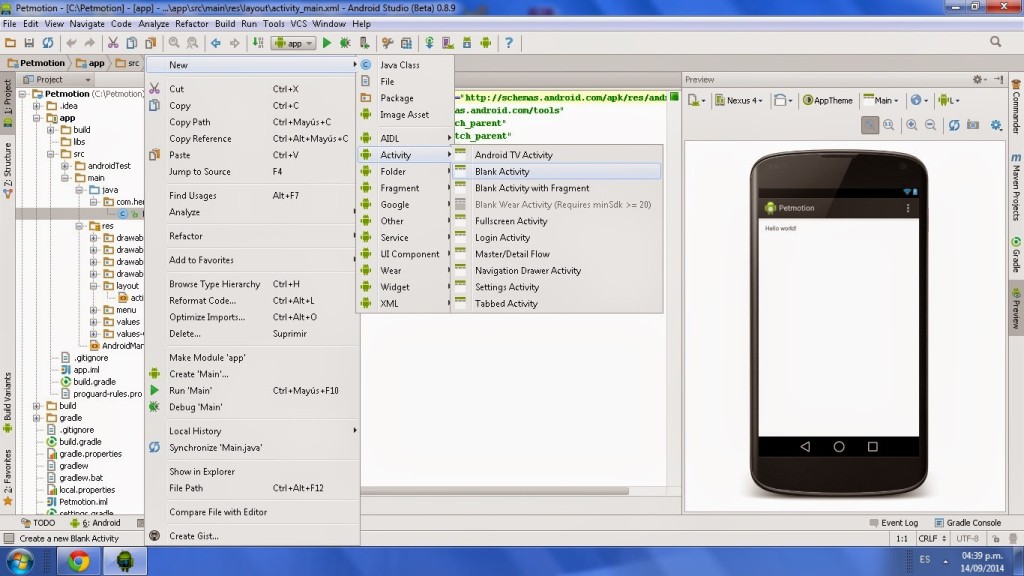
**Subtema 8.2:**

Intents

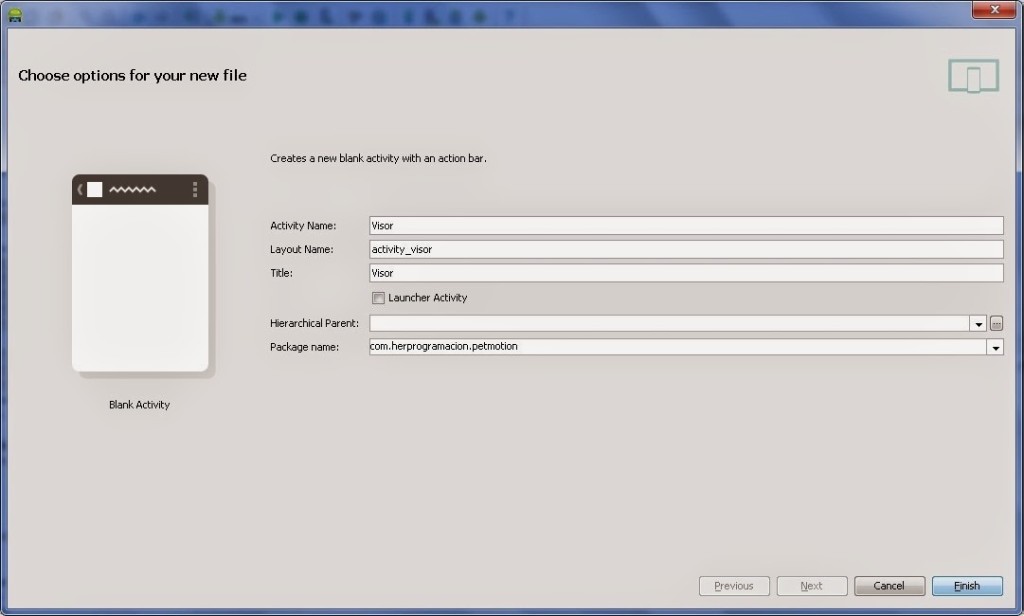
**Ejemplos:**

**Crear una nueva actividad**

A continuación, crearemos una nueva actividad en blanco. Para ello ve a la raíz de tus archivos java y presiona “**New**“. En ese nuevo menú presiona “**Activity**” y en seguida “**Blank Activity**“:



A continuación, veremos una asisten que nos proporcionará un formulario para personalizar nuestra nueva actividad:



