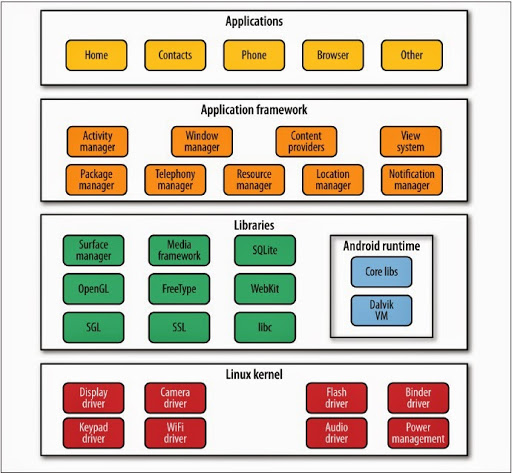
**2. CONOCIENDO ANDROID**

**ESTRUCTURA**

Android es un sistema operativo basado en el Kernel de Linux, a este se le suman otras capas en su estructura como la capa de librerías, la capa de frameworks y la de aplicaciones. El sistema, además cuenta con una máquina virtual, anteriormente llamada Dalvik; en la versión 5.0 de Android hizo el cambio definitivo a una nueva máquina virtual denominada ART.



1. Representación gráfica de la estructura de Android

**Linux Kernel**

Esta capa es el corazón de Android, ya que en él se realiza el manejo de memoria, procesos, drivers y energía. Estas son solo algunas de las tareas principales que se realizan en esta capa. Aquí es donde se logra la comunicación con el hardware del equipo. Podríamos decir que gracias a esta capa no tenemos que lidiar con la gran cantidad de hardware que podemos encontrarnos por cada dispositivo fabricado, solo nos tenemos que preocupar por la “cámara” del móvil y no por la cámara X del modelo Y del fabricante Z; para nosotros solo es la “cámara”.

**Libraties**

En esta capa hallamos librerías nativas de Android que se encuentran escritas en C o C++ para realizar tareas específicas.

* **Surface Manager:** Gestiona el acceso a la pantalla
* **Media Framework:** Reproduce imágenes, audio y vídeo
* **SQLite:** Bases de datos SQLite
* **WebKit:** Navegador nativo de Android
* **SGL:** Gráficos 2D
* **OpenGL:** Gráficos 3D
* **Freetype:** Renderizar vectores o imágenes

**Android Runtime**

Este componente hasta hace poco era la máquina virtual Dalvik, actualmente la máquina virtual es ART. La máquina virtual de Android no es la misma que la JVM o Java Virtual Machine, por eso el código escrito en java no es automáticamente funcional para Android, este tiene que ser escrito exclusivamente para Android, por las librerías y otras partes que no se tienen contempladas, además de que el ByteCode que se genera al compilar para java es inservible para la máquina virtual de Android.

**Application Framework**

Esta capa es visible para el desarrollador ya que los componentes que contiene forman para de muchos de los desarrollos que esteremos realizando.

* **Activity Manager:** Administra las actividades de nuestra aplicación y el ciclo de vida
* **Windows Manager:** Administra lo que se muestra en la pantalla
* **Content Provider:** Administrar, dependiendo de cómo le indiquemos, algunos contenidos.

Puede ser que necesitemos encapsular para enviar o compartir información.

* **View:** Las vistas de elementos que son parte de la interfaz gráfica, como los mapas, cuadros de texto, etc.
* **Notification Manager:** Administra las notificaciones
* **Package Manager:** Administra los paquetes y nos permite el uso de archivos en otros paquetes.
* **Telephony Manager:** Administra lo que tiene que ver con la telefonía, como por ejemplo las llamadas y los

mensajes.

* **Resource Manager:** Administra los recursos de la aplicación, como los XML, sonidos, imágenes.
* **Location Manager:** Gestiona la posición geográfica
* **Sensor Manager:** Gestiona los sensores que tenemos en el dispositivo
* **Multimedia:** Administra lo referente a vídeo, audio e imágenes

**Applications**

Podemos considerarla como la última capa si estamos dentro del sistema operativo, es decir, que es la capa en la que interactúa el usuario. Aquí es donde hallamos las aplicaciones como la de mensajes, el navegador y todas las demás que utiliza el usuario en el funcionamiento diario en su dispositivo móvil. Si el usuario instala una aplicación, esta es la capa en donde queda instalada.

**3. ENTORNO DE DESARROLLO**

El entorno de desarrollo Android es el conjunto de herramientas que vamos a necesitar implementar para programar una aplicación Android.

Antes de comenzar a preparar nuestro entorno de desarrollo, debemos dejar claro cuáles son los 4 elementos involucrados en el desarrollo de aplicaciones Android, ya que podemos tener diferentes herramientas. También podemos descargar algún paquete que ya tenga todo listo y configurado, pero no dejará de tener estos cuatro elementos.

* Java
* IDE
* Android SDK
* ADT

**Java**

Java es el lenguaje de programación que usamos para escribir el código para generar aplicaciones, en realidad lo que sucede es que usamos java y a este se le suman clases, API’s y otras características exclusivas de Android. El primer paso es tener java solamente, sin todo esto que acabamos de comentar de Android.

Java actualmente es de Oracle y podemos instalarlo y/o utilizarlo sin ningún problema. Tenemos que ir a la página oficial de Oracle de descargar para obtenerlo: [Java Download](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

**\*Nota:** también se puede teclear en cualquier buscador las palabras: “Java JDK Download” y los primeros enlaces dirigen a la sección de descargas de forma directa. Esto es en el caso de que Oracle cambie la URL de la descarga.

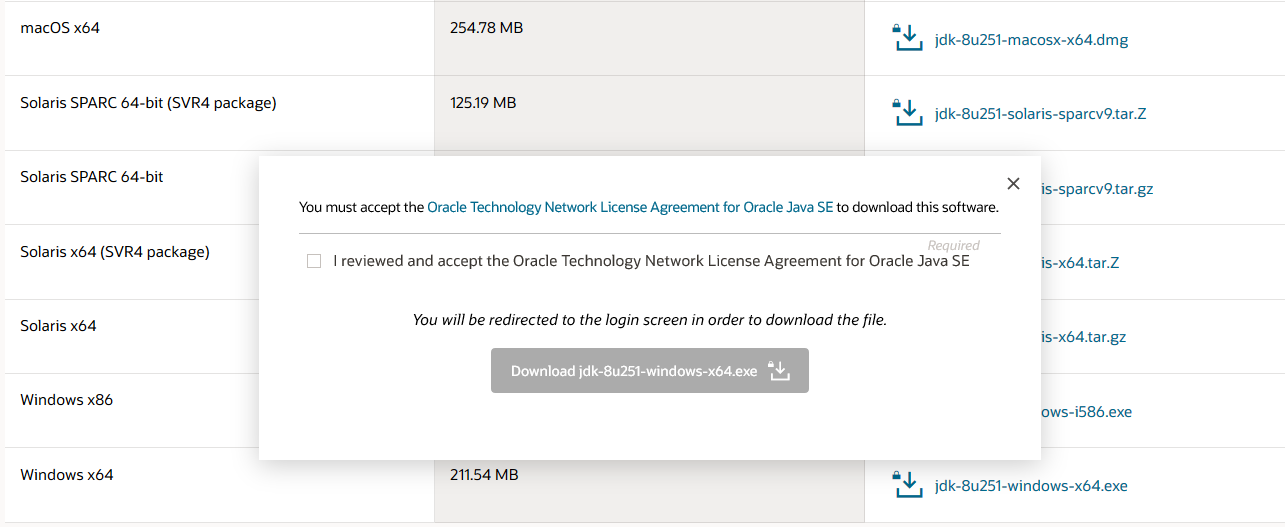
Vamos a encontrar diferentes versiones, lo que nos interesa descargar es el **Java JDK (Java Development Kit)** que es el kit de desarrollo de Java, este es muy variable que ya se mantiene actualizado de primera mano, esto quiere decir que vamos a ir viendo con el paso del tiempo que el número de versión es diferente, pero esto no importa ya que la versión que estamos viendo siempre en el primer cuadro (la primera opción que nos pone Oracle siempre en la web) es la versión estable que ellos recomiendan para desarrollo y siempre debería funcionar de forma correcta.

En la parte de la descarga siempre nos dan dos opciones como vemos en la siguiente imagen:

****Opción 1 (izquierda):** Es solamente el Java JDK

**Opción 2 (derecha):** Es el JDK y además es el IDE llamado NetBeans, en este también se puede desarrollar una aplicación Android, pero como la mayoría del mercado actual vamos a usar Eclipse como IDE para el desarrollo de nuestras aplicaciones Android.

Después de seleccionar la descarga siempre nos va a decir que aceptemos la licencia antes de descargar Java, después de aceptar tenemos que seleccionar la opción que necesitamos para nuestro PC.



En el caso de:

* **Windows:** Se tiene que verificar si la versión es el procesador de 32 o 64 bits
* **Linux:** En el caso de Linux tenemos que ver si es de 32 o 64 bits y además si la descarga se quiere como RPM

o TAR

* **Mac:** En este caso solo tenemos una opción que es para una máquina con MAC OS de 64 bits

Después de realizar la descarga, la instalación se realiza como cualquier aplicación dependiendo del sistema operativo utilizado.

**IDE - Eclipse**

El IDE (Integrated Development Environment) en su traducción significa “Entorno de Desarrollo Integrado”. El IDE es un conjunto de herramientas de programación que nos permite desarrollar diferentes actividades como: programar, ejecutar, compilar, corregir, buscar errores, entre otras.

Eclipse es un proyecto “open source” así que solo tenemos que ir a la página oficial del proyecto para descargarlo: [Eclipse Download](http://www.eclipse.org)

Nos dirigimos a la sección de descargas y nos dará diferentes versiones disponibles.

**\*Nota:** por defecto nos da la versión disponible para nuestro sistema operativo que detectó el navegador, en el caso de que veamos que no coincide con el mismo debemos seleccionarlo por nuestra cuenta.



**Android SDK**

El SDK es lo que necesitamos añadir a Java para poder crear aplicaciones Android, este lo proporciona Google ya que es la empresa que está detrás de Android.

**\*Nota:** se puede ir al buscador y colocar “Android SDK Download” y los primeros enlaces nos enviarán a la página “developer.android.com” que es el sitio oficial de Android y nos proporcionará la descargar de todo lo que necesitamos, incluido el SDK.

**ADT Plugin**

El ADT es un Plugin que nos permite integrar Android con Eclipse. Para esto tenemos que seguir una serie de pasos de instalación que se usan para agregar cualquier Plugin a Eclipse.

Vamos a la URL del ADT: [ADT Plugin](https://stuff.mit.edu/afs/sipb/project/android/docs/sdk/installing/installing-adt.html)

Después, seguimos las instrucciones siguientes:

* Abrir Eclipse
* Ir a la sección “Help” y seleccionamos “Install New Software”
* Hacer click en el botón “Add” para agregar un repositorio
* En el recuadro “Location” colocar la ruta del Plugin, se puede copiar de la web del ADT o teclear lo siguiente:

[***https://dl-ssl.google.com/android/eclipse/***](https://dl-ssl.google.com/android/eclipse/)

Después, seleccionar la casilla de “Developer Tools” para comenzar la instalación y hacer click en “Next” las veces que lo solicite y aceptar la licencia.

**DESCARGAR VERSIONES DE ANDROID PARA TRABAJAR**

Ya tenemos todo listo, pero necesitamos algo más, descargar las versiones de Android con las que queremos programar.

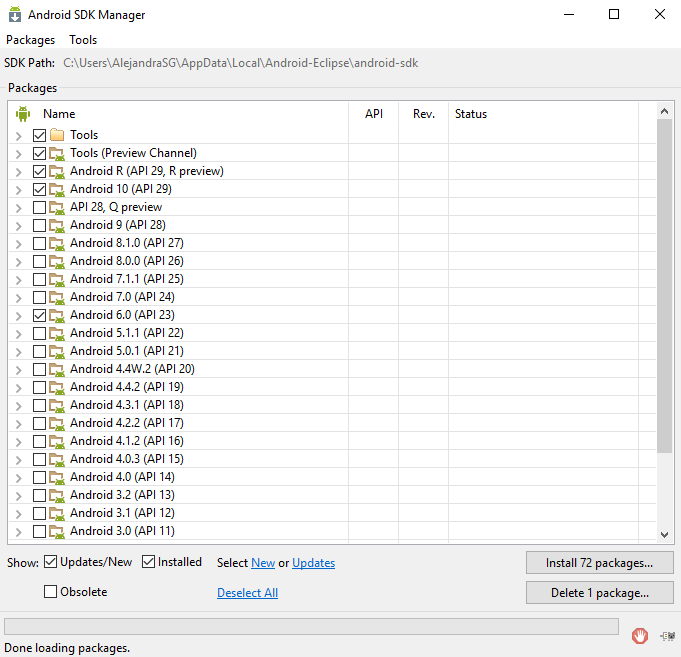
Debemos aclarar que podemos descargar desde la versión primera de Android hasta la última al día que estamos deseando hacer la descarga, pero esto no es necesario ya que Android se maneja por un orden que son las API’s.

|  |  |
| --- | --- |
| * Android 1.5 – API 3 * Android 1.6 – API 4 * Android 2.1 – API 7 * Android 2.2 – API 8 * Android 2.3.3 – API 10 * Android 3.0 – API 11 * Android 3.1 – API 12 * Android 3.2 – API 13 * Android 4.0 – API 14 * Android 4.0.3 – API 15 * Android 4.1.1 – API 16 * Android 4.2.2 API 17 | * Android 4.3 – API 18 * Android 4.4.2 – API 19 * Android 4.4W – API 20 * Android 5.0 – API 21 * Android 5.1 – API 22 * Android 6.0 – API 23 * Android 7.0 – API 24 * Android 7.1 – API 25 * Android 8.0 – API 26 * Android 8.1 – API 27 * Android 9.0 – API 28 * Android 10 – API 29 |

En la actualidad el mercado Android está dividido solo entre algunas versiones: Gingerbread, Ice Cream, Jelly Bean y KitKat. Entre estas ocupan más del 95% de los dispositivos en activo, es decir, que desarrollar basándose en otras API’s no tendría mucho sentido.

Para descargar las versiones de Android tenemos que abrir el “Android SDK Manager”:

* Abrimos Eclipse
* Seleccionamos Window – Android SDK Manager

De estas seleccionamos en realidad las que consideremos, se puede descargar solo una o todas, eso depende del desarrollador. Automáticamente se conecta a un servidor y realizará las descargas que seleccionamos.

Este procedimiento es el mismo para la actualización de las API’s.

**\*Nota:** como recomendación es bueno descargar los apartados que llevan los nombres de “Tools” y “Extras” ya que probablemente como desarrolladores Android en algún momento los vamos a utilizar.

Finalmente, ya tenemos todo lo que necesitamos para nuestro entorno de desarrollo.

**Emulador Android**

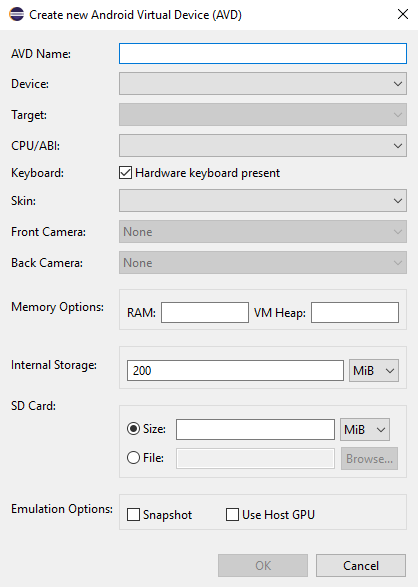
El concepto de emulador lo podemos interpretar de la siguiente forma: con software vamos a imitar el hardware. En el caso de Android lo que el emulador hace es recrear el hardware de un móvil para hacer funcionar el SO Android sobre él.

El emulador nos sirve para comparar una aplicación en diferentes versiones del SO, además en diversos tamaños de pantallas, marcas y modelos de móviles. Así no tendremos que tener un móvil de cada tamaño, modelo, etc.

En Eclipse vamos a la opción **Window**, después al **AVD Manager o Android Virtual Device Manager**, en la pestaña de **Android Virtual Devices** seleccionamos **Create**.