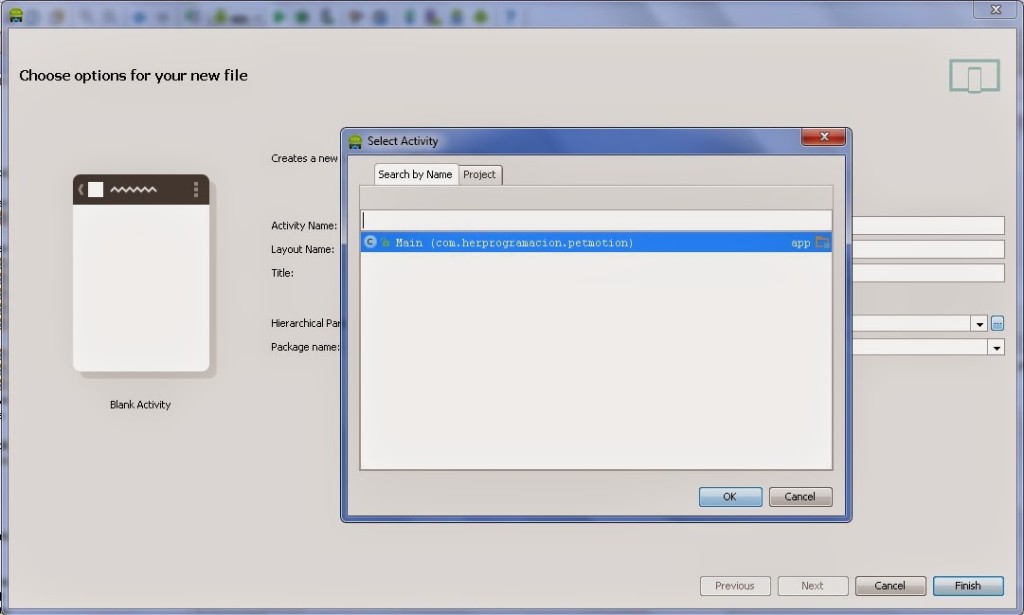
Veremos que el primer parámetro es “**Activity name**“. Este representa el nombre que le pondremos a la actividad, el cual será “**Visor**“.

Luego sigue “**Layout Name**” que hace referencia al nombre del layout de la actividad. Dejaremos su valor actual recomendado por **Android Studio**.

El tercer parámetro es “**Title**“, es decir, el título que se mostrará al ejecutar al actividad en la parte superior. Dejaremos el valor recomendado.

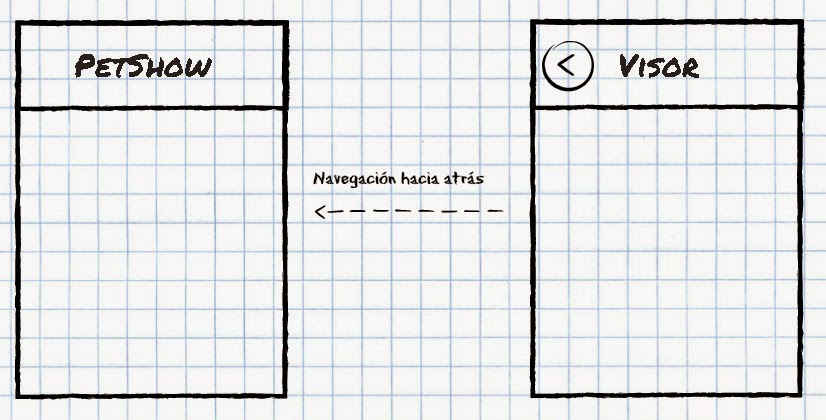
La opción “Launcher Activity” la desmarcaremos, ya que la [actividad principal](http://hermosaprogramacion.blogspot.com/2014/08/android-app-componentes.html#actividad-principal) es Main.java.

La quinta propiedad se llama “Hierarchical Parent“. Esta característica indica si nuestra nueva actividad es hija de otra actividad. Esto significa que están relacionadas jerárquicamente en la navegación de la interfaz. Para seleccionar la actividad padre pulsaremos “…” y seleccionaremos a Main.java.

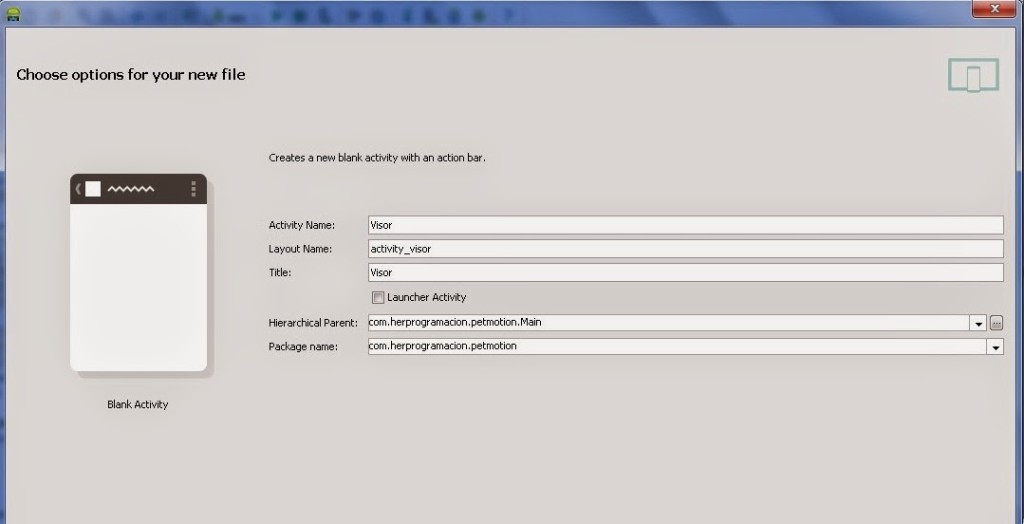


Si habilitamos esta opción podremos hacer uso del Up Button. Este botón es autogenerado por las aplicaciones Android cuando deseamos regresar de una actividad a otra. Normalmente se encuentra en la parte superior izquierda de la cabecera.