Camino: **Window** -> **Android Virtual Device Manager** -> **Create**

Después de esto vamos a ver una ventada como la siguiente:

Tenemos que llenar los campos para configurar el emulador que vamos a crear.

**AVD Name:** Colocamos el nombre del dispositivo, el nombre puede ser el que nosotros le asignemos. Por recomendación podemos colocar un nombre que identifique al emulador, por ejemplo: “android-v4.0”. Con esto detectamos que este emulador está configurado con la v4.0 de android. Podemos tener diferentes emuladores configurados o predefinidos.

**Device:** Seleccionamos el dispositivo que deseamos emular, en principio tenemos a mano todos los modelos Nexus, pero también está la familia Galaxy, entre otras. Además, podemos seleccionar diferentes modelos con varias dimensiones de pantalla.

**Target:** Ponemos la versión de Android que le vamos a colocar a este dispositivo (muchas veces conviene dejar la que está por defecto después de seleccionar el campo *Device****,*** pero se puede cambiar)

**CPU/ABI:** Seleccionamos el procesador que tendrá nuestro emulador, conviene dejar el que viene por defecto.

**Keyboard:** Los seleccionamos si queremos usar el teclado del PC.

**Skin:** Dependiendo de lo que tengamos seleccionado, nos permite ver los botones físicos de un móvil, por ejemplo, el Home.

**Front Camera:** Podemos emular la cámara, pero por recomendación si necesitamos una cámara sería bueno probarlo en un móvil real.

**Back Camera:** Lo mismo que el campo anterior.

**Memory Option:** Esta opción tiene dos apartados, RAM y VM Heap. Debemos recordar que al asignar RAM al emulador este la obtiene de nuestro PC, entonces el funcionamiento será más lento, asignándole 1 GB de RAM al emulador debe ser suficiente. La segunda opción es la memoria dinámica asignada a la máquina virtual, por recomendación podemos dejar la que aparece por defecto.

**Internal Storage:** Es la memoria interna del móvil y al igual que el punto anterior la toma de nuestro PC, al ser solo para pruebas no tenemos que asignar algo considerable, podríamos poner algo como 200-400 m y no deberíamos tener ningún problema.

**SD Card:** Esta es la tarjeta de memoria, el emulador creará un archivo y este se comportará como la tarjeta SD, no debería ser necesario asignar más de 50 mb en una aplicación promedio.

**Snapshot:** Sirve para que vaya más rápido el emulador, lo que sucede es que al cerrar el emulador la siguiente ocasión se ejecutará más rápido porque tiene el último estado antes de cerrar.

**Host GPU:** Si la seleccionamos el emulador utilizará el GPU del PC, esto nos sirve para que el emulador sea mucho más fluido ya que el procesamiento de gráficos lo realiza el PC.

Finalmente, ya tenemos configurado nuestro emulador.



El inicio del emulador depende de los recursos del equipo que lo ejecuta, si no tenemos un equipo muy actualizado puede tardar hasta 20 minutos en iniciar, si tenemos un equipo actualizado puede tardar de media de 1 a 3 minutos en iniciar.

**Ejecutar una aplicación en el emulador**

Para ejecutar una aplicación en el emulador debemos hacer lo siguiente:

1. Abrir en el editor un archivo .java de nuestra aplicación, si tenemos 20 archivos basta con tener solo uno en el editor abierto, pero debe ser un archivo que sea .java, no debe ser un XML o cualquier otro.
2. Tener abierto el emulador, en caso de no estar abierto, ejecutarlo y esperar a que el emulador muestre la pantalla que aparece cuando desbloqueamos el móvil.
3. En Eclipse ir en el menú al botón que se parece al botón Play de cualquier dispositivo electrónico de color verde. También se puede llegar si vamos a la opción Run del menú y después seleccionamos Run. Tenemos una tercera opción en la que se puede usar la combinación del teclado Ctrl + F11.

Camino 1: **Botón verde de play en el menú superior de Eclipse**

Camino 2: **Run -> Run**

Camino 3: **Ctrl + F11**

Si no tenemos ningún error, nuestra aplicación después de unos segundos comenzará a ejecutarse en el emulador Android que esté funcionando. Si tenemos dos emuladores, o un emulador y un dispositivo conectado, nos preguntará sobre cuál queremos ejecutar nuestra aplicación.

**4. ESTRUCTURA DE UNA APLICACIÓN**

**Crear una aplicación “HolaMundo”**

Para analizar la estructura de una aplicación primero vamos a crear el clásico “Hola mundo”. En realidad, en Android siempre que creamos un proyecto nuevo este es un “Holamundo” sin necesidad de hacer algo especial o extra.

Con Eclipse abierto vamos a la opción ***File*** después la opción ***New*** y luego ***Android Application Project***. Si no aparece esta opción a primera vista podemos seguir otro camino: ***File – New – Other***.

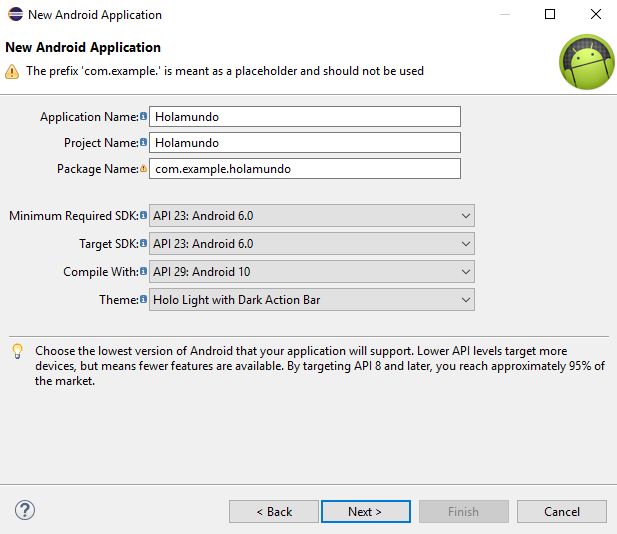
En ***Other*** encontraremos diferentes opciones, seleccionamos la opción ***Android*** y en las primeras opciones encontraremos ***Android Application Project***.

Camino 1: ***File -> New -> Android Application Project***

Camino 2: ***File – New -> Other -> Android -> Android Application Project***

Cualquiera de los caminos nos lleva al mismo destino: comenzar a configurar nuestro primer proyecto.

Después se abre una ventana que nos solicita unas opciones:

**Application Name:** Este es el nombre de la aplicación, el que aparecerá en la playstore en caso de publicarla; para el ejercicio la llamamos “Holamundo”. En el momento de colocar el nombre de los demás campos se llenan por defecto con este.

**Project Name:** Este es el nombre del proyecto para Eclipse, habitualmente se llama igual que el primer campo. Por recomendación no se colocan espacios en el nombre ni caracteres especiales, ya que será el nombre del directorio del proyecto.

**Package Name:** Este es el nombre del paquete. Debemos tratar que sea único desde el principio, este es el nombre del paquete que se subirá a la playstore en caso de publicar la aplicación. Por convención casi siempre se inicia con “com.xxxxxx” haciendo referencia al dominio de la empresa, compañía o persona que lo creó. Entre las palabras que forman parte del nombre se puede colocar punto “.” en lugar de los espacios. Por ejemplo: com.tuempresa.tunombre.nombreapp.

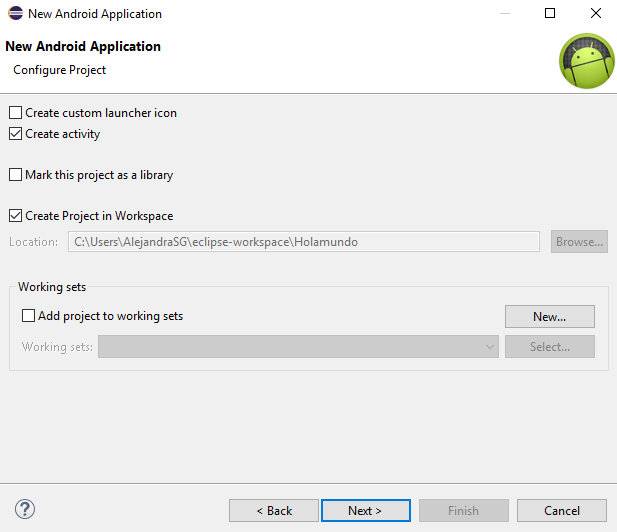
**Minimum Required SDK**: Debemos seleccionar la versión mínima del SDK que aceptará nuestra aplicación, es decir, que si colocamos 2.3.3 el móvil que quiera instalar nuestra aplicación mínimo debe tener la versión 2.3.3.

**Target SDK**: La aplicación es capaz de funcionar en las versiones antiguas (hasta minSdkVersion), pero se puso a prueba explícitamente para trabajar con la versión especificada aquí.

**Compile With:** Seleccionamos la versión del SDK con la que compilaremos el proyecto, muchas veces se acostumbra colocar aquí y en el campo anterior la versión más nueva de Android.

**Theme:** Seleccionamos un tema, aunque por ahora no trabajaremos con temas.

Al terminar de llenar los campos, hacemos click en *Siguiente* y nos abrirá una ventana como esta:

****

**Create Custom launcher icon:** Eclipse nos preguntará si queremos crear desde el principio el icono de nuestra aplicación, por ahora no lo seleccionamos ya que más adelante veremos cómo colocar el icono a una aplicación.

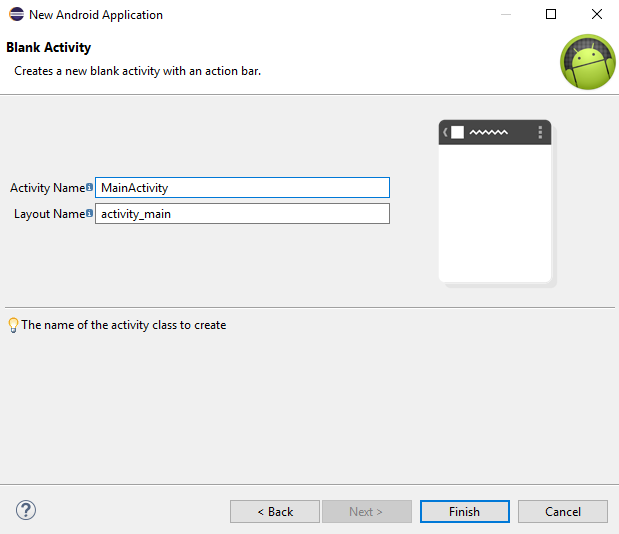
**Create activity:** Nos pregunta si queremos crear un activity, de momento manejaremos el concepto de que una actividad es una ventana de nuestra aplicación, más adelante profundizaremos en el concepto. Por ahora marcamos este campo.

**Mark this project as a library**: Esto no es necesario seleccionarlo casi nunca, sirve para que el proyecto que estamos creándolo marque como librería.

**Create Project in Workspace:** Aquí nos está indicando la ruta del proyecto que estamos creando, si la queremos cambiar la configuramos manualmente, pero por ahora la mantenemos así.

Finalmente, hacemos click en *Siguiente*y nos preguntará qué tipo de ***activity*** queremos crear, seleccionamos ***Blank Activity***.

Ahora se abre una ventana como la siguiente:

**Activity Name:** Este es el nombre de la actividad, quiere decir que es el nombre de la clase que controla la primera pantalla, por convención la primera “activity” de una aplicación se llama MainActivity.

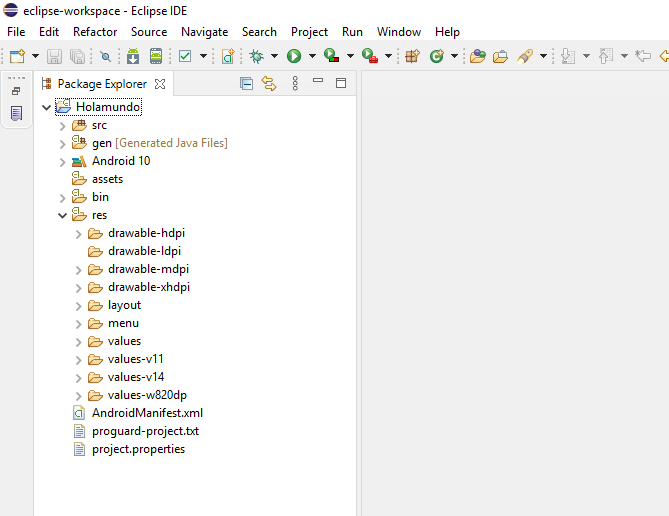
**Layout Name:** Este es el nombre de la maquetación o del archivo que contiene los elementos gráficos de la primera pantalla de nuestra aplicación. Normalmente se llama igual que la primera activity, podemos dejarla como está.

Hacemos click en finalizar.

Ahora ya tenemos lista nuestra aplicación Android, el clásico “holamundo”.

**Estructura Básica de una Aplicación**

La estructura actual de nuestra aplicación es como la de la siguiente imagen:



**Directorio src**: El directorio src contiene todos los archivos .java de una aplicación, esto quiere decir que son todos los archivos que contienen código Java. Estos archivos pueden estar organizados por paquetes si el desarrollador así lo desea.

**Directorio gen:** Son archivos que genera java y que no tenemos por qué tocarlos. Probablemente si tocamos estos archivos ya no van a servir para Eclipse y nuestro proyecto ya no funcionaría. Dentro del directorio gen encontraremos 2 archivos, el BuildConfig y R.

El archivo R contiene los identificadores de todo lo que tiene la aplicación, por ejemplo, imágenes, campos de texto, botones, etc. Java le asigna un identificador y nosotros no tenemos que preocuparnos por él, ya nosotros solo le colocamos un nombre común que podemos recordar y Java sabe cómo se llama para nosotros.

**Paquetes Android 4.0, Android Private Libraries, Android Dependencias**

Estos son solo paquetes que contienen librerías que podríamos utilizar, el paquete que dice Android 4.0, en realidad cambia dependiendo de la versión de Android para la que esté creado nuestro proyecto.

**Directorio assets:** Este contiene recursos de ayuda para una aplicación, por ejemplo, vídeos, audios, bases de datos, este directorio se confunde con el directorio “res” que también es de recursos, pero la diferencia es que los que se encuentran en “assets” no generan un identificador en el archivo R que vimos con anterioridad.

**Directorio bin:** Aquí encontraremos archivos generados por Eclipse y Java que tampoco tenemos que manipular.

**Directorio libs:** Se encuentran librerías externas que necesita el proyecto.

**Directorio res/drawble:** Contiene las imágenes y gráficos PNG que vamos a incluir en nuestra aplicación, cada uno de los directorios representa una densidad.

**Directorio res/layout:** Contiene los archivos XML que representan la parte gráfica de nuestras “Actividades”, con esto queremos decir todas las pantallas de nuestra aplicación.

**Directorio res/menu:** Si tenemos algún menú definido para nuestra aplicación, aquí se encuentran los archivos.

**Directorio res/values:** Se encuentran los archivos que contienen las cadenas de texto que usamos en nuestra aplicación, también hallamos algunos estilos.

**Archivo Manifest.xml:** Este archivo es el más importante de nuestra aplicación, se le considera la columna vertebral de todo proyecto Android, por ejemplo, ahí declaramos todas las actividades del proyecto, permisos, versión del SDK y otras cosas.

El archivo AndroidManifest.xml del ejemplo “Holamundo” contiene el siguiente código:



**Etiqueta <manifest>**

Es la etiqueta principal del archivo, tiene el atributo xmlns que siempre lo coloca Eclipse por defecto y no tendríamos porqué modificarlo.



El atributo **package** hace referencia al nombre del paquete de la aplicación, en caso de que cambiemos el nombre de la aplicación, en ocasiones Eclipse no lo cambia y nos tocará cambiarlo a nosotros.

Atributo **VersionCode,** este es el número de versión del código de la aplicación, nos sirve para saber con qué versión estamos trabajando.

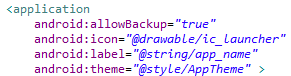
Atributo **VersionName**, se trata del número de versión, pero la diferencia con el anterior es que este número es el que se muestra en la Tienda de aplicaciones. Podemos decir que el *VersionCode* es para el desarrollador y el *VersionName* es para el público en general.

**Etiqueta sdk**

Atributo **minSdkVersion**, nos indica la versión mínima de SDK con la que debería de funcionar la aplicación, cuando creamos el proyecto lo indicamos, aunque se puede cambiar manualmente borrando y colocando otro número de SDK.

Atributo **targetSdkVersion**, es la versión de la API a la que se dirige principalmente nuestra aplicación.

**Etiqueta Application**



Dentro de esta etiqueta tenemos todos los elementos que forman parte de nuestra aplicación.

Atributo **allowBackup**, sirve para permitir al sistema realizar una copia de seguridad de la aplicación y el contenido de la misma, si colocamos el valor en “true” lo estamos permitiendo, pero si colocamos el valor en “false” lo estamos prohibiendo.

Atributo **icon**, es el icono de la aplicación.

Atributo **label**, es el nombre de la aplicación y lo tomamos del archivo **strings** que analizaremos más adelante.

**Etiqueta Activity**

La etiqueta *Activity* sirve para dar de alta una actividad, si tenemos 5 actividades en nuestra aplicación deberíamos de tener 5 etiquetas activity, si no las damos de alta en nuestro archivo AndroidManifest.xml, la actividad generará un error en el momento de ser llamada.

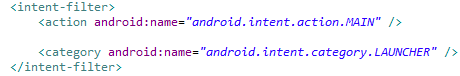


Atributo **name**, este es el nombre de la clase Java que implementa Activity

Atributo **label**, este es el nombre de Activity que el usuario va a ver.

**Etiqueta Intent-filter**

Sirve para especificar lo que se tiene permitido con este Activity; además, nos permite ver sus intenciones.



**Etiqueta action**

Nos permite colocarle un nombre para más adelante llamar a esta actividad y realizar otras operaciones

**Etiqueta category**

Esta contiene varias opciones para colocar, pero la mayoría de las ocasiones utilizamos *DEFAULT* o *LAUNCHER*.

Usamos *LAUNCHER* solo una vez dentro de todo el Manifest. Con esto le indicamos que esta es la actividad principal, la que se debe ejecutar cuando la aplicación se inicie.

Todas las demás actividades llevan el valor “*DEFAULT*”ya que solo indica que se ejecuta, en caso de tener 10 actividades, tendríamos 1 con *LAUNCHER* Y 9 con *DEFAULT*.

**5. CICLO DE VIDA DE UNA ACTIVIDAD**

**¿Qué es una Activity?**

Primero vamos a conocer el concepto de “Activity” o Actividad en español (desde aquí en adelante lo nombraremos como Actividad). Este concepto es básico en el mundo Android y para entenderlo mejor por ahora podemos quedarnos con que hace referencia a las pantallas. Cuando queremos crear una pantalla para una aplicación podemos decir que técnicamente estamos creando una actividad, si nuestra aplicación tiene 5 pantallas decimos que tiene 5 actividades. Esto puede variar porque hay otras formas de crear pantallas, pero la actividad es la forma básica, también existen los Fragment, FragmentActivity y otras más.

En Java casi todo viene dentro en una clase o es una clase, en Android sucede lo mismo. La clase más usada lleva por nombre “Activity”, cuando vamos a crear una pantalla hacemos que esta herede de “Activity” que es una clase ya definida por Android y así solo reutilizamos y modificamos a nuestro gusto. Entonces si creamos “Pantalla UNO” y queremos que esta se vuelva una Actividad, lo que hacemos es marcar una herencia de la clase “Activity” para que se comporte de esta forma y la podamos utilizar como tal.

Las actividades (pantallas) tienen un ciclo de vida, se crean, se inician, se pausan, se reinician, se detienen y se destruyen. Es importante conocer el ciclo de vida de las actividades ya que podemos aprovecharlo.

El ciclo de vida de una actividad tiene estados, pero para llegar a estos tenemos métodos predefinidos a los que nosotros les podemos colocar código para realizar algo específico.

Ejemplo: cuando inicia una actividad en el momento de crearse pasa por el método *onCreate*, es decir, que antes de crear la actividad pasamos al método y ejecutamos lo que dice, después de esto decimos que la actividad está creada, su estado después de pasar por el *onCreate* sería “creada”. Así como el método *onCreate* tenemos una lista de métodos:

|  |  |
| --- | --- |
| * onCreate * onStart * onResume * onPause | * onStop * onRestart * onDestroy |

Conozcamos para qué nos sirve cada uno de estos métodos.

**onCreate**:

Es el que debemos ejecutar siempre en un inicio de una actividad, en este definimos por ejemplo la interfaz de la actividad, las variables, etc. Este método por lógica solo se ejecuta una vez, en el momento de invocar o llamar a la actividad. En este método vamos a encontrar en la mayoría de las ocasiones cómo se define un archivo XML como la parte gráfica de la actividad o también la configuración de la interfaz.

Cuando el método onCreate termina de ejecutarse llama al método onStart seguido de onResume, esto sucede de manera muy rápida.

La actividad se vuelve visible para el usuario cuando llamamos al método onStart, pero como sigue muy rápido el onResume, en este onResume permanece la actividad hasta que sucede “algo” con esta actividad.

**onStart**

Aquí es donde la actividad se comienza a mostrar al usuario.

**onResume**

Es cuando la actividad entra en primer plano y el usuario interactúa con la actividad, la palabra con la que podríamos traducir este sería “corriendo” o “ejecutando”.

**onPaused**

Es cuando se encuentra parcialmente oscurecida por una actividad que se halla en primer plano; por ejemplo, está medio transparente o no cubre toda la pantalla, en este estado no se reciben datos de entrada del usuario y no puede ejecutarse código.

**onStop**

En este método se encuentra completamente invisible u oculto para el usuario, podemos decir que se encuentra en el “fondo” o que se congela. Por ejemplo, las variables e información se mantienen, pero no podemos ejecutar el código.

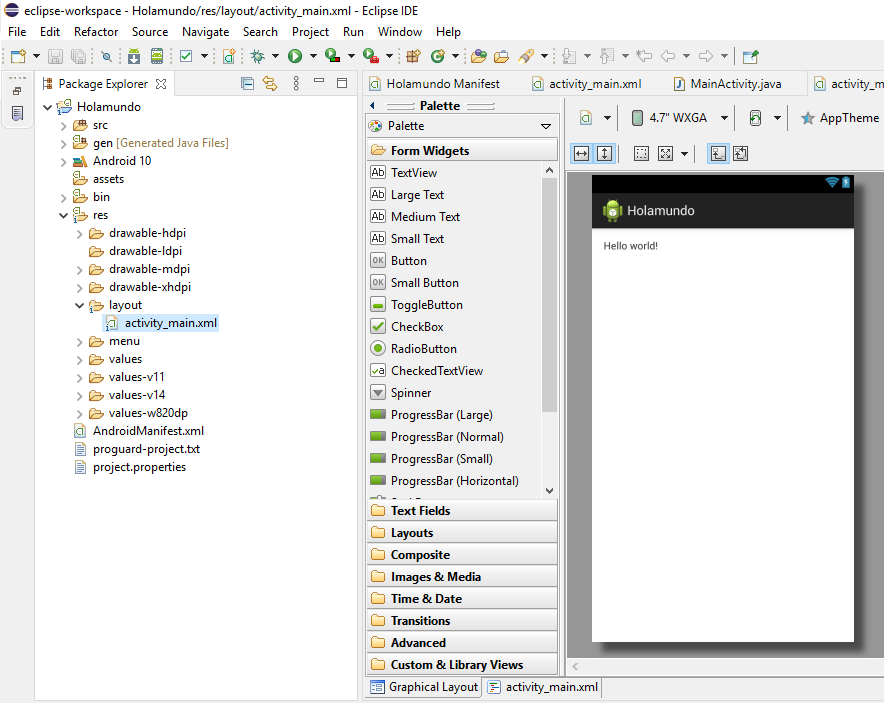
**onRestart**

Este método se llama después del onStop cuando la actividad actual se está volviendo a mostrar al usuario, es decir, cuando se regresa a la actividad. Después de este continúa el onStart, luego el onResume y finalmente ya está de nuevo mostrándose la actividad al usuario.

**onDestroy**

Cuando el sistema destruye la actividad se manda a llamar al método onDestroy para la actividad. Este método es la última oportunidad que tenemos de limpiar los recursos y si no eliminamos podrían tener un mal efecto en el rendimiento para el usuario. Es buena práctica asegurarse de que los hilos que creamos sean destruidos y las acciones de larga duración también sean detenidas.

**6. BASES DE UNA INTERFAZ GRÁFICA**



Las interfaces gráficas de Android se pueden trabajar de 2 formas: de forma visual (Drag and Drop) o con código. Para seleccionar una de ellas en la parte interior de la imagen podemos ver que toda interfaz gráfica tiene 2 pestañas, solamente debemos seleccionar el archivo XML que queremos ver con doble click. Al seleccionarlo observamos las pestañas Graphical Layout y activity\_main.xml (el segundo siempre es el nombre del archivo XML, el nombre de la primera pestaña nunca cambia). La primera pestaña hace referencia a la parte “visual” y podemos utilizar el modo Drag and Drop, esto quiere decir que tomamos un elemento del menú y lo soltamos en la posición que queremos se coloque, a muchos diseñadores este les resulta más cómodo ya que es cómo funcionan la mayoría de herramientas de diseño. El segundo modo nos mostrará el mismo archivo, pero viéndolo como un archivo XML, es decir, que se ven las etiquetas definiendo cada elemento gráfico con sus atributos y valores que les podemos asignar.

**\*Nota:** también se pueden agregar elementos desde el código Java, todo depende de las necesidades de nuestra aplicación.

**Layout**

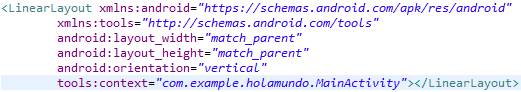
El concepto de layout lo podemos entender de una manera sencilla si lo visualizamos como un contenedor. En él vamos a ir colocando elementos como los botones, imágenes, texto y también otros layouts. Estos contenedores nos permiten una colocación estructurada de los elementos y tenemos diferentes formas de colocarlos, en principio contamos con 2 estructuras:

* LinearLayout
* RelativeLayout

**LinearLayout**

Este contenedor lo podemos ver como una caja que siempre mantiene una colocación horizontal o vertical.

Para crear LinearLayout colocamos el siguiente código:



La etiqueta que se utiliza para crear un LinearLayout es <LinearLayout> y siempre se debe cerrar </LinearLayout>. Las líneas que se encuentran entre el inicio de la etiqueta “<LinearLayout” y el cierre “>” se llaman propiedades o atributos y no siempre tienen que ser las mismas, pueden variar dependiendo de las necesidades, además no solo son para el elemento LinearLayout, cualquier elemento colocado en el XML tiene propiedades.

Las propiedades xmlns:android, xmlns:tools y tools:context son 3 líneas de código que Eclipse coloca automáticamente para el contenedor “padre” de nuestra actividad, esto quiere decir que, si tenemos un layout dentro de otro, solo el layout principal o layout padre (layout que contiene otros) tiene estas tres líneas de código.

La propiedad layout\_width asigna un ancho al contenedor y tenemos en principio 3 valores que le podemos colocar: match\_parent, fill\_parent, wrap\_content.

**match\_parent:** provoca que el layout sea del tamaño del padre. Normalmente el primer layout de nuestra aplicación siempre tiene el match\_parent, esto es para que el primer layout tenga el ancho de la pantalla del móvil o celular.

**fill\_parent:** este valor realiza lo mismo que match\_parent solo que está en desuso. En lugar de fill\_parent actualmente se coloca match\_parent.

**wrap\_content:** vuelve dinámico el tamaño, siempre va a depender del contenido. Es decir, que, si tenemos una cadena de texto de cien caracteres dentro de este layout, el ancho de este layout sería de 100 caracteres, pero si la cadena de caracteres es de 20, estos 20 caracteres serían el ancho.

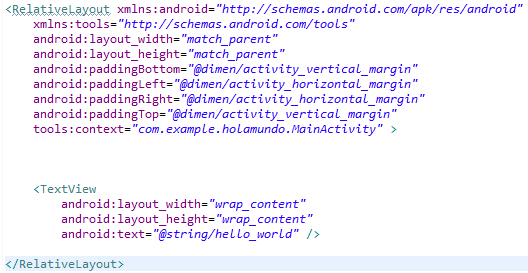
La siguiente propiedad **layout\_height** es similar al layout\_width la diferencia es que layout\_width hace referencia al ancho del contenedor mientras layout\_height hace referencia al alto del contenedor, también pueden tomar los mismos valores: match\_parent, fill\_parent, wrap\_content.

La propiedad **android:orientation=”vertical”** indica la orientación del posicionamiento de elementos, por ejemplo: si tenemos seleccionado el valor vertical, el primer elemento se coloca en la parte superior de la pantalla, el siguiente elemento se colocará debajo de este, si agregamos un tercero este estará debajo del segundo y así sucesivamente, aunque tengamos un espacio para colocar los dos elementos o más, siempre va a ir debajo del elemento anterior. Si la orientación la cambiamos a horizontal sucede que todo elemento nuevo se irá colocando a la derecha del elemento anterior.

**RelativeLayout**

Este contenedor como expresa su nombre maneja la colocación “relativa”, con esto queremos decir que la posición de un elemento toma como referencia la colocación de otro elemento. Una manera fácil de explicar esto es con el siguiente ejemplo: Tenemos dos elementos “A” y “B”, el elemento “B” al colocarlo dentro de un RelativeLayout hay que indicarle cuál es su posición y en relación con qué elemento podemos decir que el elemento “B” tiene su posición relativa a 20 px a la derecha del elemento “A”. Si el elemento “A” sufre algún cambio de posición, el elemento “B” también lo sufrirá ya que su posición depende del “A”, en caso de querer mantenerse la misma posición tendría que cambiar el valor de la distancia que tenía para mantener visualmente el mismo espacio entre ellos. Esto hace mucho más complejo el uso del RelativeLayout, ya que no solamente podemos tener una posición dependiendo de un elemento, se puede tener como referencia 1 o más elementos.

Por defecto cuando creamos el “holamundo” en Android nos coloca todo dentro de un contenedor RelativeLayout. Veamos el código.



La línea xmlns:android y xmlns:tools la coloca Android por defecto al ser el contenedor principal.

Las propiedades layout\_width y layout\_height funcionan de la misma forma que se explicó anteriormente con el LinearLayout.

Las propiedades padding sirven para rellenar, funcionan de la misma manera que se usa el padding en html, reservan un espacio en el contenedor. El padding no es exclusivo del RelativeLayout, también funciona para el LinearLayout y muchos otros elementos gráficos de Android, pero el “holamundo” en Android tiene esta estructura para que la cadena “hola mundo” no esté pegada a la parte superior e izquierda del layout.

**7. HACER UNA APLICACIÓN PARA MÚLTIPLES DISPOSITIVOS**

Para crear interfaces que funcionen en múltiples dispositivos debemos tener claras las características que rodean a los móviles Android: densidades de pantalla, tamaños de pantalla, orientación.

**Densidad de pantalla**

La palabra “densidad” la podemos definir como “la cantidad de masa en un determinado volumen”, en el caso de Android y de las pantallas en específico, podemos decir que es algo muy similar, pero con pixeles. El concepto de densidad de pantalla hace referencia a la cantidad de píxeles en un espacio físico (se mide por píxeles en pulgadas).

Que un teléfono sea de mayor calidad no es por el tamaño de pantalla, podemos tener un teléfono con un tamaño menor de su pantalla y que nos proporciona una mejor calidad de imagen, esto es por la densidad. Como su nombre lo indica la densidad es “la” cantidad dependiendo del espacio que se tiene, podríamos decir que si yo tengo la posibilidad de colocar 100 píxeles en 1cm, pero también tengo la posibilidad de meter 1000 pixeles en 1 cm, esta segunda opción se va a ver mejor, porque puedo tener mayor densidad (900 px más) y esto me va a ayudar a tener mejor calidad de imagen ya que puedo colorear o tener más matices de color. Después de esta explicación ya sabemos que un mayor tamaño de pantalla no aumenta la calidad, esta depende de la densidad.

En el mundo Android contamos con 6 densidades diferentes: Baja, Media, Alta, Extra Extra Alta, Extra Extra Extra Alta.