Veamos una ilustración:



Y el ultimo parámetro es el paquete donde ubicaremos la actividad, por lo que dejaremos el paquete por defecto. En mi caso es “***com.herprogramacion.petmotion***“. Luego presionamos “**Finish**” y nuestra nueva clase será creada.



Si te fijas en el archivo **Android Manifest** verás la nueva actividad agregada:

Texto

Descripción generada automáticamente

Como ves, la actividad Visor tiene un nuevo atributo llamado **android:parentActivityName** , el cual contiene el nombre de la actividad padre. Que para nosotros será “.Main”.

Esta relación se concluye añadiendo un elemento <meta-data> a la actividad y especificar el nombre de la acción PARENT\_ACTIVITY y el paquete donde se encuentra la clase “**com.herprogramacion.petmotion.Main**”.

**Diseñando las actividades**

Nuestro siguiente paso es diseñar los **layouts** de ambas actividades. Para la actividad principal agregaremos un botón con el identificador show\_pet\_button y el texto “Mostrar mascota”.

Su layout **activity\_main.xml** obtendría el siguiente aspecto:

Texto, Carta

Descripción generada automáticamente

No hay nada de extraño en el código, ya antes hemos [**agregado un Button**](http://hermosaprogramacion.blogspot.com/2014/09/android-layouts-views.html#agregar-button).

A continuación, pasamos a agregar un ImageView en Visor.

**Subtema 8.3:**

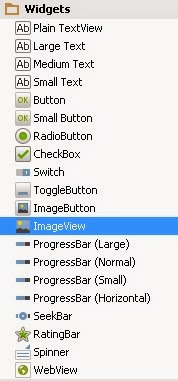
ImageView

**Ejemplos:**

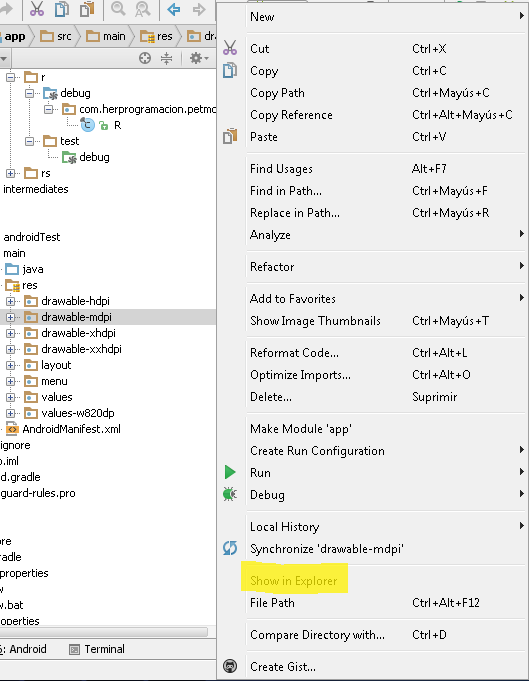
Un ImageView es un componente que permite visualizar las imágenes que tenemos dentro de nuestros recursos, como iconos, fotografías y demás. Permite controlar el **tamaño de la imagen**, la **escala**, **filtros**, **transparencia**, etc.

El objetivo de usar este componente se debe a que deseamos añadir una foto de una mascota en la actividad Visor. La idea es que el usuario presione el botón show\_pet\_button de Main y sea dirigido a Visor para visualizar la imagen.

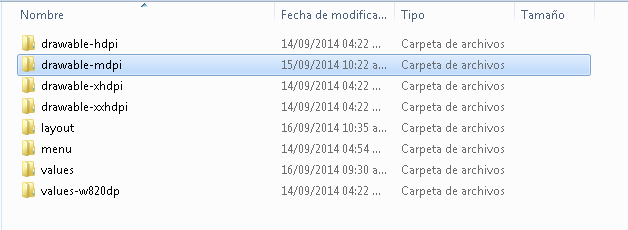
Para añadir un ImageView primero nos dirigiremos a la paleta y lo arrastraremos al centro del RelativeLayout



Ahora ubícate en la carpeta “**drawable-mdpi**” y presiona click derecho. Elige la opción “**Show in Explorer**”:



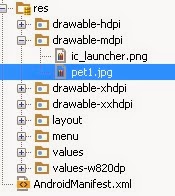
Una vez hecho esto, aparecerá la carpeta de los recursos (res) del proyecto.