Su nomenclatura es la siguiente:

Baja densidad – **ldpi** 120 dpi

Media densidad – **mdpi** 160 dpi

Alta densidad – **hdpi** 240 dpi

Extra alta densidad – **xhdpi** 329 dpi

Extra extra alta densidad – **xxhdpi** 480 dpi

Extra extra extra alta densidad – **xxhdpi** 640 dpi

Los números que colocamos a un lado de cada nomenclatura son los puntos por pulgada, pero en inglés se utiliza la abreviatura “dpi” para referirse a ellos. DPI significa (dots per inch) traducido al español: puntos por pulgada.

**Tamaños de pantalla**

En tamaños de pantalla hay 4 como los principales: pequeña, normal, larga y extra lagar.

Pequeño va desde 2 pulgadas hasta 3.7

Normal de 3.5 a 4.7

Largo va de 4.2 a 7

Extra largo de 7 pulgadas a 10

Hay dispositivos que quedan en rangos intermedios; por ejemplo, un dispositivo con 3.6 pulgadas entra en 2 rangos, en el pequeño y en el normal. Como desarrolladores no tenemos que preocuparnos tanto, ya que si creamos bien nuestra aplicación y todos los recursos que creamos están bien realizados para el tamaño pequeño y normal, va a tener un funcionamiento correcto. El fabricante y Android se encargan de colocarlo en una categoría en relación con el hardware y otras características.

**Orientación de pantalla**

La orientación se refiere a la posición del móvil y tenemos dos posiciones: horizontal y vertical. Los móviles tienen un acelerómetro y este es el que determina en qué posición de las dos se encuentra. El sistema operativo Android normalmente se acomoda a cada una de las posiciones; si estamos en la posición vertical tenemos menos espacio de ancho y de forma horizontal el ancho es mayor, esto nos ayuda a que en ciertos casos podamos mostrar diferentes interfaces para el usuario dependiendo de la posición que recibe el acelerómetro.

La orientación de pantalla muchas veces es una de las propiedades gráficas y visuales más olvidadas en el desarrollo de aplicaciones, pero cuando se utiliza bien se vuelve de las mejores armas para aplicaciones eficientes.

Nosotros podemos controlar la orientación, aunque el acelerómetro dice lo contrario. Por ejemplo, podemos decir que nuestra aplicación solamente funcione con la vista horizontal, esta es una técnica que usan mucho los videojuegos para tener más ancho en la pantalla y así obligar al usuario a girar el celular y brindarle una mejor experiencia, porque muchos juegos no podrían funcionar o funcionarían de forma incorrecta si se usaran en modo vertical.

No solamente podemos obligarlo a usar una vista, podemos crear 2 vistas para cada una de nuestras actividades Android, es decir, que si tenemos el móvil de forma vertical este muestre los elementos en cierta posición y si están en modo horizontal los muestre en otro posicionamiento, la colocación va a depender de la orientación del móvil. Esto en realidad genera una muy buena experiencia de usuario ya que estamos aprovechando el ancho y el alto.

Al modo vertical en Android se le conoce como portrait y al modo horizontal, landscape.

**Crear Una Aplicación Que Funcione En Diferentes Tamaños De Pantalla**

Lo normal cuando estamos iniciándonos en el mundo del desarrollo Android es crear nuestras actividades. La forma correcta de crear nuestra aplicación es considerando todas las características que ya comentamos anteriormente: densidades, tamaños y orientación.

Ahora vamos a ver cómo optimizar nuestra aplicación para tener un funcionamiento adecuado dependiendo del tamaño de la pantalla, para esto tenemos que adecuar nuestro archivo XML para cada uno de estos tamaños.

Primero hacemos click derecho en el directorio res de nuestra aplicación en Eclipse y seleccionamos la opción New y luego Folder, ahora creamos una carpeta con el nombre layout-large.

Camino: **directorio res (click derecho) -> New -> Folder -> nombre: layout-large**

Entonces tenemos dos directorios layout por ahora:

* res/layout
* rest/layout-large

En el primero tenemos los XML que están creados para pantallas de un tamaño normal, el segundo es para los XML optimizados para pantallas largas. Si quisiéramos adecuar nuestros XML para extra largas tendríamos que crear el directorio: res (layout-xlarge)

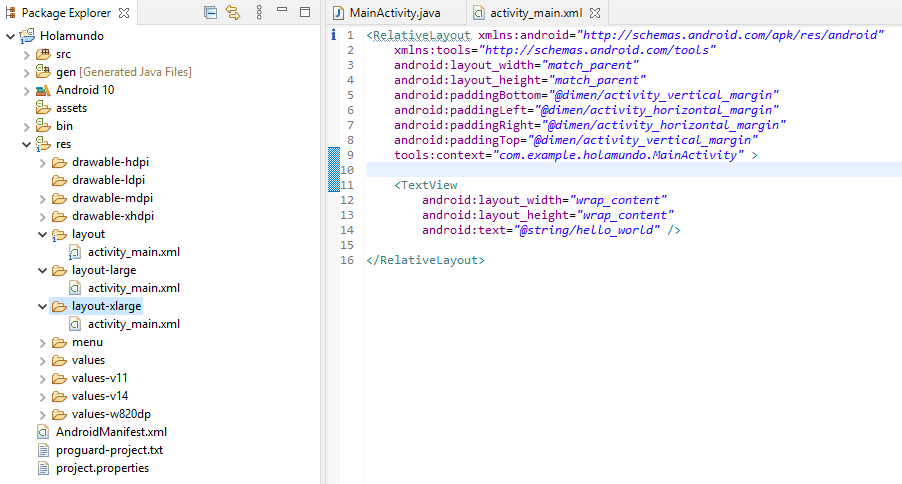
Ahora tenemos tres directorios:

* res/layout
* res/layout-large
* res/layout-xlarge

El tamaño small en la actualidad ya no se usa, la mayoría de los fabricantes abandonaron este formato, así que podríamos descartarlo. Por defecto el directorio layout (sin prefijos) es el considerado estándar, podemos crear una aplicación solo colocando sus interfaces gráficas en el directorio layout, pero probablemente la experiencia del usuario no sería la óptima.

Para que la aplicación funcione de forma correcta todos los XML que representen una pantalla deben tener el mismo nombre, si tenemos un main\_activity.xml como en el “holamundo” que creamos anteriormente, en cada directorio layout deberíamos tener un archivo con el nombre main\_activty.xml. El sistema operativo Android es el que decide qué archivo XML mostrar dependiendo del hardware de cada dispositivo.

Que se llamen igual no significa que deban tener la misma interfaz gráfica, Android optimiza las pantallas. Para la pantalla normal podríamos tener 2 botones y en la pantalla larga 5, dependiendo de las necesidades de nuestra aplicación debemos de modificar cada uno de los XML, Android es el que se va a encargar de mostrar el XML correcto dependiendo del tamaño de la pantalla del dispositivo.



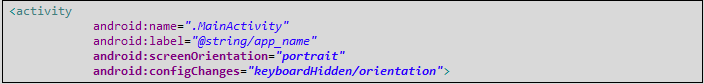
**Crear Pantallas Para Cada Orientación**

Sabemos que hay dos orientaciones: horizontal y vertical. Debemos de considerar que el usuario puede utilizar el móvil en cualquiera de las dos y tenemos diferentes opciones para resolver esto:

* Solo permitir una orientación
* Crear una vista para cada orientación

**Permitir una orientación**

Lo que tenemos que hacer es agregar en el archivo AndroidManifest.xml 2 líneas de código que son los atributos: screenOrientation y configChanges.



El atributo **screenOrientacion** señala la orientación que va a tener esta actividad, así para cada actividad tendremos que colocar su orientación, en caso de no señalar la orientación esta se inicia dependiendo de la orientación del móvil.

El atributo **configChances** es para colocar una lista de configuraciones que van a ser manejadas por nosotros y no por el sistema operativo, en este caso estamos indicando que la orientación queda bajo nuestra responsabilidad y que el móvil no interferirá en lo que nosotros decidimos.

**Crear una vista para cada orientación**

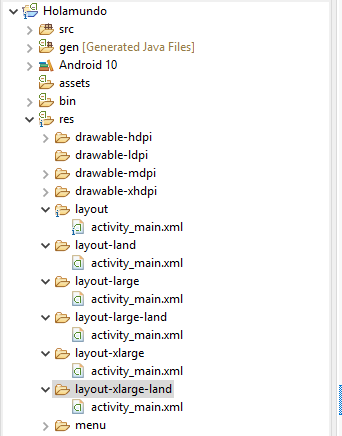
Esta segunda solución sería la recomendación, deberíamos de crear una vista para cada orientación. Para esto debemos de crear un directorio layout para la vista landscape, ya que el directorio que lleva solo el nombre de layout sin el prefijo land es por defecto la vista portrait.

Veamos el directorio res, hacemos click derecho y creamos un nuevo folder, le colocamos el nombre de layout-land. Dentro de este directorio vamos a colocar las mismas actividades que tenemos en el directorio layout si queremos que tengan su versión para las vistas horizontal y vertical.

Camino: **directorio res (click derecho) -> New-> Folder -> nombre: layout-land**

A todos los directorios que ya tenemos le podemos crear uno con el mismo nombre y agregarles el prefijo “land” para poder así tener las versiones verticales y horizontales de cada actividad; cabe recordar que sucede lo mismo a los tamaños de pantallas, cada uno de los XML puede tener elementos independientes, lo importante es que tengan el mismo nombre. Android se encarga por sí mismo de mostrar el que el usuario solicita dependiendo del tamaño de pantalla y de la posición del móvil.

Quedaría algo como la siguiente imagen:



**Crear una aplicación para diferentes densidades**

Uno de los problemas más difíciles de abordar y solucionar es el uso de gráficos en una aplicación Android, esto se debe a que una imagen con determinada medida se va a ver diferente dependiendo de la densidad de cada móvil. Por ejemplo: una imagen de 200x200 ocupa un tamaño en la pantalla en una densidad alta, pero en una densidad extra extra alta se ve de otro tamaño. La tendencia es que cuanto más alta es la densidad del móvil, la imagen de 200x200 se va a ir viendo más pequeña.

Esto no es un error, es algo que sucede por naturaleza de la densidad, al tener más pixeles en un espacio de la pantalla es obvio que la imagen ocupará menos espacio físico, aunque está ocupando la misma cantidad de pixeles, ya que en una densidad alta caben más pixeles por pulgada.

**¿Cómo resolvemos esto?**

En realidad, es muy sencillo, solo tenemos que seguir una regla que Google nos recomienda para que nuestras imágenes se vean del mismo tamaño en una baja densidad y en una extra extra extra alta densidad.

La recomendación es iniciar con el tamaño más grande de la imagen que queremos colocar, primero vamos a crear la imagen para la extra extra extra alta densidad, ocupamos una imagen de 192x192 px (es una medida cualquiera). Lo siguiente es conocer la regla que nos proporciona Google para las escalas de la imagen, los números son los siguientes:

**3:4:6: 8:12:16**

Esto significa que la escala más grande es el 16 (xxxhdpi). Si queremos crear la misma imagen (la de 192x192) para la escala extra extra alta densidad (xxhdpi), lo que tenemos que hacer es dividir 192/16=12. Este número es la base de nuestra escala, ya que para la densidad xxhdpi lo debemos de multiplicar por 12, 12x12 = 144 px. Entonces la segunda imagen queda 144x144.

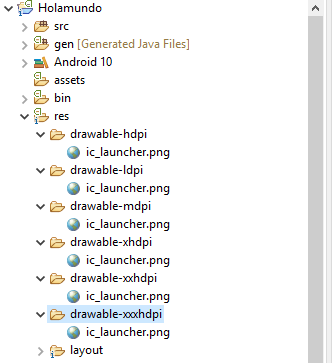
Si hacemos eso con todas las escalas sería lo siguiente:

|  |  |
| --- | --- |
| ldpi = 12 x 3 = 36  mdpi = 12 x 4 = 48  hdpi = 12 x 6 = 72 | xhdpi = 12 x 8 = 96  xxhdpi = 12 x 12 = 144  xxxhdpi = 12 x 16 = 192 |

Debemos de crear una imagen para cada densidad con las medidas respectivas que calculamos, con esto logramos que nuestra imagen se vea de la misma proporción en todas las densidades.

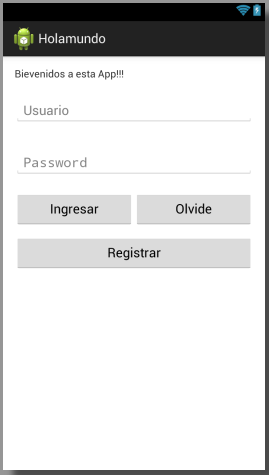
Ya que tenemos la misma imagen para cada una de las densidades, la debemos colocar en la carpeta correspondiente de cada densidad.

Las imágenes para cada densidad deben llevar el mismo nombre y por recomendación en las aplicaciones móviles se utiliza el formato png para las imágenes, esto es por el tamaño de comprensión del formato y por las transparencias. Las imágenes deberían quedar colocadas como la siguiente imagen:



**8. ELEMENTOS BÁSICOS**

Para conocer los elementos básicos de una interfaz Android vamos a crear una pantalla con los elementos, TextView, EditText y Button para estudiar el código a detalle.



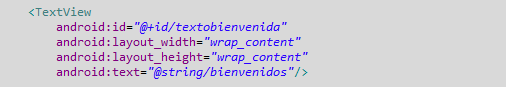
El código para generar una pantalla como la de la imagen anterior es el siguiente:



El contenedor principal ya lo estudiamos anteriormente, así que esta explicación la omitimos.

**TextView**

El TextView es un elemento que nos sirve para colocar texto, pueden ser una o varias líneas dependiendo de nuestras necesidades.



La primera línea no es exclusiva del TextView:



Este id es un atributo que le colocamos de manera individual a todos los elementos que deseamos poder manipular desde el código Java, este “id” es un identificador para que podamos manipular sus valores, estilos, colores, el texto y otras propiedades. Casi cualquier elemento lo puede tener: por ejemplo: botones, edittext, listas, imágenes, etc. Le podemos colocar el nombre que deseemos y es buena idea tratar de colocar nombres que nos digan de qué puede tratar este elemento, por ejemplo, en esta actividad le colocamos de nombre “textobienvenida” ya que es un mensaje de bienvenida al abrir la aplicación.

La segunda y tercera líneas del elemento TextView ya las explicamos anteriormente, así que no repetiremos la explicación de estos atributos.

La última línea



Android nos proporciona un archivo llamado strings, ahí colocamos todas las cadenas de texto que deseamos contenga nuestra aplicación y estas se mandan a llamar colocando el “@string/nombredelacadena” en este tema entraremos a detalle en el capítulo de Aplicación Multilenguaje.

**EditText**

En nuestra pantalla tenemos dos EditText:



Como comentamos en el ejemplo del TextView este es un id que se les coloca a todos los elementos que deseamos más adelante manipular desde el código Java.



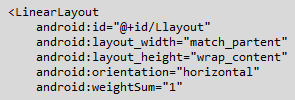


La línea de texto:

El hint es un atributo que nos permite colocar texto dentro del EditText, esto sirve para indicarle al usuario en el mismo campo en donde se escribe el texto cuál es el dato que esperamos este introduzca, nos ayuda a no tener que escribir un texto afuera indicando qué datos esperamos introduzca, el texto colocado con hint se borra de manera automática cuando el usuario comienza a escribir, así facilitamos la experiencia de usuario “diciendo más con menos”.

**LinearLayout**

El segundo LinearLayout contiene lo siguiente:



Todas las líneas ya nos deben ser familiares a excepción de la última

**Pesos en los elementos gráficos**

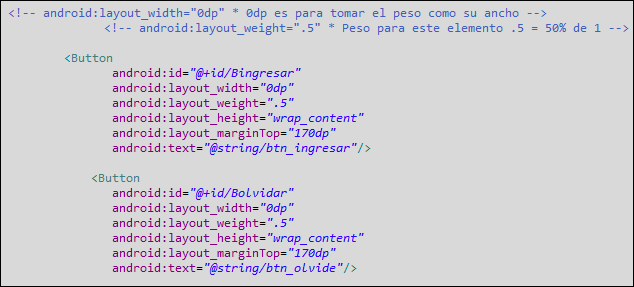
La última línea del segundo LinearLayout:



Esta línea hace referencia a un concepto que aún no conocemos que es el de “peso”. El peso no sirve para distribuir los elementos gráficos en un espacio determinado. En la imagen de nuestro ejercicio podemos ver cómo tenemos dos botones (el botón Ingresar y Olvide) compartiendo la misma línea y en una proporción equitativa, 50% el primero y 50% el segundo. Esto lo logramos con el peso.