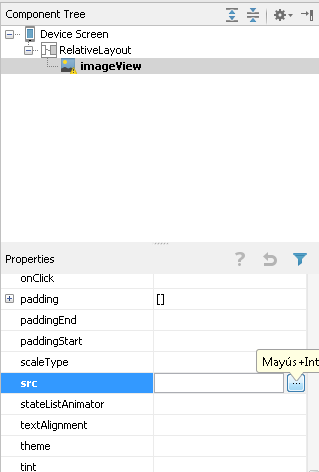
Si abres “**drawable-mdpi**” encontrarás el archivo “**ic\_launcher.png**” que representa al icono de la aplicación. Junto a él pega la siguiente [imagen de nuestra mascota](https://drive.google.com/file/d/0B2gGL1jnLPWVdi1vc2ZmQ2kwWkE/edit?usp=sharing).

(<https://drive.google.com/file/d/0B2gGL1jnLPWVdi1vc2ZmQ2kwWkE/edit>)

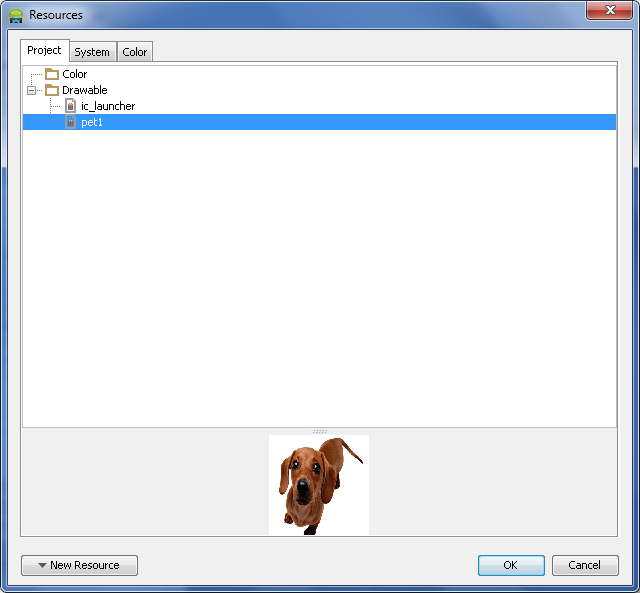
Automáticamente **Android Studio** refrescará nuestro proyecto y mostrará el archivo pet1.jpg en la carpeta de recursos:



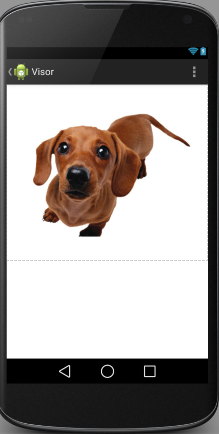
Una vez ubicada la imagen, nos situaremos en su atributo src en el **panel de propiedades**. src representa la dirección del recurso(**source**) o imagen que usaremos para visualizar en el ImageView. Por lo que presionaremos el botón “**…**“.



En seguida se mostrará un asistente para pickear la imagen que necesitamos ubicada en el proyecto. Rápidamente escogeremos a nuestro ***perrito lindo*** y presionaremos “**OK**“.



Veamos cómo quedó el diseño:



*¡Hermoso!, ¿cierto?*

El código XML del layout de la actividad Visor tendría la siguiente estructura:

Texto, Carta

Descripción generada automáticamente

**Subtema 8.4:**

Analizando nuestras vistas

**Ejemplos:**

**Iniciar una actividad**

El paso final es iniciar nuestra actividad Visor una vez sea presionado el botón show\_pet\_button. Para ello simplemente usaremos el método startActivity() de la clase Activity. Este método recibe como parámetro de entrada un Intent. Este intent será el encargado de portar el mensaje para ejecutar la acción, que en este caso es iniciar una actividad.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

El constructor de intent recibe dos parámetros. El primero es el contexto desde deseamos enviar el mensaje, en este caso es nuestra propia actividad. El segundo hace referencia a la clase del componente receptor del mensaje de inicio, será nuestra clase Visor.

El código final de la actividad Main sería el siguiente:



**Enviar Datos Hacia Una Actividad**

Los Intents permiten que enviemos datos al enlazar dos actividades en nuestra aplicación. A estos datos se les denominan ***Extras*** y se componen de un **identificador** y un **valor.**

Antes de iniciar la actividad debemos adherir los datos al intent con el método putExtra(). Este método es polimórfico y es capaz de enviar varios tipos de datos.

Para mostrar su uso vamos a suponer que por alguna razón que necesitamos enviar una cadena desde la actividad principal hasta el visor. Esta cadena será el nombre de la imagen que se va a mostrar, que es este caso es “pet1.jpg”. Para ello agregaremos un TextView que mostrará la cadena en su atributo text.

Ahora dentro de la nuestra actividad Main declararemos una constante de tipo String que represente el identificador único del dato Extra para nuestro Intent. Esto nos permitirá diferenciar de otros pares:



Es importante encontrar una cadena que sea única para que no interfiera con otros datos, por eso elegimos el nombre del paquete de nuestro proyecto para asegurarnos que no se repita. Ahora solo queda implementar el método putExtra() antes de iniciar la actividad:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**¿Y cómo recibo ese Extra desde otra actividad?**

¡Es súper sencillo!, solo invocamos el método getIntent() de la clase Activity y obtendremos el intent que inicio la actividad. Todas las actividades son iniciadas por un intent, incluso podemos obtener el Intent de la actividad principal iniciado por el sistema.

Desde el método onCreate() de Visor obtendremos el intent. Luego extraeremos la cadena del mensaje que enviamos con getStringExtra() y se la asignaremos al atributo text del TextView:



Al ejecutar obtendremos el siguiente resultado:



**Iniciar Una Aplicación Externa Desde Nuestra Actividad**

Un intent también nos permite iniciar una actividad externa para apalancarse y cumplir algún cometido que necesita nuestra aplicación. Como por ejemplo **enviar un correo** con los datos que se encuentra en nuestra actividad, darle **like**(*facebook app*) a una publicación relacionada con un link en nuestra aplicación, ver alguna dirección en [*Google Maps*](http://www.hermosaprogramacion.com/2016/05/google-maps-android-api-v2/) *App, etc*.

Todo esto suena muy emocionante, pero antes de proseguir aclararemos que tipos de comportamiento puede llegar a tomar un **Intent** al iniciar otro componente.

**Intents Explícitos**

Cuando un intent se crea para enviar un mensaje a un componente específico se le llama **Intent explícito**. Por ejemplo, cuando iniciamos otra actividad en el ejemplo anterior, sabíamos previamente que la actividad era Visor.java y no ninguna otra. Lo que significa que se hizo explicita la asociación del Intent a esta acción.

**Intents Implicitos**

Un intent se comporta **implícitamente** cuando no se sabe con exactitud que componente se iniciará para recibir el mensaje que vamos a enviar. Aunque no se sabe específicamente, si se tiene claro cuál debe ser el propósito del componente que sea iniciado.

Un ejemplo que te aclarará esta situación: Cuando estás en el navegador de tu dispositivo móvil y vez un artículo que te ha gustado mucho y deseas compartirlo.

Para esta acción no se ejecuta una aplicación inmediatamente, si no que el sistema provee una lista de todas aquellas aplicaciones sociales que pudiesen ayudarte a culminar la acción de **compartir**.

Seguramente en la lista estarían como opciones **Facebook**, **Google Plus**, **Twitter**, **Gmail**, **Outlook**, etc. Donde tú escogerás la más pertinente. Así que el **intent implícito** solo transmitió un llamado al sistema, quién se encargó de alertar a las aplicaciones capaces de responder a la acción de compartir (**Share**).

**Iniciar un navegador web Android desde nuestra actividad**

A continuación, vamos a ejecutar una aplicación de navegación web desde nuestra actividad principal. Añadiremos un TextView en la parte inferior del RelativeLayout con el mensaje “**Autores**” y la enlazaremos al [**Home** de **Hermosa Programación**](http://www.hermosaprogramacion.com/).

Para este cometido usaremos un intent implícito. Fíjate como lo haremos



*¿Se ve sencillo de entender, no?*

Y lo es… en estas líneas se encuentra uno de los poderes más maravillosos del **desarrollo Android**.

En la primera línea vemos el uso de la clase Uri del paquete import android.net.Uri. Esta nueva instancia es clave para especificar el sitio web que deseamos visualizar. Si recuerdas el [**concepto de URI**](http://hermosaprogramacion.blogspot.com/2014/08/xml-lenguaje.html#concepto-uri) sabrás que debemos incluir un identificador con un formato esquematizado.

Obtendremos un nuevo objeto Uri a través de la **URL** de nuestro home (una URL es una URI, recuérdalo).

El método parse() solo traduce el formato, interpreta el esquema y produce los datos necesarios para el Intent.

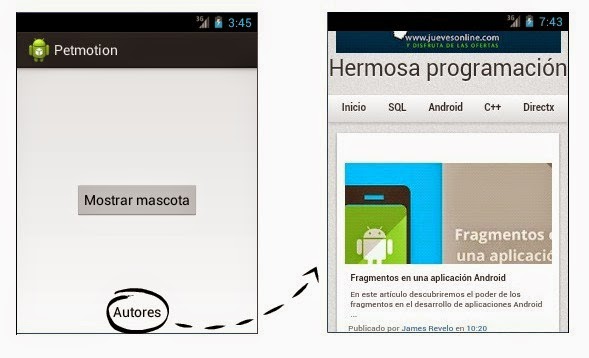
Luego se crea una nueva instancia de nuestro intent llamado webIntent y esta vez inicializamos en línea con un constructor nuevo. Su primer parámetro es una constate que indica la acción genérica que deseamos realizar. El segundo parámetro es la **URI** que representa el mensaje a interpretar.