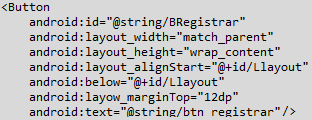
Si queremos distribuir 2 o más elementos en un espacio de la pantalla, lo recomendable es colocarlos en un contenedor (nuestro segundo layout), a este asignarle un peso total, por ejemplo, en nuestro caso le asignamos el valor de “1”, después a cada elemento le colocamos un peso proporcional dependiendo del espacio que queremos que ocupe.

En el siguiente código se explica con comentarios lo que tenemos que hacer por líneas para utilizar el concepto de peso.

La cuestión de los pesos es matemática, si el contenedor padre tiene el peso total en 1, significa que podemos repartir el peso desde 0.1 hasta 0.9, por ejemplo, si un elemento toma el peso de 0.3 significa que del 100% disponible está ocupando el 30%. Si tomamos el valor de 0.4 para un peso, significa que estamos asignando el 40% del total.

**Button**

Los botones son otros de los elementos básicos que tenemos para nuestras interfaces Android.



Como podemos ver los atributos del botón son similares a los del TextView y EditText, el id, el ancho, el alto, y el texto que llevará como mensaje. La mayoría de los elementos como vemos en este ejemplo tienen propiedades similares, aunque también cuentan con propiedades especificas dependiendo de la naturaleza del elemento.

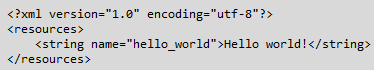
**9. APLICACIÓN MULTILENGUAJE**

Una aplicación multilenguaje significa que tenemos una aplicación que puede mostrar texto dependiendo del idioma seleccionado en el dispositivo.

Vamos a ver cómo crear una aplicación que esté en los idiomas inglés, español y francés.

Android nos proporciona un archivo XML con el que manejamos las cadenas de texto que colocamos en nuestra aplicación. El archivo se llama strings.xml y se encuentra en el directorio: res/values/strings.xml

El archivo tiene esta estructura:



La etiqueta:



Crea una cadena con el nombre “hello\_world” y cuando se le llama desde un elemento como el TextView, EditText u otro, muestra el texto que se encuentra entre las etiquetas <string></string>.

El resto de las líneas del archivo strings.xml que estamos mostrando son líneas que se tienen que colocar por defecto, es parte del formato del archivo.

**Creando un archivo strings para otro idioma**

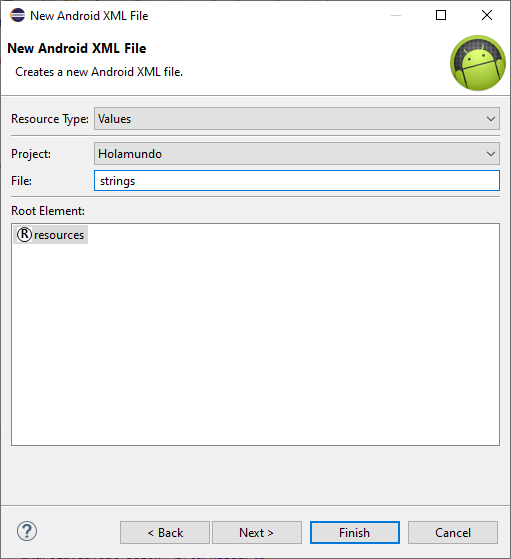
Para que nuestra aplicación sea multilenguaje tenemos que crear otro archivo con las mismas cadenas de texto, pero en otro idioma. Para esto vamos al directorio res y hacemos click derecho, seleccionamos New y luego Folder, en este caso vamos a crear un directorio para nuestras cadenas de texto en francés, entonces tenemos que llamarlo values-fr.

Camino: **Directorio res (click derecho) -> New -> Folder –> nombre: values-fr**

Dentro del directorio values-fr creamos un archivo con el mismo nombre del archivo que se encuentra en la carpeta values, se debe llamar strings.xml.

Para crearlo hacemos click derecho en el directorio values-fr, después en New, seleccionamos New Android XML File, le colocamos el nombre strings. Deberíamos de tener algo como la siguiente imagen.

Camino: **Directorio values-fr (click derecho) -> New -> Android XML File -> nombre: strings.xml**

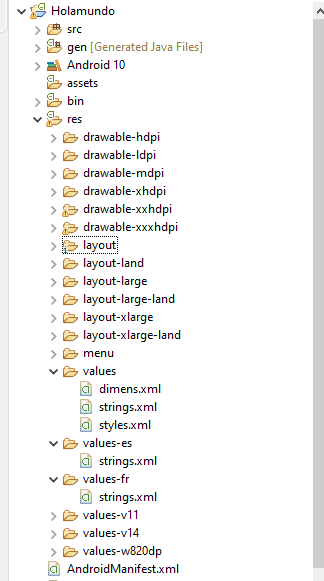
Abrimos el archivo strings.xml que acabamos de crear y que se encuentra en el directorio values-fr, después colocamos el siguiente código.



Con esta línea estamos creando la cadena de texto que lleva por nombre “hello\_world” pero en la frase “¡Hello world!” en francés: “¡Bonjour le monde!”. Como podemos ver, la cadena debe llamarse igual en todos los archivos strings, si tenemos una cadena que se llama “hello\_world” este nombre debe estar en todos los archivos sin importar el idioma, ya que si no la tenemos tendríamos un error.

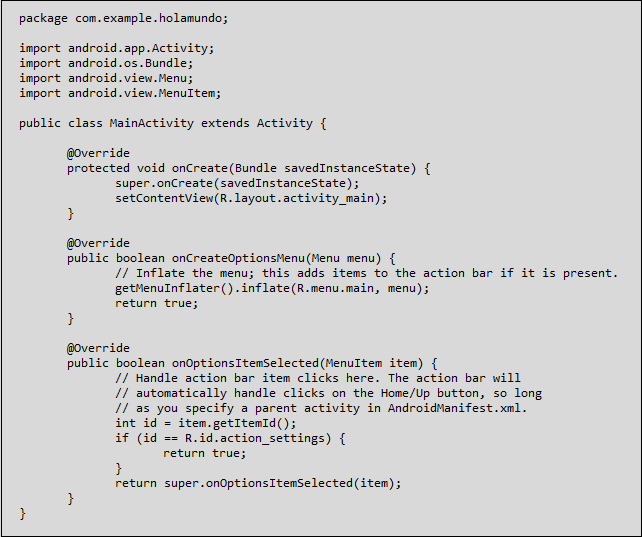
Tenemos que crear un directorio values para cada idioma y le tenemos que agregar la nomenclatura del idioma, por ejemplo, para español es “es”, francés es “fr”, italiano es “it”, para nomenclatura del idioma se utiliza el estándar ISO 639-2.

Ahora para crear nuestra aplicación para español creamos un directorio “values-es” y agregamos la cadena “hello\_world” pero con el texto en el contenido de “¡Hola mundo!”, la estructura de nuestra aplicación debería quedar como la siguiente imagen.

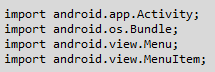


**10. JAVA EN ANDROID**

Nuestro proyecto “Holamundo” contiene un archivo MainActivity.java que es el objeto de estudio en este capítulo, el código es el siguiente:



Se paramos por bloques el código de MainActivity.

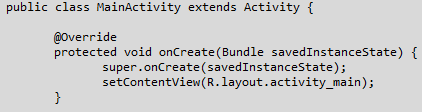


La línea **package** nos indica el nombre del paquete al que pertenece el archivo.

Los **imports** nos permiten utilizar código que ya está definido con anterioridad y así evitarnos escribir código en nuestras aplicaciones desde cero, los imports se hacen tanto en Java como en Android. Aunque cada uno tiene sus clases específicas, por ejemplo si vemos que tiene el prefijo android “import android.app.Activity” probablemente esta es exclusiva de Android y en Java no existe o tiene su derivado, en este caso en concreto Activity es exclusiva de Android.

Lo habitual es que al utilizar una clase de Java tengamos siempre que hacer un import, usemos de ejemplo la clase Activity. Java necesita saber de dónde obtiene el código de esa clase, para poder hacer el import de esta clase. Cada clave que vamos a usar en Java/Android debe ser importada en la cabecera del archivo en el que se está utilizando.

**Crear una clase**



Para crear una clase en Java utilizamos la palabra “class” y ponemos el nombre de la clase, en caso de ser necesario colocamos un modificador de acceso *public*.

El formato básico es el siguiente: Modificador de clase + palabra “class” + nombre de la clase

En código se ve de la siguiente formar:

******

**Modificadores de acceso**

En java tenemos 4 modificadores de acceso que también utilizamos en Android: private, protected, public, default.

Los modificadores de acceso los colocamos a los atributos, métodos y clases. Esto lo hacemos para controlar en nuestro código quién puede utilizar y desde dónde se puede acceder al elemento que ponemos, por ejemplo, si queremos que se pueda acceder a un elemento desde cualquier lugar le colocamos el modificador public.

**Private**: permite acceder solo desde la misma clase.

**Protected:** permite acceder desde la clase, los hijos de la clase y del mismo paquete.

**Public:** permite acceder desde cualquier lugar.

**Default:** (cuando no colocamos un modificador se toma como Default): permite acceder desde el paquete y desde la misma clase.

**Herencia**

Java es un lenguaje orientado a objetos y una de las características del paradigma orientado a objetos es que se utiliza la herencia. La herencia es un concepto fácil de entender, creamos clases y estas clases pueden tener hijos y a su vez estos pueden tener hijos. Cuando tenemos una clase hija significa que está heredando características de la clase padre, pero si la clase de la que hereda también contaba con padre, entonces cuentas con las características de su clase padre y de la clase padre de esta.

Clase A es padre de la clase B

Clase B es padre de la clase C

Entonces la clase C tiene características de la clase B y de la clase A. En código Java para crear la herencia utilizamos la palabra reserva “extends”.

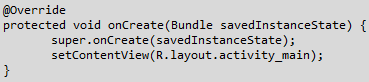
Ya podemos leer completamente la primera línea de nuestro archivo MainActivity.java.

******

Estamos creando una clase con el nombre MainActivity con un modificador public para se pueda acceder a ella desde cualquier lado y estamos haciendo que tome (herede) las características de una clase definida por Android que se llama Activity.

**Métodos**

Las clases al estar definidas en muchas ocasiones necesitamos adecuarlas a nuestra aplicación, para eso las clases cuentan con métodos que nos permiten modificar el estado de objetos, acceder a atributos, asignar valores, entre otras cosas. Muchas veces necesitamos redefinir o colocarles código porque simplemente no lo tienen, como es el caso del método onCreate.

******

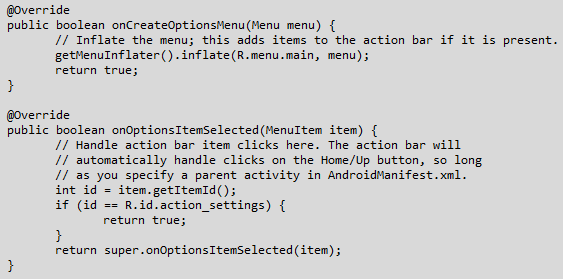
El **@override** significa que estamos sobrescribiendo el método onCreate. Sobrescribir quiere decir que lo estamos redefiniendo porque queremos que haga algo que necesitamos.

En el método onCreate lo que estamos haciendo es indicarle a esta actividad que cuando es creada, su interfaz gráfica será el archivo activity\_main.xml. Esto lo hacemos en la línea:

******

La instrucción setContentView es la que nos permite completar nuestra actividad, ya que tenemos la parte lógica del archivo MainActivity.java y la ligamos a una interfaz gráfica, el archivo XML main\_activity.xml.

Tenemos los siguientes métodos de nuestro archivo:

******

El primer método es para crear un menú y el segundo es para que este menú funcione cuando se selecciona una opción del menú.

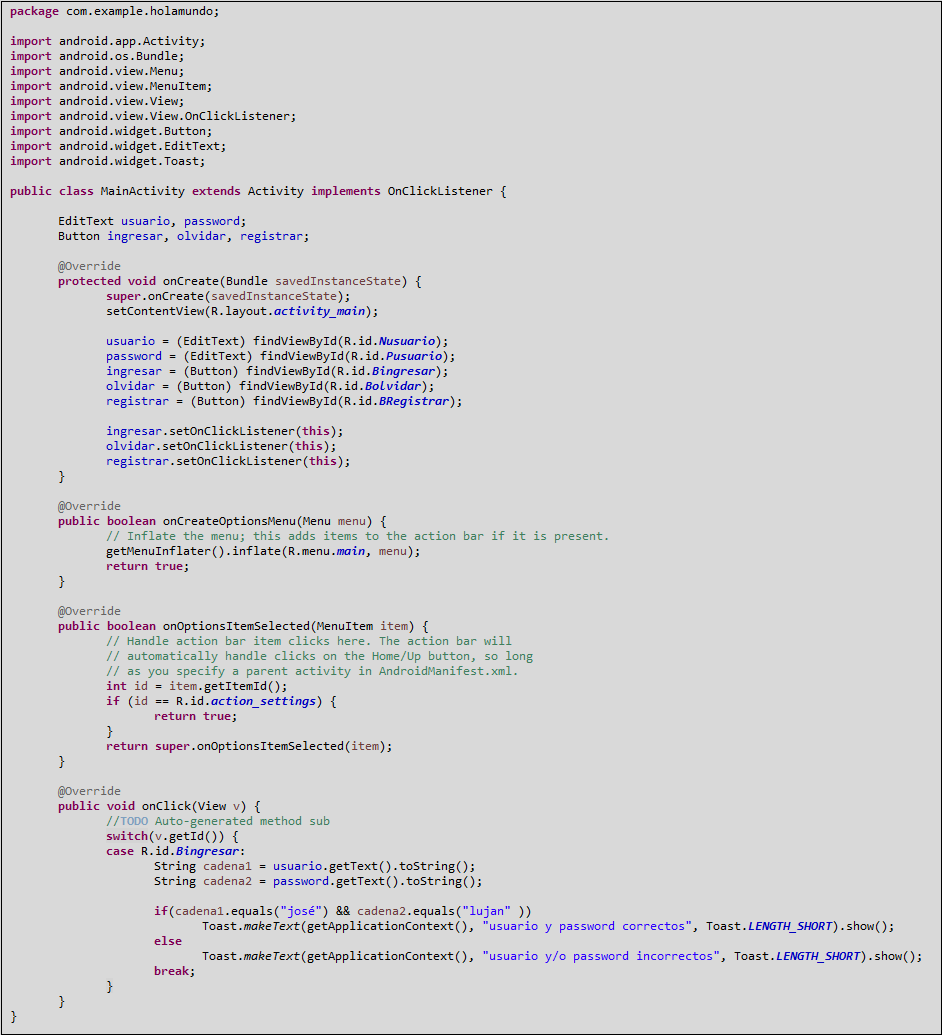
En realidad, estos métodos no los necesitamos, los podríamos eliminar y nuestra actividad seguiría funcionando, estos métodos son para tener y colocar un menú en nuestra aplicación, pero no son esenciales para su funcionamiento. En muchas ocasiones el menú se necesita de alguna forma específica por la naturaleza de la aplicación y acabamos utilizando otro tipo de menú o simplemente se elimina.

**11. INTERACCIÓN DE ELEMENTOS**

En este capítulo vamos a dar interacción a los elementos que creamos en el capítulo 8, esto lo haremos desde nuestro archivo MainActivity.java, las líneas de código que ya se han explicado anteriormente se omitirán.

Nuestra aplicación va a recibir 2 datos: un usuario y una contraseña, de los 3 botones solamente vamos a usar el botón ingresar, y al presionarlo, en el campo “usuario” el texto que se colocó debe ser “josé” y la contraseña “lujan” para que aparezca un mensaje de que es “correcto”, en caso de que alguno de los dos no coincida con “josé” o “lujan” respectivamente se mostrará que es incorrecto.

El archivo MainActivity.java es el siguiente:



**Declarar elementos en un archivo Java**

Agregaremos los elementos que deseamos manipular en código, le podemos colocar el nombre que deseemos, se recomienda declarar al inicio del archivo los elementos que se van a usar, ya que esto estructura mejor el código. Nosotros usaremos 5 elementos.