En el primer parámetro usamos la constante Intent.ACTION\_VIEW. Esta representa la **visualización de contenido**, en este caso una página web asociada a un esquema de datos URI.

Todas estas acciones difieren en su forma de ejecución. No todas requieren que usemos URIs para activarlas, hay otras que necesitan que usemos el método putExtra() para ingresar parámetros necesarios para el funcionamiento. En otros artículos podremos ver más sobre estas acciones y detallar su funcionamiento con ejemplos.

Retomando con nuestro tema, la última parte es iniciar la actividad de la aplicación externa con el método startActivity(). Solo se pasa como parámetro el intent recién creado y corremos nuestra aplicación.

Al ejecutar nuestra aplicación y hacer click en nuestro TextView se ejecutará el **Navegador de Android.**



Es posible que en el dispositivo que ejecutemos este intent no haya aplicaciones que muestren contenido web. Si es así, la aplicación presentaría un error en tiempo de ejecución. Para evitar esta situación podemos usar el siguiente código:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

No entraremos en detalle sobre esta implementación, pero a simple vista podemos notar que estamos llamando a PackageManager. Esta entidad puede proporcionarnos aquellos elementos que respondieron al intent con el método queryIntentActivities(). Luego comprobamos si al menos uno fue retornado, si fue así entonces iniciamos la actividad.

## 

**Recibir Datos De Una Actividad**

Bueno, era de esperar que en algún momento nos surgiera la necesidad de obtener de una actividad, si es que nuestra aplicación está diseñada con un flujo de datos entre actividades.

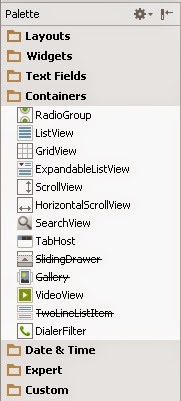
Para comprender la recepción de datos de otra actividad adaptaremos nuestra aplicación **Petmotion** para que permita seleccionar su opinión sobre la imagen de nuestro perrito en la actividad Visor. La idea es usar un [RadioGroup](http://www.hermosaprogramacion.com/2016/04/radiobutton-android/) para la selección de opciones y un botón para enviar. Si no sabes nada sobre este View, no te preocupes, la siguiente definición te ayudará.

**¿Qué es un RadioGroup?**

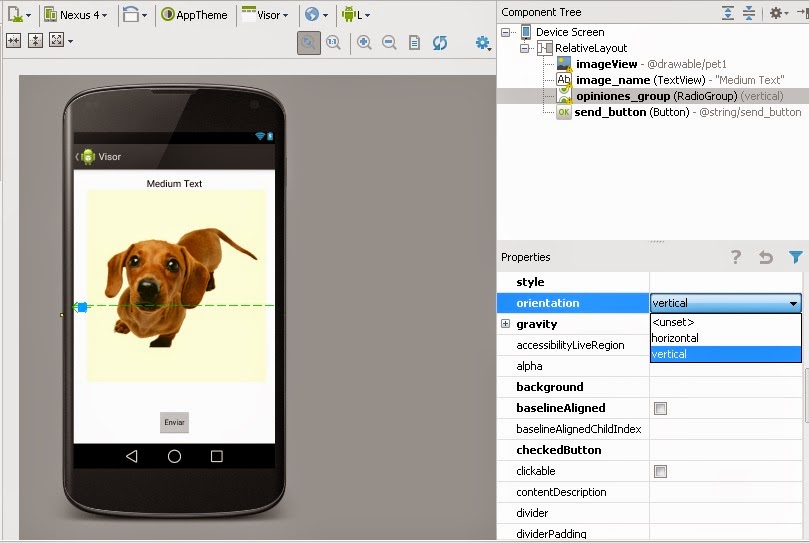
Un RadioGroup es un ViewGroup cuyo contenido está destinado a albergar RadioButtons. Un [RadioButton](http://www.hermosaprogramacion.com/2016/04/radiobutton-android/) es un botón circular que visualmente representa dos estados. Uno de selección y otro de inactividad. Dependiendo del estado la aplicación actuará.

Al ubicar un RadioButton dentro de un RadioGroup inmediatamente su estado obedecerá a una múltiple exclusión entre todos los elementos que habiten en el contenedor. Lo que quiere decir, que, si hay 3 radios, solo uno podrá estar en estado de selección, los otros 2 restantes deben mantener muertos.

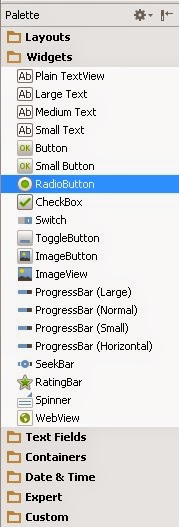
Para añadir un RadioGroup a nuestra actividad Visor iremos a la **Paleta del editor de diseño**, buscaremos la categoría **Containers** y luego arrastraremos uno:



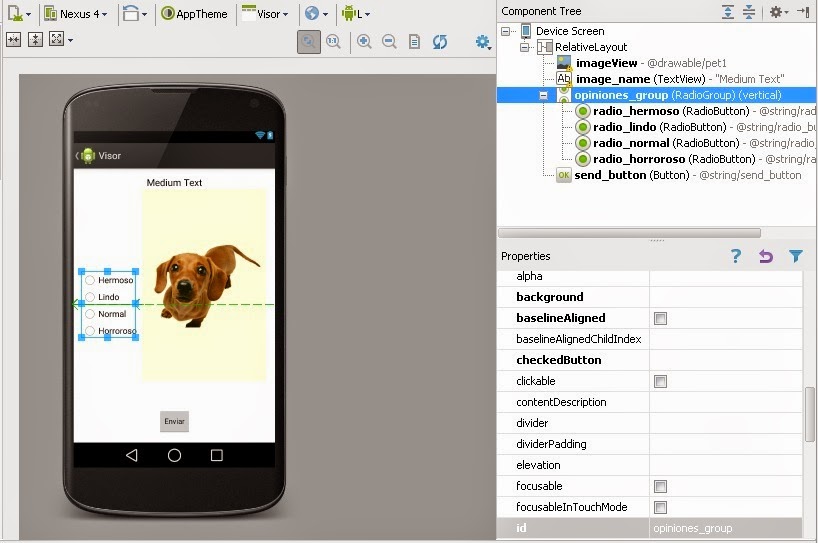
Al insertarlo en el Layout pondremos sus dimensiones en match\_parent, a su id le asignaremos “opiniones\_group” y le cambiaremos su atributo orientation a vertical. orientation controla la orientación que tendrá los radios cuando los insertemos como hijos.



Ahora insertaremos dentro del RadioGroup cuatro RadioButtons. Arrastra los radios desde la categoría **Widgets** hasta el interior del opiniones\_group:



Una vez terminada esta tarea deberíamos tener algo como esto:



Si observas el **Component Tree** verás nuestro RadioGroup como padre de 4 radios. En este caso usaremos cuatro distintos de belleza: “***Hermoso***“, “***Lindo***“, “***Normal***” y “***Horroroso***“. Cada texto podemos implementarlo en el atributo text de los radiobuttons.

En cuanto a la ubicación del grupo, puedes elegirla a tu preferencia, en mi caso ubique el borde izquierdo del ImageView alineado al borde derecho del grupo. También lo centre verticalmente y alineado a la izquierda del RelativeLayout.

Adicionalmente usaremos un botón en la parte inferior llamada “send\_button”. Al presionarlo enviaremos de regreso la opinión que elegimos en el radiogroup.

Mira el archivo de diseño:

Texto

Descripción generada automáticamente



Recuerda [añadir al archivo strings.xml los textos importantes](http://hermosaprogramacion.blogspot.com/2014/09/android-layouts-views.html#a%C3%B1adir-strings).

**Usando la función startActivityForResult()**

Una vez terminado nuestro diseño de la actividad Visor usaremos el método setActitivtyForResult() de la clase Activity para recibir datos de esta actividad.

Este método informa al contexto de la aplicación que deseamos obtener datos de la actividad que se va a llamar cuando sea terminada. Al igual que startActitivity(), recibe el intent portador del mensaje, pero adicionalmente usaremos otro parámetro para indicar un código identificador de la recepción.

Este código representa un canal único de envío y recepción por el cual comprobaremos si hubo éxito en la petición o si todo falló. Para definirlo crearemos una constante entera que indique un número positivo.

Veamos como cambió nuestro código:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Listo!, con esa modificación nuestra petición está en camino. Ahora veamos como devolver desde Visor la cadena del RadioButton que actualmente está seleccionado al presionar el botón “Enviar”:



1. La primera instrucción obtiene una instancia del RadioButton actualmente seleccionado. Para ello usamos como parámetro la instancia obtenida a través el RadioGroup(obtenida en onCreate() previamente) por el método getCheckedRadioButtonId(). Este método nos proporciona el id del radio seleccionado.
2. Luego obtenemos el valor del atributo text del radio con getText().
3. Es necesario enviar un mensaje de vuelta, por lo que creamos una instancia de un Intent vacío.
4. Añadiremos como ***Extra*** el texto del radio.
5. En esta instrucción usamos el método setResult() para enviar el resultado de forma exitosa. El primer parámetro es la constante RESULT\_OK de la clase Activity que indica que todo ha salido bien y el segundo parámetro es nuestro Intent portador del mensaje.
6. Finalizamos la actividad Visor con el método finish() de la clase Actitivy.

Ahora te preguntarás… *¿Cómo hago para recibir desde Main este dato?*…y la respuesta es la siguiente:

Sobrescribiremos un método especial de la clase Activity llamado onActivityResult(), el cual es autoinvocado para recibir la información de una actividad que fue llamada con startActivityForResult().