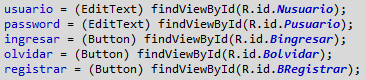
******

Siempre tenemos que colocar el mismo tipo, si en el XML estamos colocando un elemento como EditText si queremos usarlo en el archivo.java debemos crear un elemento del tipo EditText para que estos coincidan y no provoquemos un error.

**Ligar los elementos declarados en archivo Java con los del archivo XML**

Ahora vamos a unir los elementos que declaramos al principio del archivo con el elemento gráfico correspondiente del archivo main\_activity.xml, como ya comentamos tienen que ser del mismo tipo, si yo declaro un EditText en el archivo .java lo debo unir con EditText en el XML, en caso de no ser del mismo tipo marcaría un error o tendríamos que ver la forma de crear una compatibilidad de tipos.

Para ligarlos utilizaremos dos cosas, el findViewById y el ID que le asignamos en el XML, quedarían como los siguientes:

******

**Interacción con botones**

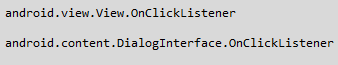
Para que los botones puedan interactuar (ser presionados y ejecutar algo) vamos a utilizar el método onclick, pero primero debemos darles la capacidad de reaccionar al ser presionados, para esto utilizamos la interfaz onClickListener.

Para usar la interfaz tenemos que añadirla al crear la class MainActivity:

******

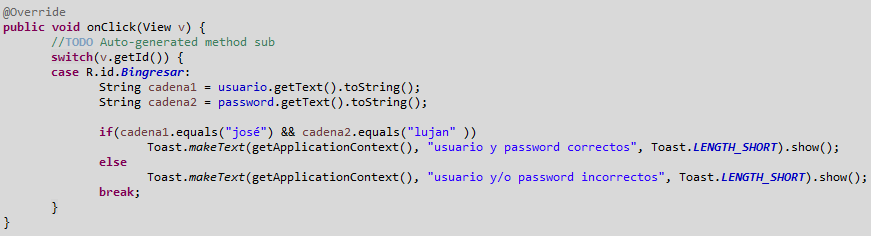
Agregando el **implements OnClickListener** habilitamos la posibilidad de que los elementos de nuestra actividad puedan ser presionados.

Al agregar el onClickListener en el import tenemos dos opciones:

******

Lo que va a funcionar correctamente para este ejercicio es el primero.

Ahora tenemos que definir nuestro método onClick (View v)

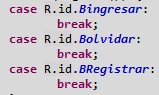
******

El método **onClick** recibe una vista **View** **v** como parámetro, que es el elemento que fue presionado o al que se hizo click.

Para poder usar el mismo onClick para todos los elementos con los que deseamos interactuar, lo que hacemos es agregar una instrucción **switch** dentro del mismo para que cada uno de los elementos realice algo diferente. Dentro del switch tenemos que preguntar el ID para saber cuál es el código que vamos a ejecutar. Para preguntar el ID utilizamos el siguiente código.

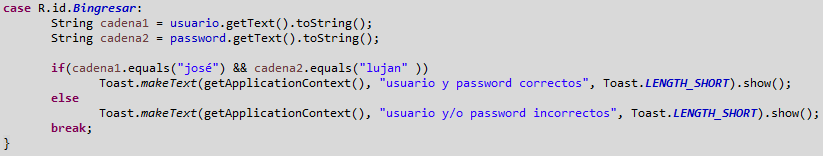
******

Ahora creamos un caso para cada uno de los ID a los que deseamos reaccionar:

******

Podemos ver que los ID son similares a los que tenemos en el archivo XML y que funciona como parte de nuestra interfaz gráfica. Entre el case y el break de cada uno de ellos debemos colocar las instrucciones que deseamos ejecutar para cada uno.

En el caso del botón Bingresar sabemos que es el que va a evaluar si usuario = “josé” y password = “lujan”. Entonces colocamos el siguiente código.

******

Creamos dos cadenas, cadena1 y cadena2, para recibir del EditText el texto y poderlo comparar, tenemos que volverlo del tipo String para poder compararlo.

Utilizamos la instrucción **if** para comparar, en Java para comparar cadenas se utiliza **equals**. Realizaremos la doble comparación en el mismo if, tanto de josé = usuario y lujar = password.

******

En caso de no ser correcto le pedimos que muestre un mensaje, para mostrar un mensaje corto al usuario podemos usar la clase Toast.

**Clase Toast**

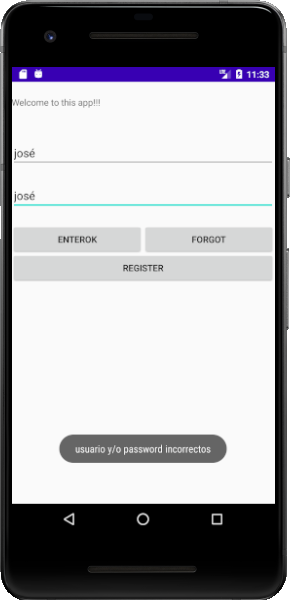
El elemento Toast nos permite crear un mensaje con duración “larga” y “corta” que se mostrará al usuario en la pantalla, su código sería el siguiente:

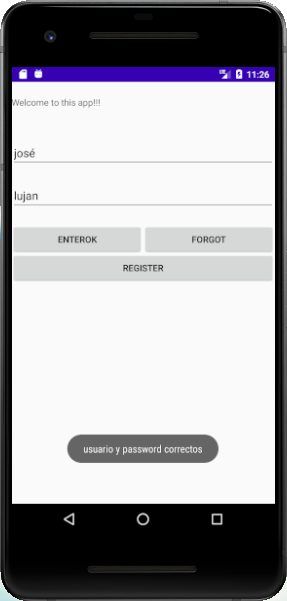
******

En estas líneas de código lo que estamos haciendo es crear el mensaje Toast que nos pide 3 parámetros.

El contexto **(getApplicationContext)**, el mensaje que vamos a mostrar **(usuario y password correctos)** y la duración del mensaje **(Toast.LENGTH\_SHORT).** Finalmente tenemos que agregar el **show** para que sea visible el mensaje, en caso de no colocarlo, el mensaje no se va a mostrar.

En caso de que no coincidan el usuario y el password lo que tenemos que hacer es mostrar un Toast similar, pero cambiando el texto por “usuario y password incorrectos”.

En caso de colocar “josé” en usuario y “lujan” en password al hacer click con el botón ingresar debería de mostrar el siguiente resultado.



**12. AGREGAR ACTIVIDADES**

Para agregar una actividad nueva tenemos que seguir estos pasos:

* Crear la interfaz gráfica (archivo XML)
* Crear la parte lógica (archivo Java)
* Registrarla en el manifest de la aplicación

Después de esos pasos ya solo tendríamos que invocarla cuando sea necesaria.

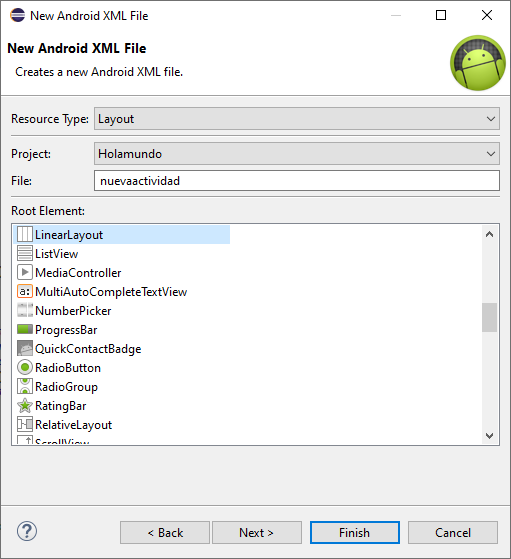
**Crear la interfaz gráfica**

Para crear la interfaz gráfica vamos a colocarnos en el directorio layout que queremos crearla, para nuestro ejemplo vamos a usar el directorio layout por defecto, el que solo lleva de nombre layout (sin prefijos). Pulsamos botón derecho y seleccionamos New, después Android XML File.

Camino: **directorio layout (click derecho) -> New –> Android XML File**

Camino: **directorio layout (click derecho) -> New –> Other -> Android XML File**

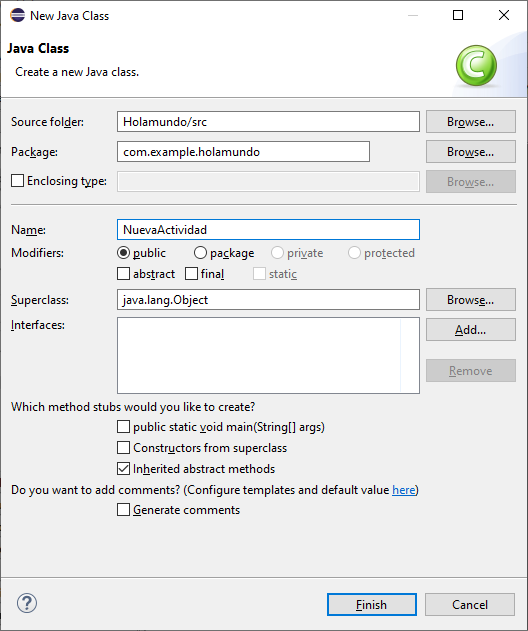
Nos mostrará una ventana como la de la siguiente imagen:

Colocamos en el campo File el nombre del archivo y en la lista de elementos que lleva el nombre de Root Element nos permite colocar el elemento principal del XML, seleccionamos el LinearLayout y hacemos click en Finish.

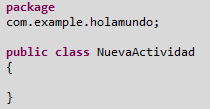
Ahora ya tenemos nuestro segundo XML que será la interfaz de la segunda actividad de nuestra aplicación.

**Crear un archivo .java de una actividad**

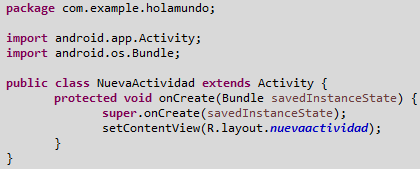
Para crear el archivo .java vamos al directorio src de nuestra aplicación, pulsamos botón derecho y seleccionamos la opción New y luego Class. Nos mostrará una ventana como la siguiente imagen:

El único campo que debemos llenar es el del nombre, en este campo estamos colocando el nombre de la clase, las clases en Java tienen ciertas normas que debemos de seguir. Para escribir el nombre se debe utilizar la notación Came|Case. Esta notación tiene como regla que no se pueden colocar espacios entre palabras y que siempre la primera letra de cada palabra debe ir en mayúscula. Las demás opciones vienen seleccionadas por defecto, en realidad es muy rara la ocasión que tendremos que modificarlas. Después de colocar el nombre hacemos click en Finish.

Ahora ya tenemos una clase nueva y que está vacía, solo contiene el siguiente código:

******

Vamos a agregar el siguiente código a la clase NuevaActividad y debería quedar de la siguiente manera:

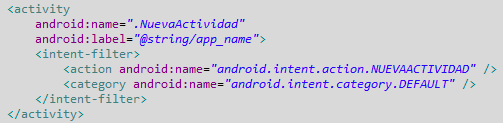
******

Si examinamos el código podemos observar que es similar al MainActivity del “Holamundo” que creamos en el capítulo 4, la diferencia es que ahora el setContentView contiene otro valor dentro, como la interfaz de este se encuentra en el archivo nuevaactividad.xml debemos de colocar el siguiente código:

******

**Registrar la actividad nueva en el manifest**

Para registrar la nueva actividad que acabamos de crear debemos de agregar el siguiente código en el manifest.

******

Podemos copiar y pegar la etiquita <activity> que ya tenía el manifest del MainActivity, pero cambiando los siguientes valores:

Propiedad android:name debemos de colocar el nombre del archivo .java que tiene la parte de código de la actividad.

******

Propiedad action de la etiqueta intent-filter: debemos de colocar el nombre del archivo XML todo en mayúsculas.

******

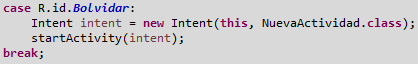
Propiedad category de la etiqueta intent-filter: debemos colocar el valor DEFAULT a todo aquella que no es la actividad principal.

******

Finalmente tenemos agregada una segunda actividad a nuestra aplicación. Para cualquier actividad que deseamos agregar en una actividad debemos de realizar los mismos pasos que acabamos de hacer.

**Abrir una actividad a partir de un botón**

Para que una actividad sea llamada a partir de presionar un botón, agregaremos el siguiente código, usaremos el botón “olvidar” de nuestra primera actividad.

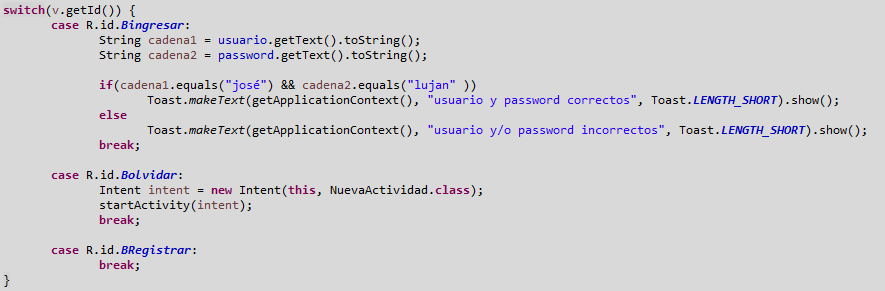
******

Lo que vamos a utilizar es una clase que se llama Intent. Esta clase sirve para invocar componentes, por ejemplo, una actividad. Para crear un intent hacemos lo siguiente:

******

En el momento de crear el intent (new intent) tenemos que pasarle dos parámetros: el contexto y la clase que vamos a mandar a llamar. En el contexto colocamos this, en la clase colocamos el nombre de la segunda actividad que creamos: NuevaActividad.class.

El código completo de nuestro switch debería de quedar como el siguiente:

******

En el momento de presionar el botón “olvidar” de nuestra primera actividad debería de mostrarnos una actividad vacía nueva, está vacía porque al instante de crearla no le colocamos ningún elemento en el XML. Se debería de ver como la siguiente imagen:



**13. TIPOS DE SCROLL**

En el mundo de la informática cuando nos referimos al término “scroll” hacemos referencia a lo que en español conocemos como desplazamiento. En las aplicaciones Android tenemos 2 tipos de desplazamiento, el horizontal y vertical.

En este capítulo vamos a crear una actividad que tenga un grupo de imágenes y dependiendo de la que seleccionemos vamos a cambiar el fondo de pantalla de nuestro teléfono.

Abordaremos varios temas en este capítulo:

* HorizontalScroolView
* ScrollView
* ImageView
* Permisos

**HorizontalScrollView**

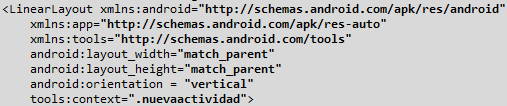
Primero vamos a crear nuestra interfaz en la que vamos aplicar a una serie de imágenes un HorizontalScroolView.

Nuestra interfaz se ve como la siguiente imagen:



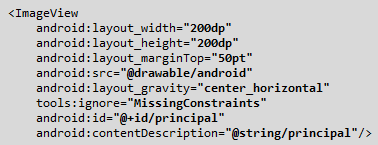
De las líneas de código que ya se han explicado anteriormente en otros capítulos se omitirá su explicación.

Tenemos el LinearLayout principal del que ya entendemos todas las líneas de código:

******

**ImageView**

Después declaramos un ImageView, este es un elemento que sirve para colocar imágenes en una aplicación Android. Tiene sus propiedades.

******

Los atributos layout\_width y layout\_height para este ejercicio están definidos en 200dp, en realidad los elementos no deberían tener un tamaño fijo, pero para este ejercicio queremos forzar una medida equitativa para todas las imágenes y asegurar que podemos verlas en la pantalla. Una aplicación que va a subirse a la tienda de aplicaciones y queremos que tenga soporte multipantalla, no debería colocar valor fijo en ningún elemento. Deberíamos de aplicar lo visto en el capítulo 7 de este libro.

En el ImageView tenemos un atributo src, este sirve para indicarle la fuente de la imagen que Android debe colocar para este elemento, esta imagen debe estar en el directorio drawable y por lo menos en una entidad. Este ImageView tiene como imagen asignada una que lleva el nombre de “android”. Esta imagen es Andy (androide verde que sirve de logo para este sistema operativo). El código es el siguiente:

******

El atributo **layout\_gravity** sirve para alinear el elemento ImageView en el centro de la pantalla de forma horizontal, podemos poner diferentes valores de alineación, centrado horizontal, centrado vertical, derecha, izquierda, arriba, entre otros. La línea de código es la siguiente:

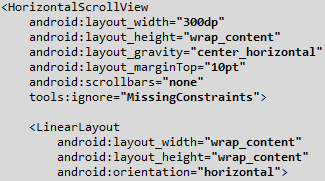
******

El ID que le asignamos a este ImageView es el texto “principal”, ya que la imagen que se esté mostrando en este contenedor en el momento de pulsar el botón “Lay background” la tomará para colocarla como “Wallpaper” del móvil.

**HorizontalScrollView**

Este elemento sirve para realizar un desplazamiento de otros elementos que colocamos en su interior y realizar desplazamientos de forma horizontal. Tendremos que colocar su ancho, su alto y tenemos la opción de colocar una scrollbar que nos sirva como referencia, en este ejemplo colocamos el valor como none porque no queremos ver la barra de desplazamiento.

Dentro del HorizontalScrollView colocamos un LinearLayout para colocar los elementos de forma horizontal; el código es el siguiente:

******

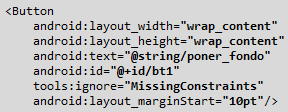
Ahora vamos a declarar las 4 imágenes que vamos a colocar para este ejercicio del desplazamiento, todas tienen un alto y ancho de 150dp, se les coloca su respectiva fuente en el atributo src y una alineación en la etiqueta layout\_gravity, además se les coloca un ID para acceder más adelante a ellas desde el código Java.

******

**Botón**

Finalmente declaramos un botón fuera del HorizontalScrollView y LinearLayout para que este no entre en el desplazamiento y sea el que en el momento de hacer click cambie el fondo de pantalla de nuestro móvil.

El código queda así:

******

**Código de la actividad**

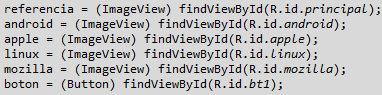


Como en todos los ejercicios del libro, se omite la explicación de las líneas repetidas de código en anteriores capítulos o ejemplos.

Primero declaramos los elementos que vamos a usar, son 5 ImageView, 1 Button y 1 int (entero)

******

A continuación, asignamos a cada elemento del archivo XML al que queremos hacer referencia desde el código Java.

******

En el evento onClick tenemos que tener 4 casos para las imágenes del ScrollViewHorizontal (android, apple, Linux, mozilla) y uno para el botón (button)

En cada uno de los casos de las imágenes le vamos asignar el “ImageView referencia” que es el principal, la misma imagen a la que se le hizo click, con la siguiente instrucción vamos a decirle que tome la fuente de la imagen desde el directorio drawable y con el nombre correspondiente de cada caso, en el supuesto de ser Android, por ejemplo:

******

La línea anterior la tenemos que hacer en cada caso, con la asignación de su respectivo recurso.

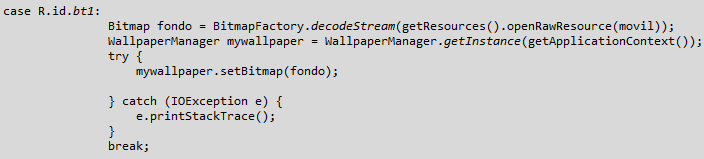
Ahora le vamos a asignar el mismo recurso del directorio drawable en cada caso (instrucción case) a la variable *movil* del tipo entero, esto es porque en el momento de hacer click al botón, este tomará el valor que contenga la variable *movil* para cambiar el fondo de nuestro móvil.

La variable *movil* siempre va a tener el valor de la última imagen a la que se le hizo click, esto se tiene que hacer para cada caso también como la línea anterior de código y el código es el siguiente:

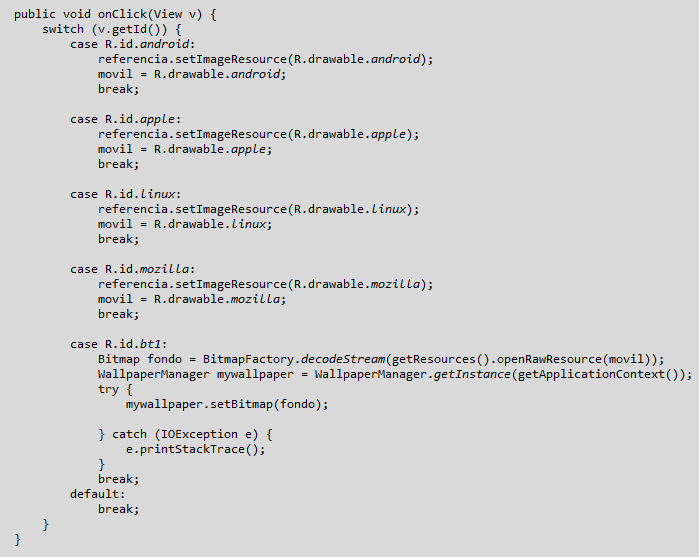
******

En el caso del botón tendremos algo diferente, ya que el botón tiene que utilizar la variable movil para obtener la imagen y esta colocarla como fondo de nuestro móvil.

Utilizamos la instrucción openRawResource y la clase Bitmap para poder colocar el fondo, ya que la instrucción setWallpaper nos solicita un Bitmap. Además, al querer colocar el fondo tendremos que crear una instrucción try/catch, esto porque debido a que interactuamos con algo que se encuentra fuera de nuestra aplicación y que pertenece al sistema operativo, tenemos que manejar los posibles errores para evitar problemas al usuario. El código es el siguiente:

******

Si hacemos todo lo anterior en el onClick, el código debe ser como el siguiente:

******

**Permisos**

Cuando estamos queriendo acceder a recursos fuera de la aplicación, lo más común es que tengamos que solicitar permisos al usuario. Los permisos son una forma de mantener un nivel de seguridad en el que el usuario puede mantener cierto control sobre lo que se hace en su dispositivo.

Los permisos los tenemos que solicitar en el archivo manifest, el usuario cuando descarga nuestra aplicación es informado de los permisos que necesita para funcionar, en el momento de descargar el usuario está aceptando los permisos, en muchos casos también se pueden controlar desde la configuración del móvil, pero el usuario habitual no configura los permisos de forma personalizada, en realidad solamente revisa los permisos cuando descarga una aplicación desde la tienda.

Para solicitar el permiso de cambio de fondo de pantalla tenemos que ir al manifest y agregar la siguiente línea:

******

Los permisos por convención se colocan debajo de la etiqueta <uses-sdk> del manifest.

Este permiso ya nos da la posibilidad de acceder a la configuración del Wallpaper y cambiarlo. Si ejecutáramos nuestra aplicación, esta se cerraría de forma inesperada ya no contamos con el permiso de acceso al Wallpaper.

**Probando la aplicación**

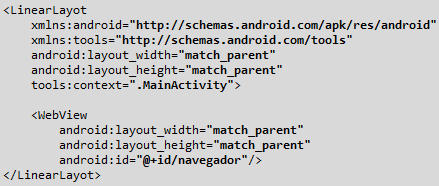
Nuestra aplicación se debe ver así:



**14. WEBVIEW**

WebView es una clase que nos proporciona Android para usarla como base y crear nuestro propio navegador o también podemos utilizarla para ver archivos html en nuestro navegador. Esta clase utiliza el motor de renderizado WebKit, incluye todas las características básicas de un navegador como zoom, búsquedas de texto, desplazamiento atrás y adelante, entro otros.

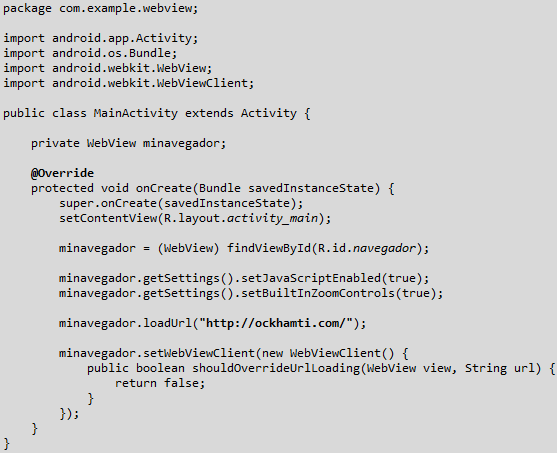
El XML que vamos a utilizar para WebView debe tener el siguiente código:

******

El LinearLayout es como el de cualquier otra aplicación de las que ya creamos. El elemento <WebView> es nuevo. Al elemento WebView de nuestro XML lo expandimos al tamaño total de nuestra pantalla y para eso los atributos layout\_width y layout\_height los colocamos con el valor match\_parent.

Finalmente le colocamos un ID que lleva de nombre “navegador” para poder acceder a este elemento desde el archivo Java.

La actividad tiene el siguiente código:

******

Declaramos un elemento WebView:

******

Enlazamos el elemento minavegador que ya declaramos con el elemento WebView del elemento XML:

******

Habilitaremos dos de las propiedades de muchas que podemos manipular de un WebView, en la primera línea habilitamos el uso de Javascript en nuestro elemento minavegador y en la siguiente línea estamos habilitando la propiedad que no permite hacer Zoom, en caso de no querer permitir su uso cambiaríamos el valor que encontramos ahora en “true” por “false”, el código es el siguiente:

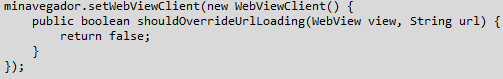
******

Finalmente colocaremos el código que cargue una web en nuestro elemento WebView, para esto utilizamos la instrucción loadUrl y colocar una url entre comillas, quedaría como el siguiente código:

******

Hasta ahora el código que hemos descrito funciona, el problema es que si en la web que cargamos tenemos enlaces como normalmente tiene la mayoría, al hacer click nos pediría abrir el navegador nativo de nuestro dispositivo para continuar navegando, esto ocasiona doble carga de trabajo y además afecta a la experiencia del usuario.

Para que al navegador sobre una web se mantenga en el mismo WebView que hemos creado debemos utilizar la instrucción shouldOverrideUrlLoading, esta instrucción lo que hará es que siempre que va a cargar una URL sobre el mismo WebView, pero para usarla utilizamos la instrucción setWebViewClient, con esto logramos mantenerlo como un “Cliente” del WebView, el código debe ser el siguiente:

******

Para nuestro WebView siempre pueda cargar cualquier URL accediendo a internet, debemos colocar el permiso de acceso a internet en nuestro archivo manifest, recordemos que la recomendación para colocar los permisos siempre es entre la etiqueta uses-sdk y Application.

******

Así es como finalmente se ve nuestro WebView cargando la URL: <http://ockhamti.com>, pero se le puede colocar cualquier URL y por recomendación sin utilizar el “www” al principio de ella.



**15. WEBAPP VS NATIVO**

En este capítulo nos dedicamos a definir los conceptos de WebApp y aplicación nativa, hablaremos de las características, ventajas, desventajas y el estado actual del desarrollo en ambos lados.

**Webapp**

En los últimos años con el crecimiento del desarrollo móvil la creación de aplicaciones para estos creció de manera exponencial, además de que la gente está adoptando esta tecnología con tanta velocidad y facilidad que representa un mercado bastante atractivo para cualquier empresa, la tendencia sigue aumentando y esto significa que en los años siguientes seguirá creciendo.

El mayor problema de las empresas al ingresar en el mundo de las aplicaciones es que tenemos varios sistemas operativos en el mercado, iOS, Blackberry, FirefoxOS, Tizen, Android, Windows Phone, entre otros. Al ser un área relativamente nueva para las empresas, la inversión que tienen destinada no es del todo suficiente para cubrir los mercados, en algunas ocasiones de manera inteligente optan por estudiar cuál es la preferencia de sus clientes y deciden realizar una aplicación solamente para estos. Otro problema reside también que a diferencia del desarrollo de sitios web (backend y frontend) el desarrollo móvil es relativamente nuevo y es más caro muchas veces un proyecto de esta naturaleza.

Este problema y la evolución de tecnologías como HTML5, CSS y Javascript llevaron a planear una nueva solución que es el “Desarrollo multiplataforma” o las famosas “Webapps”.

En un principio una webapp es una aplicación que funciona independientemente del sistema operativo que lo ejecute, es decir, que si tenemos Android debe funcionar, si tenemos iOS, Blackberry, Windows Phone o cualquier otro sistema operativo su funcionamiento debe ser el mismo.

La mayoría de las webapps concentran su fuerza en el uso de HTML, CSS y Javascript dentro de un navegador, ya que cualquier sistema operativo se conecta a internet y cuenta con un navegador nativo, así habilita el uso de una webapp en cualquier navegador. Esto en teoría reduce la inversión ya que al utilizar la misma aplicación significa que no tenemos que pagar un desarrollo para cada sistema operativo y reduce los costos sustancialmente. Muchas empresas ven esto como una solución variable al comenzar en el mundo del desarrollo móvil.

Actualmente existen muchos frameworks que nos ayudan a mejorar y agilizar el desarrollo de una webapp, como, por ejemplo: sencha, phonegap, Titanium, JqueryMobile, entre otros. Todos usan por parte de la programación Javascript y por parte de diseño html y css.

Muchos de estos frameworks son de pago o se tiene que pagar una licencia para su uso, que van desde los 100 dólares hasta los 25000 dólares. Depende del nivel de soporte, librerías y utilidades que desee implementar el desarrollador en su proyecto.

**Desarrollo nativo**

El término desarrollo nativo hace referencia a crear una aplicación en el lenguaje nativo de cada sistema operativo, por ejemplo: Android está basado en Java, iOS en Objective-C, entonces si creamos una aplicación nativa quiere decir que estamos creando una aplicación en el lenguaje de programación en el que está basado el sistema operativo.

Al necesitar un desarrollador especialista en un lenguaje de programación normalmente eleva el costo del desarrollo nativo, pero siempre al utilizar la tecnología nativa. Por ejemplo, cuando Android sufre una actualización solo es cuestión de horas o días para tener acceso a las nuevas APIs, utilidades o librerías, esto es porque siempre que se anuncian cambios es para anunciar a las comunidades de desarrolladores las nuevas formas, metodologías y técnicas de programación que se van a utilizar.

**Webapp vs Nativo**

**Ventajas de una webapp**

* El desarrollo de la mayoría de webapps utiliza HTML, CSS y Javascript, así que tendremos la posibilidad más amplia de encontrar profesionales que puedan trabajar en ello.
* El costo es relativamente más bajo por la relación oferta-demanda en los profesionales
* El tiempo de desarrollo debería de ser más rápido debido a la no especialización a nivel de hardware, APIs, con los dispositivos

**Desventajas de una webapp**

* Difícil acceso al hardware del dispositivo, por ejemplo, la cámara, almacenamiento interno de datos, entre otros
* Casi nulo acceso a los sensores y a los datos que arrojan, el navegador aún no cuenta con la posibilidad en la mayoría de los casos y con la velocidad que aparecen nuevos dispositivos, parece que va dos pasos por detrás.
* Al ser una webapp el rendimiento en muchos casos no es el óptimo ya que estamos procesando el navegador en donde se monta la webapp y además el procesamiento que realiza la webapp en el navegador.
* Usabilidad, al ser una aplicación para varios sistemas operativos la iconografía difiere entre ellos, además de contar con diferentes botones y significados, por ejemplo, en Android contamos con un botón atrás nativo que todos los dispositivos Android tienen, esta misma aplicación en iOS (el procedimiento para atrás o menú) lo tendría que hacer de otra forma. Esto afecta de manera importante a la usabilidad para el usuario.

**Ventajas de desarrollo nativo**

* Acceso completo al hardware de cada dispositivo
* Rendimiento óptimo dependiendo de la programación de la aplicación
* Usabilidad nativa para el usuario en las aplicaciones si esta respeta los estándares dictados por los propietarios

**Desventajas de desarrollo nativo**

* Más costoso
* Más lento por el nivel de especialización requerido por el desarrollador

**Opinión del autor**

Antes de leer lo siguiente quiere comentar que no deseo cambiar la opinión de nadie, cada uno de nosotros es un mundo y eso es lo que enriquece la mayoría de los proyectos; respeto cualquier opinión.

Comparo con ustedes unos pequeños párrafos conforme a mi mucha o poca experiencia en el mundo del desarrollo de las aplicaciones.