En este método comprobaremos que todo esté bien y luego obtendremos los datos del intent de respuesta.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Como ves, el método recibe tres parámetros. El primero de ellos es el **código de petición** que usamos como canal entre ambas actividades. El segundo es el **código de resultado** que expresa si la operación fue exitosa o fallida. Y el tercero es el intent de reenvío que declaramos en Visor.

Cuando sobrescribimos onActivityResult() además de enviar al constructor de la superclase los parámetros, debemos comprobar si este mensaje vino con el mismo código OPINION\_REQUEST\_CODE. Si es así, pasamos a comprobar si fue exitoso el trámite con RESULT\_OK. Si todo salió bien entonces procedemos a asignar a un TextView de prueba ubicado en Main, el texto que contiene el intent de reenvío.

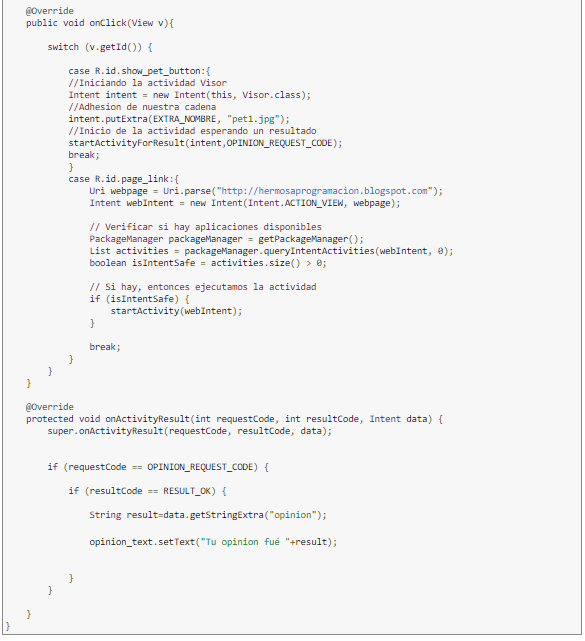
### Nuestro código final

Si todo salió bien y seguiste los pasos expuestos con anterioridad, tendremos una actividad Main con la siguiente lógica:

Texto

Descripción generada automáticamente

Continúa:

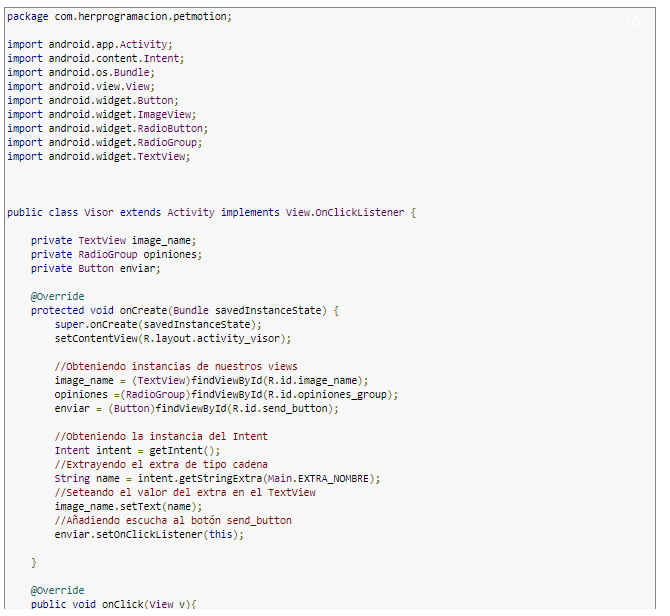


Nuestro archivo layout activity\_main.xml se vería asi:

Texto

Descripción generada automáticamente

Ahora veamos la lógica de Visor.java:



Continúa:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Y por último la estructura de **activity\_visor.xml**:

Texto

Descripción generada automáticamente

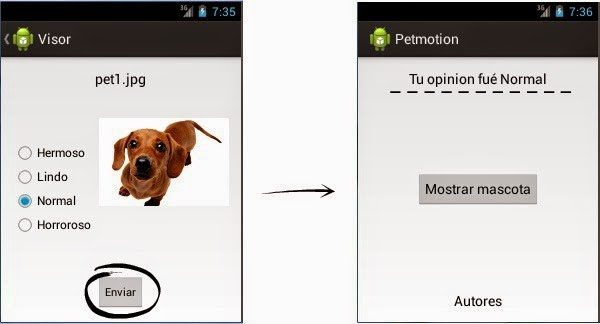
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Al correr la aplicación en Android Studio el resultado sería este:



**Actividad:**

1. CUESTIONARIO TÉCNICO

* ¿Cuándo utilizamos Intents en el contexto del Desarrollo de Aplicaciones Móviles en Android?
* ¿En qué archivo se pueden visualizar las nuevas activities agregadas?
* ¿Qué pasos debemos de configurar para la implementación del UpButton en nuestras aplicaciones?
* ¿Qué atributos permite controlar el widget Imageview?
* ¿Qué funcionalidad desencadena el método startActivity()?

1. CONCLUSIONES DE LA EXPERIENCIA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_