Para mí como desarrollador lo nativo lleva ventaja por mucho sobre las webapps, ya que como desarrolladores nos permiten un acceso total al sistema operativo, al hardware y a los sensores. Podemos obtener ventajas de accesibilidad a las aplicaciones que creamos si implementamos estas tecnologías.

Sumándole también la ventaja de que en cuestión de tiempo es el mínimo lo que tenemos que esperar, las actualizaciones las tenemos de primera mano al liberarlas Google en el caso de Android.

El especializarse en una tecnología conlleva un mejor pago sobre el desarrollo, en la actualidad la necesidad de aplicaciones con calidad es más alta por mucho a la cantidad de profesionales que pueden abastecer esta especialidad, el saber que la preparación nos puede abrir oportunidades en un mercado con gran crecimiento y bastante bien posicionado económicamente, como empleado o empresa, para mí esta buena razón es un gran motivo.

**16. DIBUJOS**

Uno de los temas más complejos al trabajar en móviles es la parte de gráficos y animaciones. Esto es debido a las limitaciones, ya que, aunque los teléfonos móviles actuales cuentan con recursos muy elevados y similares a los de un PC, muchos siguen teniendo límites en el momento de trabajar y procesar los gráficos. Como vemos es un tema complicado ya que además de hacerlo tenemos que pensar en la mejor forma de hacerlo. En este capítulo trabajaremos con una forma básica de realizar dibujos, así que estos métodos no nos sirven para realizar algún videojuego, pero sí para entender y comenzar a analizar los conocimientos necesarios y sus límites.

La clase principal que vamos a usar es Canvas. En muchos lenguajes de programación no solamente en Java se utiliza una clase u objeto llamado “canvas” y casi siempre funciona de la misma forma, aunque sí que tiene excepciones. Canvas es una clase que hace referencia a una superficie en donde se puede dibujar, se disponen de métodos que nos permiten dibujar de manera sencilla líneas, curvas o figuras predefinidas como, por ejemplo: círculo, cuadrado, rectángulo, triángulo, óvalo, entre otros.

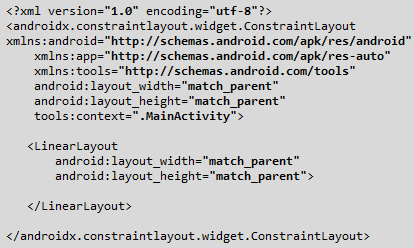
Para dibujar en Android, además de necesitar el canvas, utilizamos un pincel que llamamos Paint y en este definimos por ejemplo el color con el que dibujaremos, el grosor, la transparencia u otros efectos.

Nuestro lienzo (canvas) será toda la pantalla de la aplicación, en él podremos dibujar colocando el dedo y arrastrando para formar una figura.

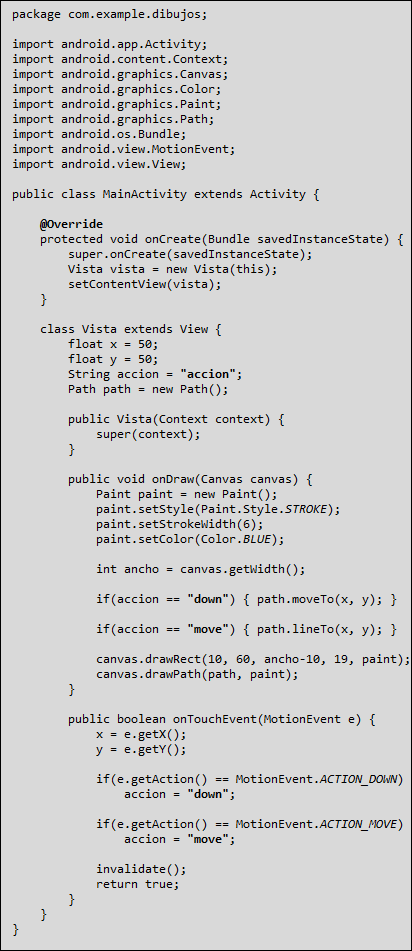
Además, dibujaremos un rectángulo en la parte superior para usar una figura predefinida de todas las que tenemos disponibles.

La interfaz gráfica es muy básica, en realidad solo tendremos un LinearLayout básico sin nada dentro, todo lo haremos en el código,

Este es el código del main\_activity.xml

******

El archivo MainActivity.java tiene este código:



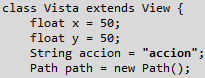
Declaramos la actividad principal y heredamos de Activity.

******

Creamos un objeto de clase Vista que declaramos más adelante, indicamos que la Vista de esta actividad es “vista” que acabamos de crear:

******

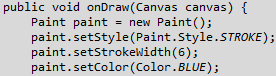
Creamos una clase que tendrá el nombre “Vista” y hacemos herencia de View. Declaramos dos elementos que serán las coordenadas que utilizaremos para dibujar, las llamamos “X” e “Y”. Creamos una cadena que se llamará “acción” que utilizaremos para saber si está sobre la pantalla el dedo del usuario o en movimiento sobre la misma. Creamos además un elemento que viene de la clase Path ya que esta nos permite definir un trazo a partir de curvas o líneas. Una vez que esté definido podremos después dibujarlo sobre el canvas:

******

Creamos el constructor de nuestra clase, y le decimos que lo tome del super:

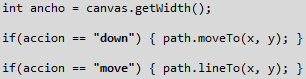
******

Utilizamos el método onDraw que es el que se encarga de dibujar. Creamos nuestro pincel con la clase Paint para indicarle el grueso, el estilo y el color de lo que vamos a dibujar.

******

Creamos una variable “ancho”, esta nos sirve para saber el ancho del espacio que tenemos para dibujar y que utilizaremos para dibujar el rectángulo más adelante.

Creamos 2 condiciones, una para saber si está abajo el dedo y otra para saber si está en movimiento y dibujar:

******

Creamos un rectángulo con draw.Rect y lo dibujamos utilizando el pincel que definimos anteriormente:

******

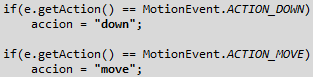
Le indicamos al canvas que dibujaremos lo que sucedió con las condiciones:

******

Utilizamos onTouchEvent que nos permite detectar cuándo colocamos el dedo, pero además el movimiento que hacemos presionando la pantalla, le decimos que del evento que se genere nos pase las coordenadas “X” e “Y” para saber en dónde sucede y poder tomarlas como referencia para dibujar.

******

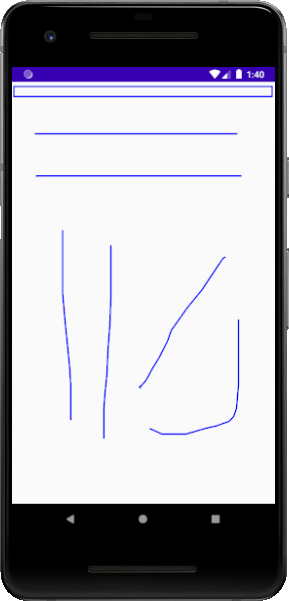
Utilizamos una condición para saber cuándo sucede un movimiento y cambiar el contenido de la cadena de texto que se llama “accion”, tenemos dos opciones: si solamente se coloca el dedo en la pantalla, cambiamos el texto por “down”; la otra opción es cuando se hace un movimiento con el dedo, entonces cambiamos el texto por “move”:

******

Agregamos al final invalidate, este nos sirve para indicarle al hilo de la interfaz gráfica que se redibuje lo más pronto posible, finalmente devolvemos el valor de true:

******

La pantalla inicial dibuja el rectángulo que le indicamos inicialmente:



En el momento de presionar sobre la pantalla del móvil se dibuja una línea sobre el trazo que realizamos.

**17. ANIMACIONES**

Para crear animaciones tenemos diferentes métodos en Android, en este capítulo aprovecharemos los archivos XML para realizarse.

Una forma sencilla para hacer animaciones es aprovecharnos de archivos que Android nos proporciona para crearlas y definir los movimientos que deseamos realizar. En este ejercicio vamos a utilizar la imagen por defecto que tenemos en nuestra estructura básica del “holamundo” y cuando presionemos un botón hacer un giro de 360 grados con la imagen.

La interfaz gráfica tendrá un ImageView que será el elemento al que le aplicaremos la animación y abajo declaramos un botón, estos dos elementos ya los hemos declarado en capítulos anteriores en este libro.



Ahora vamos a crear el archivo “rotar.xml”, primero vamos al directorio “res” pulsamos botón derecho, seleccionamos new, luego folder y colocamos el nombre “anim”.

Camino: **directorio res (click derecho) -> New -> Folder -> nombre del directorio: anim**

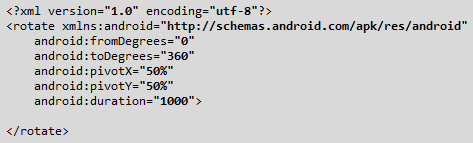
Ahora vamos a crear un archivo con el nombre rotar.xml dentro del directorio que acabamos de crear:

Camino: **directorio anim (click derecho) -> New -> Android XML File -> seleccionamos rotate y colocamos el nombre rotate.xml**

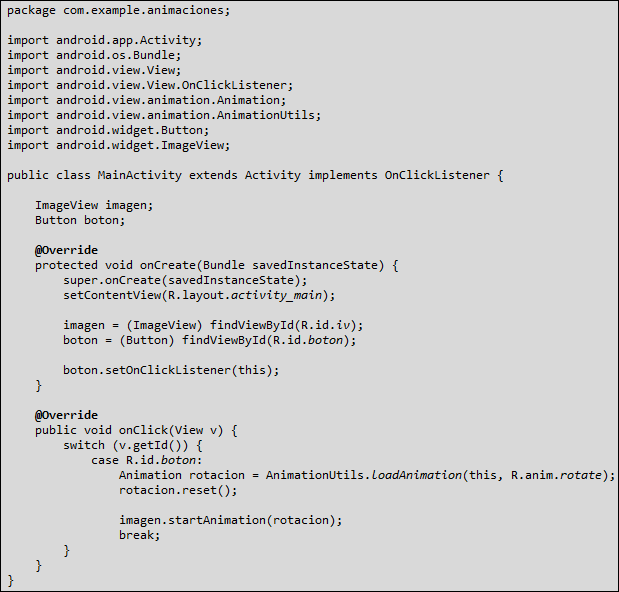
Dentro de la etiqueta rotate utilizaremos el atributo FromDegress y toDegress para indicar los grados de la rotación, de 0 a 360 grados, es decir, un giro completo.

Los atributos pivotX y pivotY sirven para indicar el punto de pivote para el giro. Al indicarle que es 50% en ambos casos le estamos diciendo que tome como punto de referencia para el giro el centro de la imagen, ya que de su largo en el eje de la “X” le decimos que sea el 50% y lo mismo en el de la “Y”. En el atributo duration le estamos indicando la duración en milisegundos.

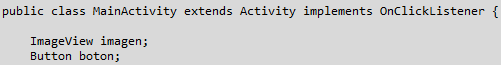
Tendremos el siguiente código en el archivo:

******

El código del MainActivity.java es el siguiente:



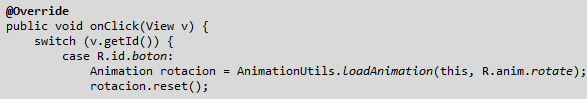
Crearemos la clase principal y heredamos de la clase Activity, además implementaremos el OnClickListener, declaramos un elemento para la imagen y otro para el botón.

******

Enlazamos los elementos imagen y botón con los que tenemos en el XML y además le habilitamos al botón el OnClickListener.

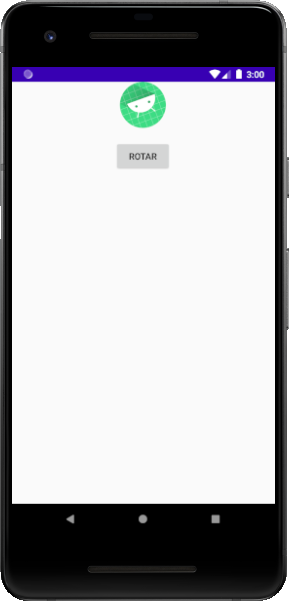
******

Declaramos el onClick en el que colocamos un switch y un caso para el botón. Dentro del caso del botón utilizamos la clase Animation para crear un elemento que llamaremos “rotación”. A continuación, a este le indicamos que cargue la animación que declaramos en el recurso rotate.xml y después aplicamos reset:

******

Para aplicar las instrucciones de la animación a un elemento, colocamos el nombre del evento y cargamos la animación:

******

Nuestra aplicación se ve así al inicio:

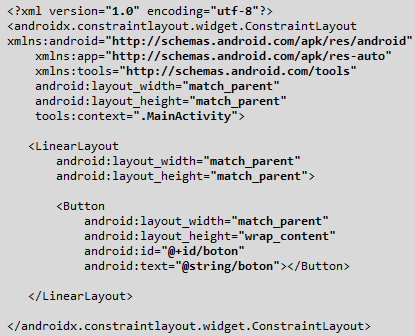
**18. CUADRO DE DIÁLOGOS**

En Android tenemos una forma muy común de comunicarnos con el usuario de forma dinámica y rápida, esta forma son los cuadros de diálogos. En ellos podemos colocar pequeños mensajes para el usuario para transmitirle un mensaje, además de colocar texto podemos colocar botones, imágenes y otros elementos, siempre va a depender de las necesidades de la aplicación.

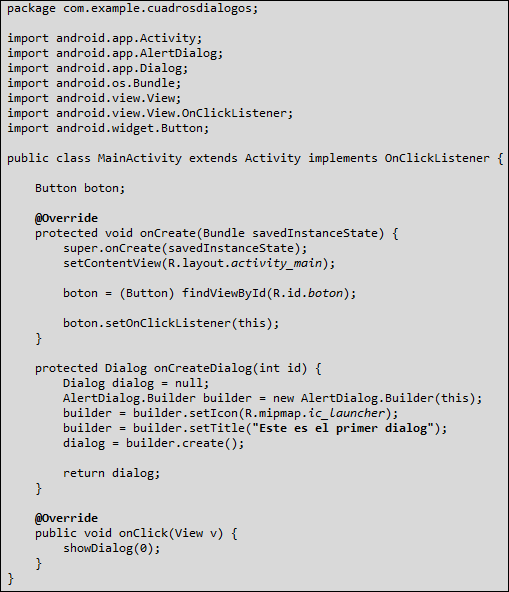
**Cuadro de diálogo simple**

Vamos a crear primero una interfaz que contenga un botón, este botón cada vez que se le haga click va a ser el encargado de mostrar el cuadro de diálogo.

Primero creamos un LinearLayout, después colocamos dentro un botón del ancho de la pantalla y con una cadena de texto cualquiera, en el caso del ejercicio el texto que colocamos dentro es “Dialog”. El XML contiene este código:

******

El código de la actividad es el siguiente:



Creamos la clase de la actividad e implementamos el OnClickListener:

******

Declaramos un elemento button:

******

Enlazamos el elemento del XML al botón que ya declaramos en Java y habilitamos el OnClickListener:

******

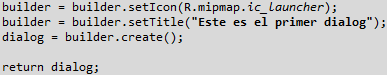
Declaramos el método onCreateDialog en el que vamos a crear el cuadro de diálogo, utilizaremos la clase Dialog. Primero creamos un Dialog, pero le colocamos el valor de null ya que por ahora no lo utilizaremos.

******

Con la siguiente línea creamos un builder, nos pedirá un contexto para crearlo y le colocaremos this:

******

Al elemento builder que acabamos de crear le podremos configurar y colocar diferentes opciones, en la instrucción setIcon le colocamos un icono con la imagen del ic\_launcher que nos proporciona Android por defecto. Además, le colocamos un título con la instrucción setTitle y finalmente en la última línea utilizamos el Dialog que creamos anteriormente y lo combinamos con el builder para crear el cuadro de diálogo. Al final de todo devolvemos el elemento dialog que acabamos de crear.

******

Para que el cuadro de diálogo se pueda mostrar tenemos que colocar en el onClick la instrucción showDialog (0), ya que esta va a mostrar el cuadro de diálogo hasta que el usuario presione atrás o sea presionada un área que no sea el cuadro de diálogo de la pantalla.

******

La interfaz se ve de la siguiente manera: