Manual de Administrador

**Programar apps para Android**

Antes de programar un programa para android, es necesario aclarar un par de cosas

Herramientas de desarrollo de Intel HTML5 están diseñados para ayudar a los desarrolladores Web móviles desarrollar, probar y desplegar aplicaciones a tantas plataformas como sea posible. Uso de tecnologías web como HTML5, JavaScript \*, \* y CSS3 , estas herramientas le permitirán no sólo crear y probar aplicaciones Web móviles , sino también aplicaciones nativas híbridos.

El propósito de este documento es demostrar a los nuevos usuarios cómo crear una aplicación híbrida nativa desde cero. Se expondrá cómo obtener el Intel® XDK , cómo codificar una aplicación híbrida nativa simple, cómo probarla en el dispositivo y, finalmente, cómo utilizar App Dev Center de Intel para construir una aplicación binaria .

**I.- ¿Cómo se estructurará esta saga?**

A diferencia de otras sagas de programación que hemos hecho en el pasado, en las cuales veíamos varios contenidos teóricos y finalmente un ejercicio, en esta el enfoque será netamente práctico, ya que creemos que la mejor forma de aprender a crear apps es *"metiendo mano"* y programando, probando código y features, intentando hacer cosas. De todas maneras no se preocupen, dentro de cada número, y como es nuestra costumbre, intentaremos explicar de la mejor manera posible cada punto, para que así cada cosa que hagamos quede lo más clara posible.

## II.- ¿Es necesario saber programar? ¿En qué lenguaje?

Sí, es necesario saber programar para poder seguir esta saga, aunque no a nivel avanzado ni experto. Si es ideal haber escrito código antes para poder tener mejores referencias de lo que haremos y poder entender todo con mayor facilidad. De todas maneras, como mencionamos en el punto anterior, intentaremos hacer las explicaciones lo más claras posibles, para que así quienes no han programado y lean estos artículos, puedan hasta cierto punto entender lo que estamos haciendo e incluso motivarse a entrar de lleno.

En cuanto a lenguajes, lo ideal es haber programado anteriormente en Java, el lenguaje en el que desarrollan las aplicaciones nativas de Android. No obstante, si nunca han escrito código en este lenguaje, pero tienen conocimientos en otros orientados a objetos, es probable que puedan relacionar bien los contenidos y tener las nociones necesarias para partir desde ahí.

## III.- Momento, ¿aplicaciones nativas?

Si, esta saga cubrirá el desarrollo de aplicaciones para Androide de forma nativa. Actualmente hay una serie de aplicaciones y DreamWorks que permiten crear aplicaciones móviles multiplataforma  (Androide, iOS, BlackBerry, Windows Phone, etc.)  Utilizando una sola tecnología de desarrollo (habitualmente Javascript o algún otro lenguaje). En este caso no las cubriremos, pero si son una alternativa a tener en cuenta, considerando las ventajas y desventajas que presenta cada una de ellas.

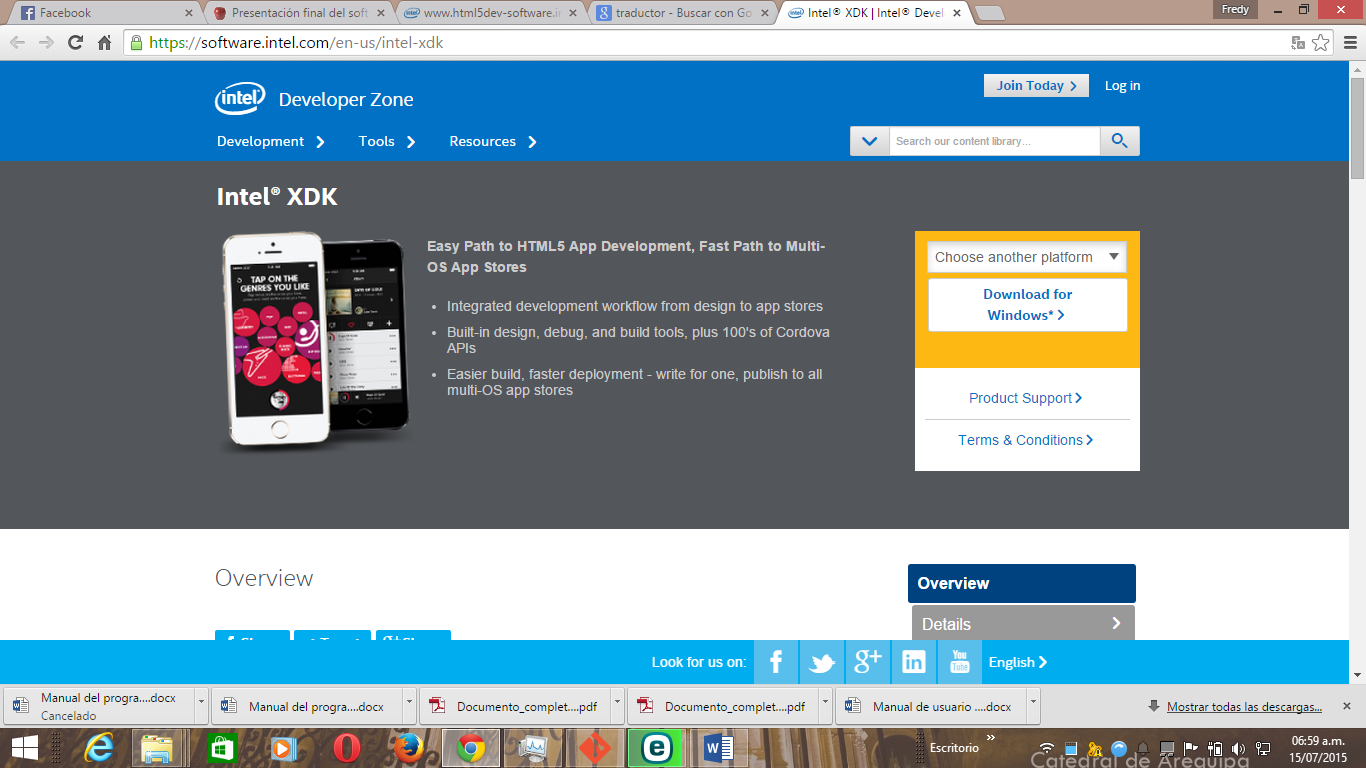
## IV.- ¿Es esta la guía definitiva para crear apps en Android?

Como hemos intentado decirles siempre, en CLH no somos expertos ni nos consideramos gurús en nada, especialmente en programación. Solo intentaremos llevarles una saga paso a paso basada en nuestra experiencia de aprendizaje de desarrollo de apps móviles. Muchos de los casos presentados acá serán en base a mi experiencia y en base a los escenarios y alternativas que me ha tocado enfrentar. En mi caso particular, partí desarrollando hace no mucho tiempo, por lo que hay bastante de lo que presentaremos acá que iremos descubriendo en conjunto. Por ello, si en cualquier momento detectan un error o alguna alternativa mejor para llevar a cabo algo.

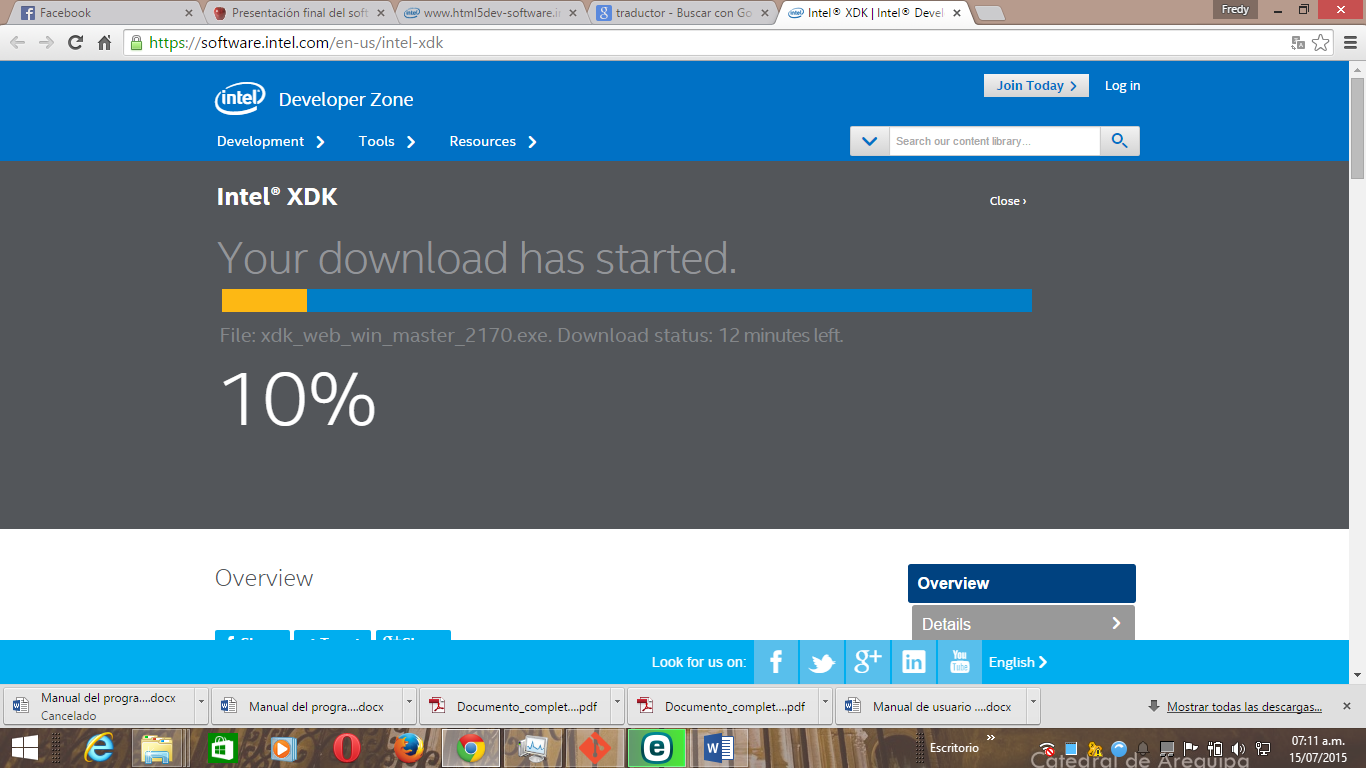
## 1.- Descarga del Intel XDK

La forma más fácil para empezar es crear una aplicación inmediata. Intel® hace las cosas más fáciles, proporcionando una camino directo a la creación de una aplicación, obtener las herramientas, y para llegar al trabajo. Utilizando el Google Chrome Navegador, los usuarios de la marca nuevos puede hacerlo bien en las herramientas directamente del Desarrollo Intel® HTML5.

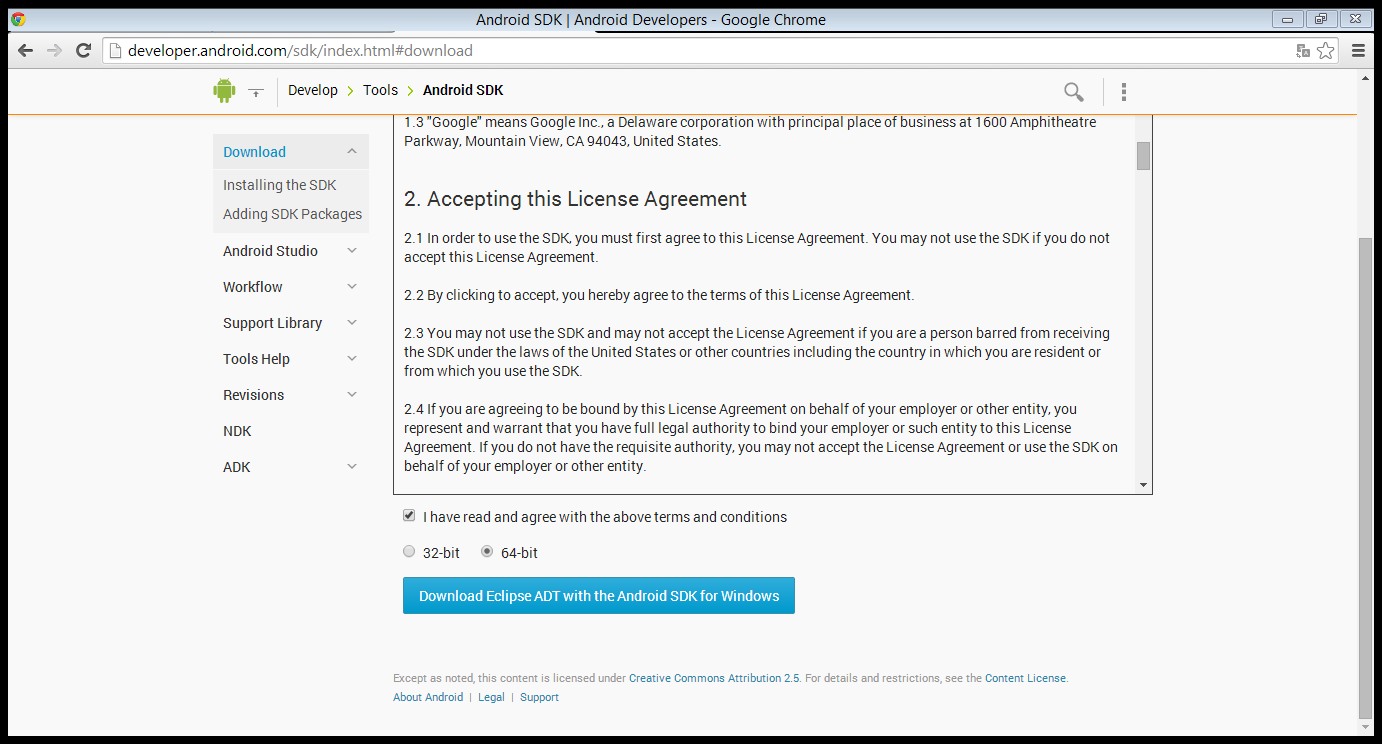
Para descargar, vamos a [https://software.intel.com/en-us/intel-xdk](https://software.intel.com/en-us/intel-xdk ) y verán en los textos de la parte derecho, un vínculo que dice **"Download for Windows"**, la plataforma se puede seleccionar **“Choose another platform”** según requiere el usuario. Hacemos click ahí para descargar:



Luego de ello, nos llevará a la pantalla de descargas, en donde automáticamente reconocerá nuestro Sistema operativo y empieza descargar para nuestra plataforma, la descarga puede demorar según la velocidad del internet.



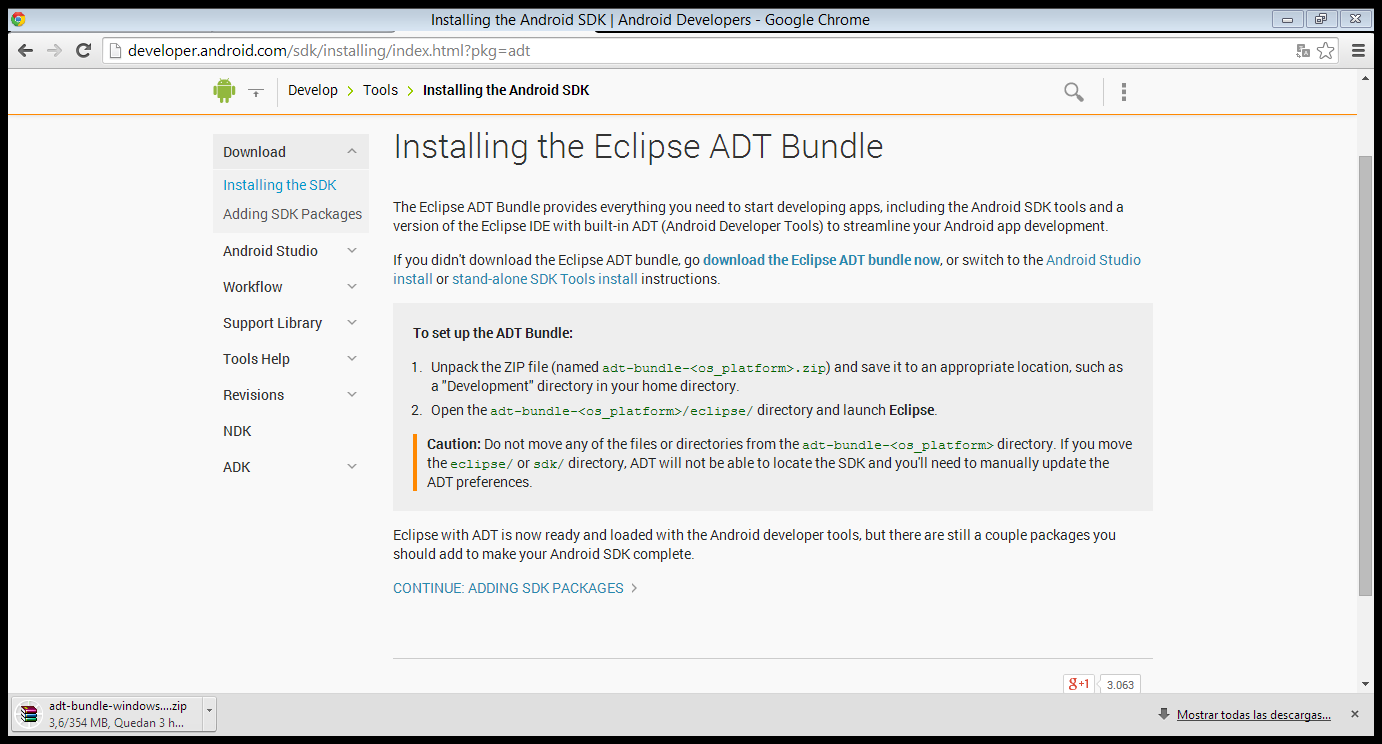
Cuando haya terminado se busca el instalador en nuestra PC , por defecto se guarda en la carpeta de descargas de nuestra pc.



Al aceptar los términos y presionar el botón **"Download the SDK ADT Bundle for (tu S.O)"**, se iniciará la descarga de un archivo .zip de aproximadamente 400mb, el cual contiene todo lo necesario para empezar a desarrollar, es decir:

* El IDE para desarrollar (en este caso Eclipse)
* El SDK que contiene todas las herramientas y clases necesarias para poder crear las apps.

Además, nos llevará a la siguiente pantalla de confirmación:



**2.- Instalación**

La instalación del SDK de Android es bastante sencilla, ya que para empezar a desarrollar, solo debemos descomprimir el archivo que acabamos de descargar y estamos listos. Así que doble click y descompriman el contenido en la ubicación que más les acomode *(su directorio personal es un buen lugar)* y tendrán un directorio ahí de nombre parecido a **"adt-bundle-su sistema-x86\_64"**, el cual a su vez contendrá 2 directorios más: **"eclipse"** y **"sdk".**

El directorio **"eclipse"**, contiene el IDE *(y sus respectivos archivos)*, con el que desarrollemos. Desde las primeras versiones de Android, ha venido integrado a este IDE, por su facilidad e integración para trabajar con Java. Si bien recientemente Google anunció el lanzamiento de Android Studio, el cual utiliza otro IDE para desarrollar, se encuentra aún en fases preliminares, por lo que la gran mayoría de los desarrolladores utiliza Eclipse, que es con el que trabajaremos en esta saga. Es bastante sencillo porque además el SDK viene directamente integrado por lo que es llegar y empezar a trabajar.

Además es factible instalar el SDK en otros IDE's por si Eclipse no es su preferido. [Acá hay indicaciones](http://developer.android.com/sdk/installing/index.html) bastante detalladas de como realizar esta tarea: http://developer.android.com/sdk/installing/index.html

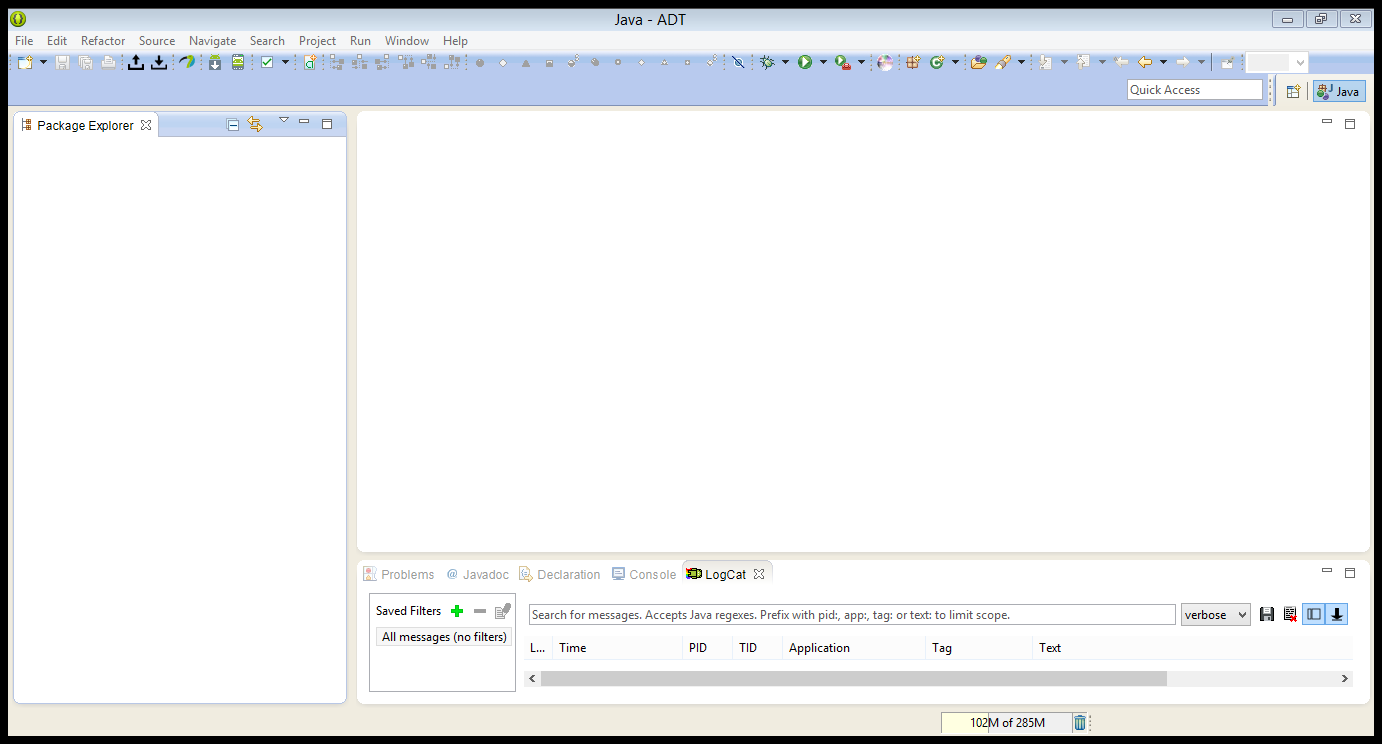
**Importante:** Si bien el archivo descargado pueden descomprimirlo y ubicarlo en cualquier parte de su equipo, es muy importante **no mover los directorios ni archivos que están en su interior**, ya que para futuras actualizaciones se requiere respetar esa estructura para poder ubicar las preferencias del SDK y modificarlas de ser necesario.

En próximos números, a medida que vayamos desarrollando y probando nuevos features, requeriremos de distintas utilidades que deberemos incorporar al SDK. Esto lo haremos con una utilidad llamada **SDK Manager**, el cual veremos más adelante en detalle, que además nos permite actualizar la versión de Android con la que estaremos trabajando.

Ahora, si vamos a nuestro directorio que acabamos de descomprimir, y ahí a **eclipse/eclipse.app o eclipse.exe**dependiendo de la plataforma, ejecutaremos el IDE y nos pedirá definir una ruta para nuestro espacio de trabajo. En este directorio que definamos se crearán y guardarán automáticamente todos los proyectos que creemos en eclipse. Pueden elegir la ruta que quieran y/o la que más les acomode. En nuestro caso, dejamos todo en un mismo directorio, el bundle de Android Development Tools y el espacio de trabajo, por lo que queda una estructura similar a esta:

* Android
  + adt-bundle
  + workspace

Y al abrir Eclipse, podrán ver algo como esto, dependiendo de su versión y plataforma: Y con esto, ahora empezaremos a trabajar.

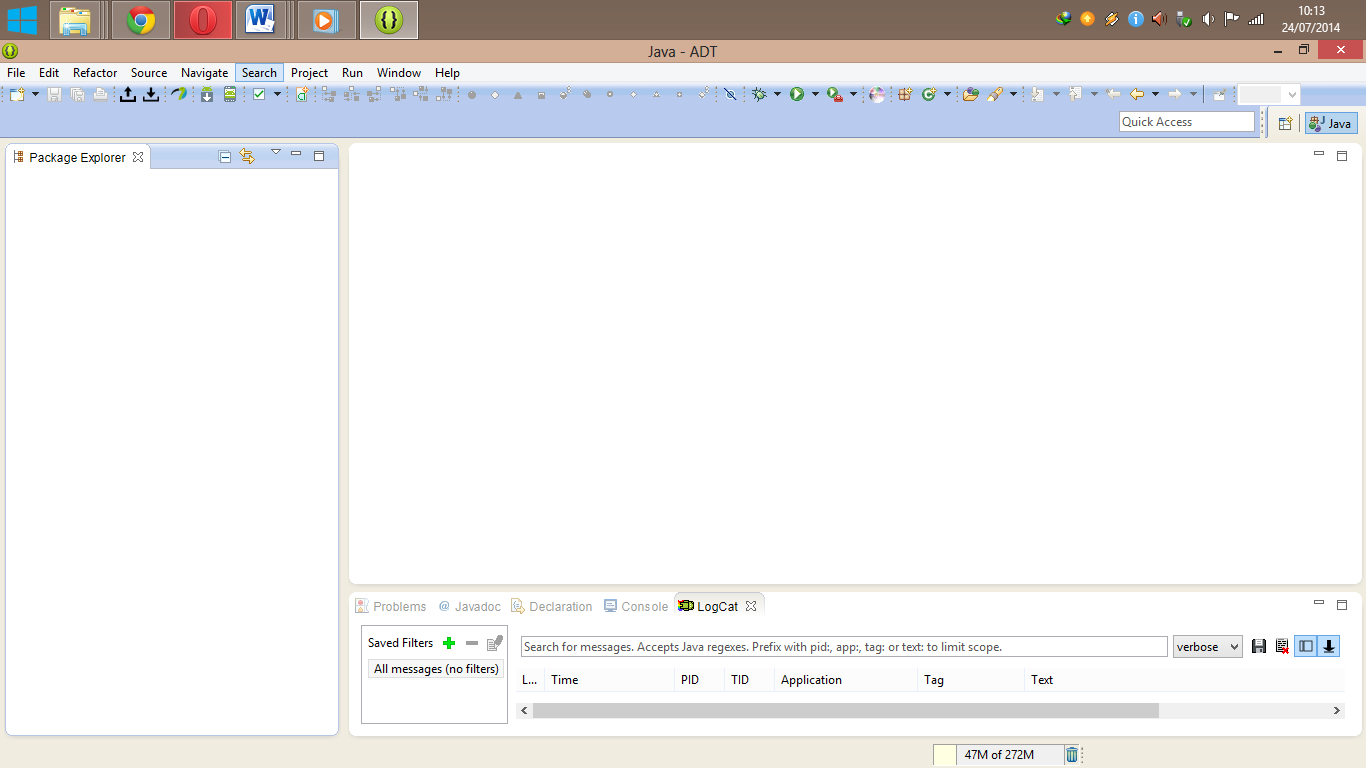


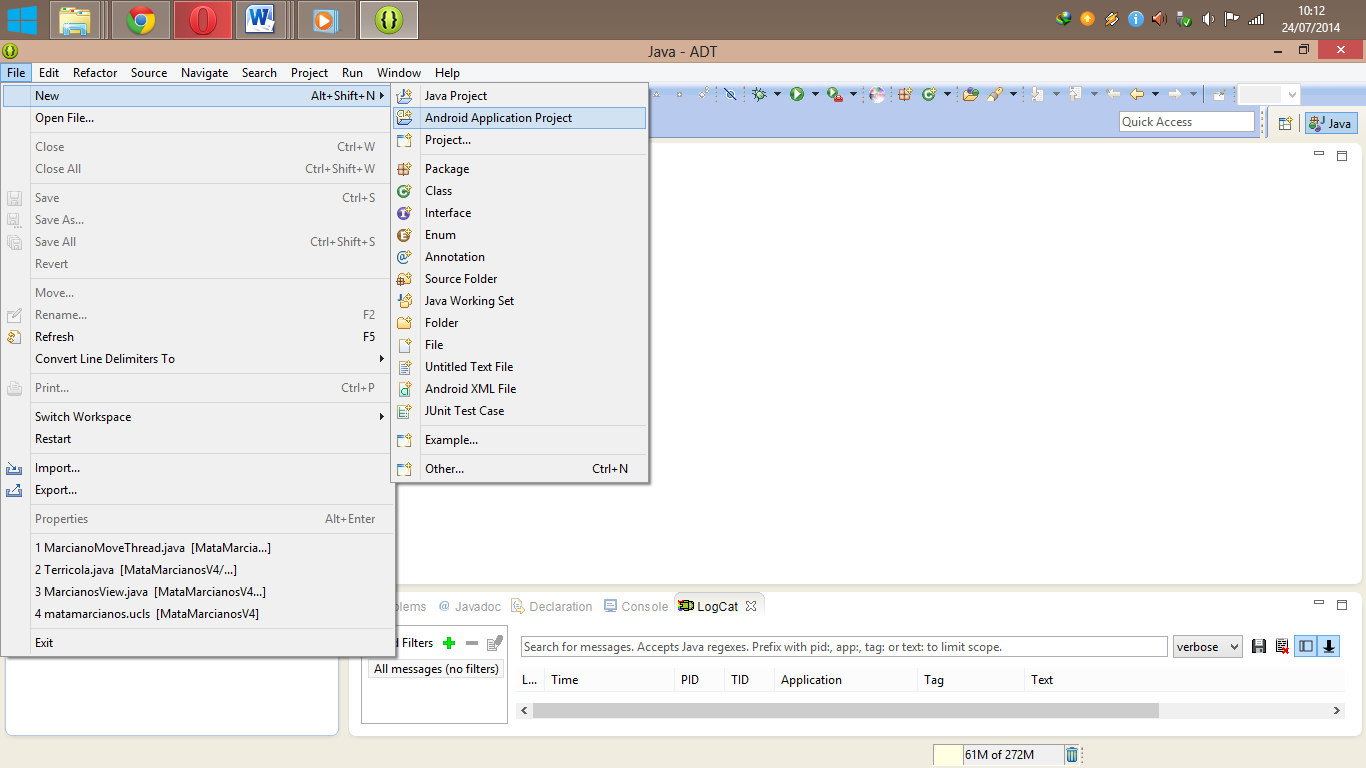
**3.- La primera app**

Luego de descargar e instalar, que mejor que meternos de lleno a hacer nuestra primera app, mediante la cual podremos entender como se estructuran los proyectos en Android para futuras referencias.

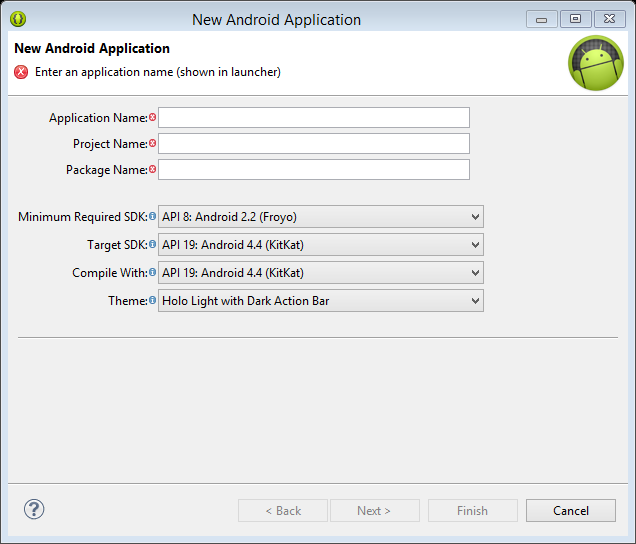
Lo que haremos, será crear una aplicación bastante sencilla, la cual contendrá una sola pantalla y en la que mostraremos un mensaje, nada más. Ya más adelante iremos incorporando otros elementos a esta app para hacerla un poco más completa.

Para comenzar, iremos al primer ícono del lado izquierdo, el cual lanzará el **New Project Wizard**, o asistente de nuevos proyectos *(también podemos acceder al mediante****File > New o Archivo > Nuevo****),*y en la ventana que se abrirá, seleccionaremos **Android Application Project**:





Al hacer click en Siguiente, llegaremos a la pantalla de configuración inicial de la nueva aplicación, en la que veremos algo como esto:



¿Qué significa cada uno de esos campos?, pues bien, veámoslo en detalle:

* **Application Name:** Como su nombre lo indica, es el nombre de nuestra app. Es el que el usuario verá cuando la descargue y la instale en su dispositivo, y el que aparecerá bajo el ícono una vez instalada, así como la tienda de Google Play y en el listado de apps en las opciones del dispositivo.
* **Project Name:** Al igual que el caso interior, su nombre indica claramente que trata del nombre del proyecto. A diferencia eso si del application name, es solo el nombre bajo el cual se guardará el proyecto en el espacio de trabajo de Eclipse *(se creará un directorio con ese nombre para guardar todos los subdirectorios y archivos delproyecto)*, y es de uso interno. **Debe ser único**, aunque si hay otro utilizando este nombre y lo eliminamos, quedará disponible para utilizar. Por lo general se acostumbra utilizar el mismo nombre que se usa para el Application Name.
* **Package Name:** Es un identificador único de la aplicación. Si bien es de uso interno y privado *(los usuarios nunca lo conocerán)*, es importante recalcar que una vez definido **no puede ser cambiado mientras la aplicación exista**. Mantener el mismo package name es la única forma de mantener coherencia entre las distintas versiones de la app *(así como con las firmas, algo que veremos más adelante)*. Si en un momento dado decidimos cambiar este package name, y subimos una nueva versión a Google Play o algún otro método de distribución, y el usuario intenta instalar este APK, no actualizará la app, si no que instalará una nueva instancia y se generará un duplicado, ya que para el sistema son 2 aplicaciones distintas. Lo que se usa para nombrar los package names comunmente es el nombre o dominio de la organización responsable de la app al revés y un nombre único, que por lo general es el nombre de la aplicación. Algo así:

**dominio.organizacion.app**

Por ejemplo, si en CLH decidiéramos hacer una app llamada "Chistosidades", un nombre de paquete apropiado sería:

**cl.comolohago.chistosidades**

### ¿Por qué la estructura de puntos?

El package name se define en esa estructura debido a que así son los nombres de paquetes válidos en Java, y por ende para este caso, debe tener una estructura igual.

**Nota:** Si bien al crear la app es necesario definir un Package Name, una app puede contar con múltiples paquetes dentro de si, siendo el definido inicialmente el principal. Muchos desarrolladores por orden prefieren crear varios paquetes para agrupar clases similares y así mantener una estructura más limpia dentro del proyecto. Veremos más sobre esto un poco más adelante.

Adicionalmente, se nos pide seleccionar 4 valores (aunque viene ya una opción seleccionada por defecto en cada uno de ellos):

* **Minimum Required SDK:** La versión mínima de Android que nuestra app soportará. Incluir versiones más antiguas por lo general asegurará que habrán más dispositivos que podrán hacer uso de la app, pero por otro lado también implica que existe la posibilidad de que algunas funcionalidades que implementemos no estén disponibles para esas versiones. Este es un punto importante de estudio al momento de comenzar un proyecto, para ver que es más importante, si sacrificar una funcionalidad o apuntar a una cantidad de público más exclusiva.
* **Target SDK:** Al contrario de lo anterior, esta opción representa la versión más reciente de Android sobre la cual hay certeza que esta app correrá sin problemas. Por lo general se suele apuntar a la más reciente que viene incluida en el SDK, aunque también dependerá del público objetivo de la app.
* **Compile With:** Es la versión de Android sobre la cual compilará Eclipse todo el código que escribamos y ejecutables que creemos. Lo recomendable acá es elegir la misma versión seleccionada en **Target SDK**, para poder asegurar compatibilidad.
* **Theme:** Es la plantilla base que utilizará nuestra aplicación. Esta plantilla le dará estilos a todos los elementos gráficos nativos de la app (barra superior de acción, botones, checkboxes, radio buttons, barras de progreso, etc.). Las versiones más recientes de Android incluyen 3 plantillas por defecto: Holo Dark, Holo Light y Holo Light with Dark Action Bar. Para más información y muestras visuales de las plantillas, pueden referirse a [este enlace de la documentación oficial de Android](http://developer.android.com/design/style/themes.html).

### ¿Por qué desarrollar para versiones más antiguas?

Si bien la tentación de solo desarrollar para versiones recientes de Android (sobre 4.0) es alta, debido a nuevos features y estilos visuales avanzados que se han ido integrando, incorporar soporte para versiones más antiguas es clave, debido a una situación existente en el mercado de Android desde hace un tiempo, llamada **Fragmentación**.

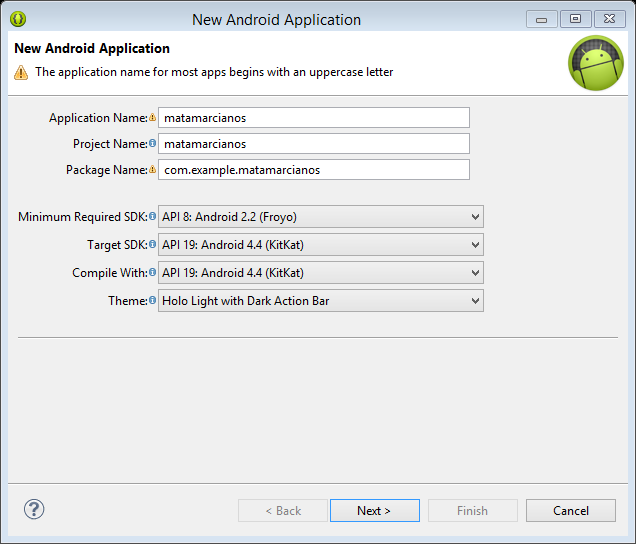
La fragmentación se refiere a las variaciones existentes hoy con respecto a los dispositivos con Android, tanto a nivel de hardware, como de software. Al ser un Sistema Operativo libre, muchas empresas productoras de hardware han decidido utilizarlo como sistema para sus dispositivos, y en base a ello han modificado y creado versiones propias para incluir en teléfonos y tablets, por lo que hoy hay features que pueden estar presentes en un dispositivo Samsung con Android (como por ejemplo TouchWiz) y no en un dispositivo HTC, que incluso podría tener la misma versión (a nivel de números), pero con features distintos.

Adicionalmente, hasta hace un tiempo, el canal de actualización de las versiones del sistema operativo muchas veces era a través de los operadores, por lo que si bien en el mercado se podía lanzar una nueva versión de Android, esta podía no alcanzar a todos los usuarios en un tiempo considerable. El mejor ejemplo a tener en cuenta es que si bien hoy la versión actual de Android es la 4.3, hay una gran cantidad de dispositivos que aún están corriendo versiones 2.3 (40% hasta hace poco), 2.2 e incluso 2.1, por lo que pensar en desarrollar solo para versiones nuevas puede excluir a una gran parte de usuarios de Android, y por ende estaríamos directamente perjudicando la popularidad de la app.

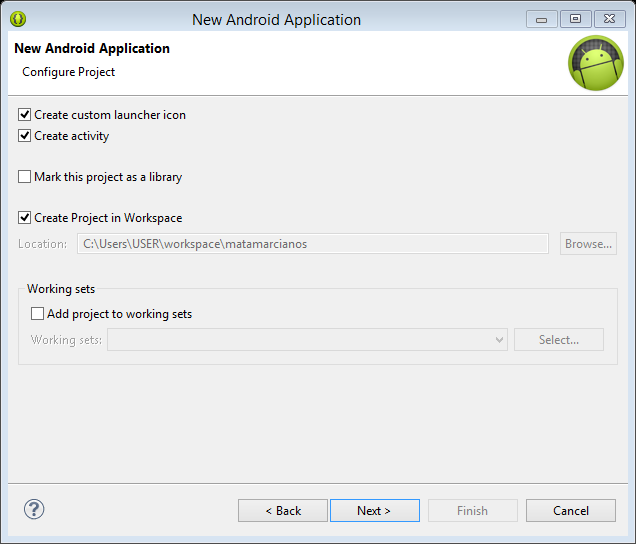
Este escenario es completamente diferente en iOS, donde al existir una arquitectura centralizada de actualización, hay una certeza que sobre el 90% de los usuarios hoy tiene la versión más reciente del S.O(o al menos en el rango) por sobre un porcentaje bastante menor en Android.

Afortunadamente, hace algún tiempo ya se vienen tomando medidas para esto. El lanzamiento de los servicios de Google Play ha permitido crear esta arquitectura de actualización directa, por lo que es factible que el problema de fragmentación (al menos en términos de versiones de S.O.)vaya disminuyendo con el pasar del tiempo.

Pero para eso falta, así que por ahora, a pensar en todos los casos.



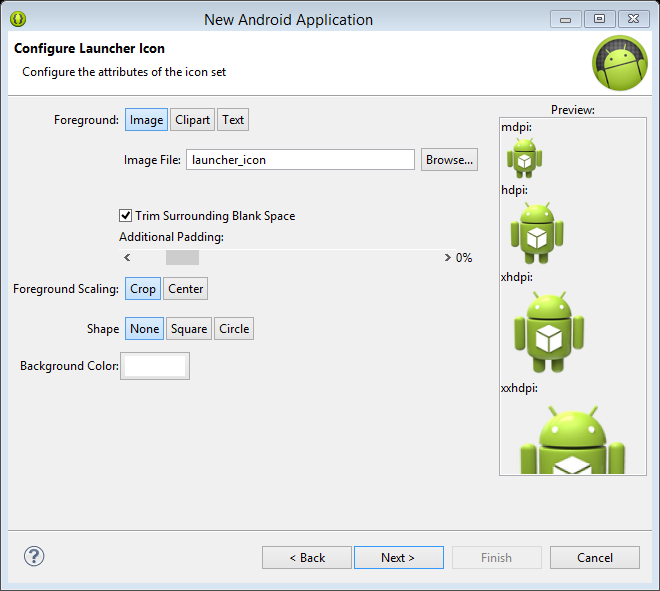
Al hacer click en Siguiente, nos encontraremos con la siguiente ventana:



Las opciones que vienen marcadas por defecto en esta ventana representan lo siguiente:

* **Create custom launcher icon:** Al tener marcada esta opción, se creará un ícono para la app. Este es el ícono que aparecerá tanto en Google Play como en el dispositivo del usuario al instalar.
* **Create activity:** Al marcar acá, se creará automáticamente una actividad para comenzar a trabajar. Veremos en detalle un poco más adelante que lo que es una actividad.
* **Create Project in Workspace:** Al marcar aquí, se creará un directorio para el proyecto dentro del directorio de nuestro espacio de trabajo *(el que definimos inicialmente)*. El nombre de este directorio será el que definimos en **Project Name** en la ventana anterior.

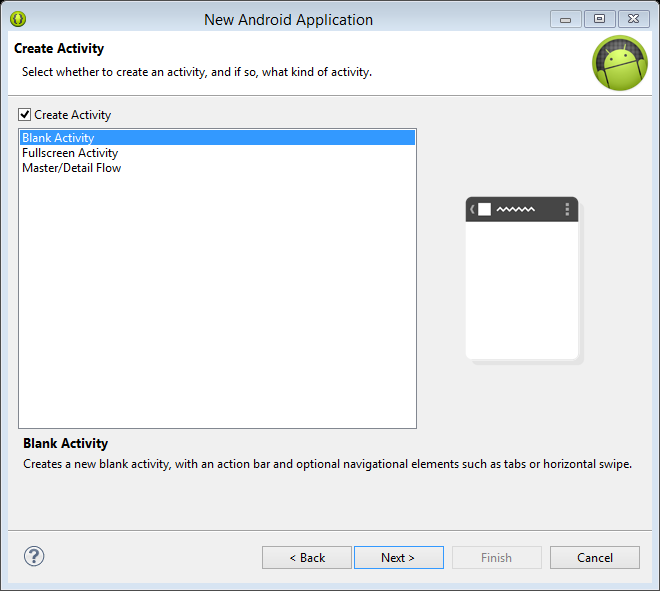
Las otras opciones no las veremos por ahora. Hacemos click en siguiente, y veremos la siguiente ventana:



Esta ventana nos permite definir una imagen y ajustarla para ser utilizada como ícono de la app. Es importante recalcar que el ícono **puede ser modificado posteriormente**, por lo que normalmente saltarán esta configuración, ya que son pocos los casos en los que tendrán definido un ícono al momento de empezar a trabajar en una app.

Entre las opciones que se nos entregan está el definir la imagen a usar *(o texto si así lo deseamos)*, darle padding o espacio al ícono, la forma *(círculo, cuadrado o forma natural de la imagen)*, el color de fondo y podemos ver una vista previa de cómo será visualizado en distintas resoluciones *(mdpi, hdpi, xhdpi, xxhdpi, detallaremos las resoluciones más adelante)*.

Para este caso, como no definiremos un ícono, hacemos click en Siguiente y veremos:



En esta ventana se nos preguntará si crear una actividad y de qué tipo. Y como ya no puedo seguir excusando ni demorando la explicación, que es una Actividad o Activity en Android.

### ¿Qué es una Actividad o Activity en Android?

Para entender que es una "Activity" en Android, es bueno mirarla desde 2 aspectos: Uno conceptual y uno técnico.

Conceptualmente, se le llama "Activity" en Android a lo que está ocurriendo en la pantalla del dispositivo en un determinado momento del ciclo de vida de la aplicación (desde que se lanza hasta que se cierra).

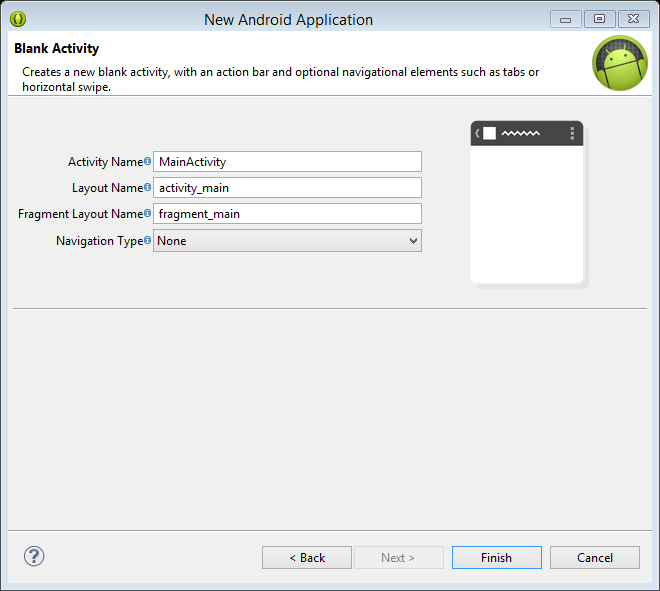
Tecnicamente, una "Activity" es una clase en Java que **hereda** desde la clase Activity de Android y ayuda a definir el comportamiento de una aplicación en distintas instancias. Al heredar desde la clase Activity, es posible manejar el ciclo de vida de la app y definir mediante código como queremos que se comporte en cada una de esas instancias (veremos esto en detalle en el próximo número). Adicionalmente, nos permite codificar distintas acciones que serán ejecutadas por la aplicación en ese momento, cargar vistas (representación visual en pantalla de elementos gráficos como botones, formularios y otros), interactuar de forma interna con el dispositivo y mucho más.

Básicamente, una aplicación está compuesta de muchas actividades (clases) que interactuan entre si cada vez que el usuario realiza una acción. Un ejemplo cercano para entender este concepto, especialmente si han hecho desarrollo Web, es pensar en una actividad como "las páginas de un sitio". Concepto cercano. Quizás no acertado, pero cercano.

Si bien hay un ciclo de vida definido en cada actividad (desde que se inicia hasta que se cierra), en una actividad es posible crear una infinidad de métodos personalizados para el código de nuestra aplicación y también es factible cargar todas las vistas que queramos dentro de ella (aunque no al mismo tiempo). A pesar de eso, por lo general lo que se hace es una relación 1 a 1 entre una actividad y una vista para mantener un mejor orden (a excepción de cuando realmente no se justifica crear una actividad para una vista y es mejor agruparla en otra).

Si el concepto es complejo de entender, no se preocupen. A medida que vayamos desarrollando algunos ejemplos se comprenderá con mayor facilidad.

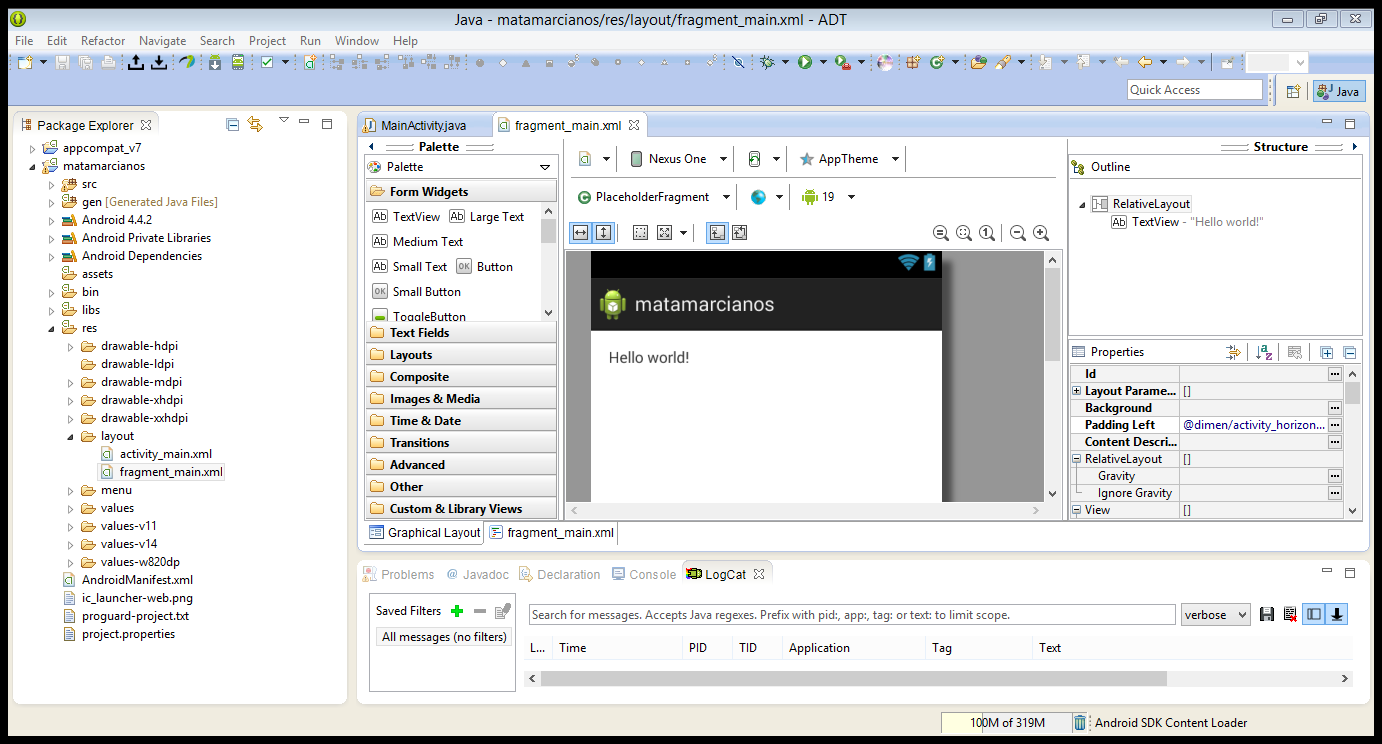
Con eso claro, marcamos en la ventana que si queremos crear una actividad, y seleccionamos **Blank Activity** (veremos los otros tipos en futuras ediciones), ya que queremos una actividad con lo menos posible para empezar a trabajar (los otros tipos incluyen algo de código de ejemplo para orientarnos). Le damos a Siguiente y veremos:



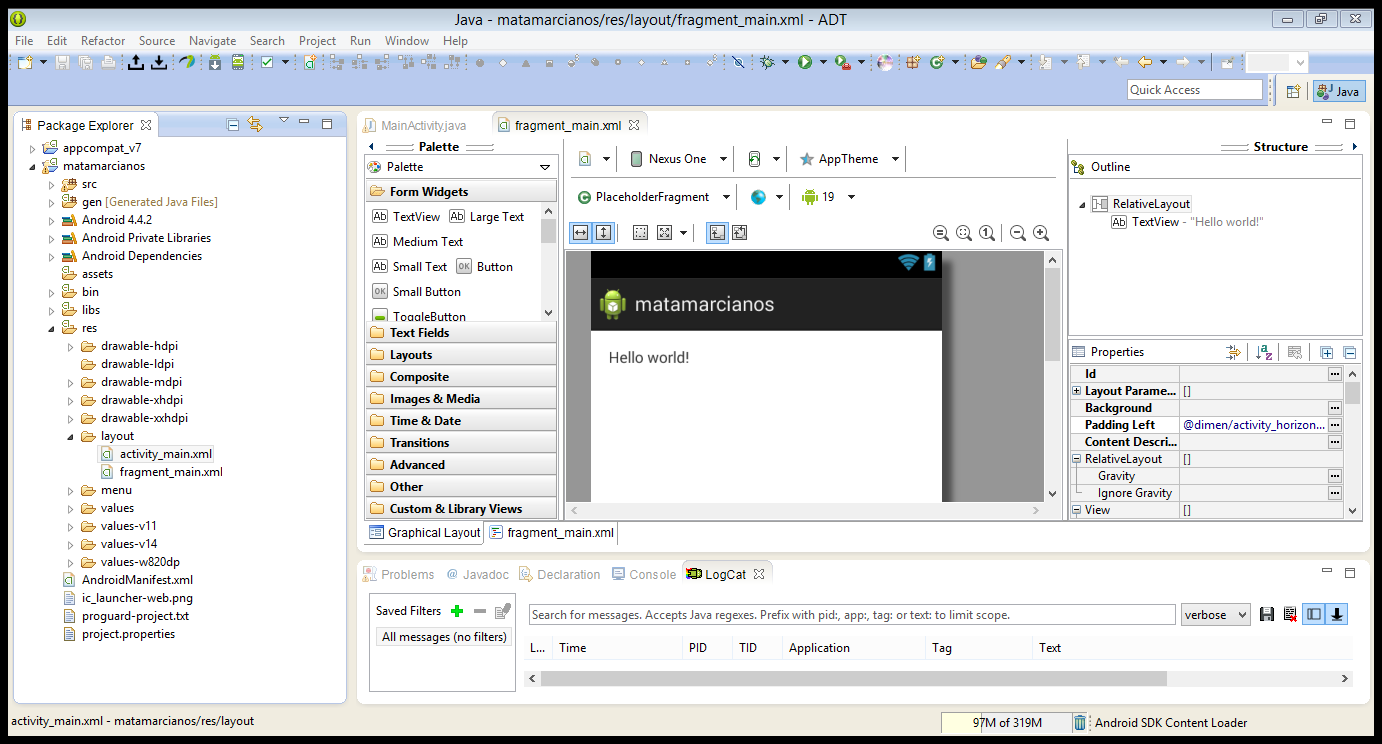
En esta ventana definiremos 3 cosas:

1. **Activity Name:** Será el nombre de la clase que representa la actividad. Puede ser cualquier nombre *(mientras respete las convenciones de nombramiento de clases en Java, o sea, sin símbolos extraños)* aunque en algunos casos se usa agregar después del nombre la palabra Activity para mayor orden *(por ej. si es una actividad de búsqueda, se nombra Search o SearchActivity)*. Ese mismo nombre tomará el archivo .java que representa a la actividad dentro del paquete de la aplicación. El nombre de la actividad debe ser único dentro del paquete de la aplicación.
2. **Layout Name:** Será el nombre de la vista asociada a la actividad. Esta vista es un archivo XML en donde se realiza la ubicación y representación gráfica de los elementos que mostraremos en pantalla *(veremos esto en detalle en el próximo número)*. Recuerden que como indicamos inicialmente, esta es solo **una** vista asociada a la actividad. Eventualmente podríamos tener todas las que queramos.
3. **Navigation Type:**Dentro de esta lista, podemos seleccionar si queremos integrar algún tipo de navegación a la actividad. Por ahora seleccionamos **None**, ya que queremos ver cada tipo en detalle más adelante, porque es algo que debemos ver con atención.

Con eso listo, terminamos el asistente y hacemos click en **Finish**, con lo que veremos que en Eclipse se abrirá una pantalla en el editor, que representa a la vista asociada a la actividad que acabamos de crear, en la cual habrá un texto de ejemplo **"Hello World"**:



Además, podremos ver que al lado izquierdo se ha agregado el proyecto que acabamos de crear, el cual al ser ampliado (click en la flechita del lado izquierdo), desplegará una estructura de directorios y archivos de esta forma:



Esta estructura contiene directorios, entre los que por ahora destacaremos:

* **src:** En este directorio se incluyen todos los paquetes y clases (entre ellas las actividades) de la aplicación. Como su nombre lo indica, es el código fuente (source) del proyecto.
* **gen:** Aquí se incluyen las clases generadas automáticamente por Eclipse para el proyecto, en base al código que creamos y los recursos que incluimos. Entre estas clases, se incluye el famoso **R.java,**una clase generada automáticamente que incluye las referencias a las áreas de memoria de cada recurso de la aplicación y genera identificadores que pueden ser llamados desde la aplicación para no referirse a estos recursos de forma directa (veremos más sobre esto en un próximo número).
* **assets:** En este directorio se incluirán recursos gráficos para ser utilizados en la app, los cuales no serán indexados y se referenciarán directamente (por ruta) en tiempo real. Un buen ejemplo es añadir acá tipografías que queremos utilizar en la aplicación. La diferencia con los otros recursos se entenderá mejor a medida que vayamos desarrollando.
* **libs:** Acá se incluyen librerías externas (en formato jar) que utilizaremos en nuestras apps.
* **res:** Y aquí se incluirán todos los recursos gráficos que **si serán referenciados** mediante identificadores (definidos en el R.java) en la app. Elementos como imágenes, colores, dimensiones y estilos irán acá.

Adicionalmente hay un elemento más a destacar que es el **AndroidManifest.xml**. Este archivo incluye toda la información de configuración de la app. Entre esta info:

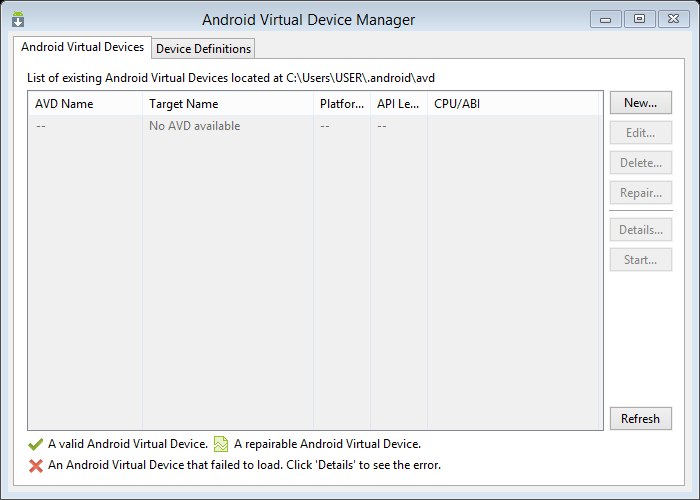
* Actividades de la app (marcando cual es la que se lanza al iniciar la app)
* Permisos solicitados por la app
* Versiones de Android soportadas
* Versión
* Ícono y nombre de la app
* Estilo

Y algunos otros valores que iremos viendo a medida que avancemos. Bien, antes de empezar a programar nuestra primera app, falta un solo elemento para ver: los emuladores.

### Emuladores

Para poder probar las apps que creemos, tenemos 2 posibilidades: Probar en un dispositivo físico y probar en un emulador. La ventaja de disponer con emuladores es que por lo general es difícil contar con distintos tipos de dispositivos para saber como la app se verá en diferentes resoluciones, una posibilidad que los emuladores nos dan. Si bien la representación no es al 100%, es muy pero muy cercana a la realidad, aunque en términos de velocidad un poco más lento, ya que comparte recursos con nuestro equipo. Por eso es bueno tener en cuenta que si una app les funciona un poco lento en el emulador, en el dispositivo por lo general funcionará más rápido.

Crear un emulador en Android es muy sencillo, mediante la herramienta **Android Virtual Device Manager** (el 6to ícono de izquierda a derecha o bien el menú ***Window > Android Virtual Device Manager***):



Ahí hacemos click en **New** y veremos el asistente para crear un nuevo emulador:

Entre los datos que debemos llenar acá:

* **AVD Name:** Un nombre para identificar el emulador. Debe ser único entre todos los que creemos.
* **Device:**Que tipo de dispositivo representará el emulador. Para este caso podemos elegir por ejemplo 3.2" HVGA Slider (ADPI1) (320x480:mdpi), aunque en realidad cualquiera por ahora sirve. Solo tener en cuenta que a mayor resolución, mayor espacio ocupará en pantalla el emulador, así que para comenzar una resolución así o un 480x800 es bastante cómodo.
* **Target:** La versión de Android que tendrá el emulador. Lo ideal es parear con la que seleccionamos para el proyecto.
* **Front y Back Camera:**Si el dispositivo tendrá cámara, y si la emulará o tomará imágenes desde la Webcam del dispositivo *(si es que hay alguna)*.
* **Internal Storage:** La capacidad del dispositivo en memoria interna.
* **SD Card:** La capacidad del dispositivo en memoria externa.

Hay otros que por ahora no veremos, para revisarlos en mejor detalle.   Y ahora si, después de todo esto, vamos a codificar.

Primero abriremos la actividad para revisar su contenido, haciendo doble click en **src/cl.comolohago.chistosidades/MainActivity.java** y veremos algo como esto:



Como podemos ver, la actividad viene con 2 métodos declarados por defecto: **onCreate** y **onCreateOptionsMenu**. El primero es el método que se ejecutará automáticamente al iniciar esta actividad *(que a su vez es la actividad por defecto que se ejecutará al iniciar la app).*El segundo corresponde al menú de opciones de la actividad, que por ahora ignoraremos para verlo en más detalle posteriormente.

El método **onCreate** tiene 2 lineas de código que analizaremos a continuación. La primera,

super.onCreate(savedInstanceState)

Lo que hace es llamar a la clase de la que hereda, pasándole como parámetro el estado actual de la app. Esta línea siempre estará presente y vendrá incluída por defecto, así que no será necesario modificarla.

A continuación, viene la línea:

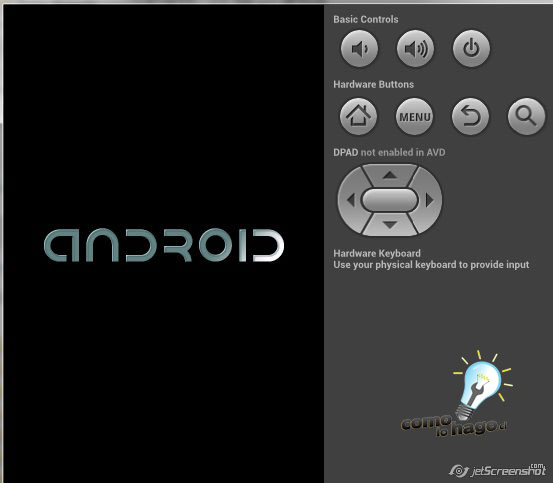
setContentView(R.layout.activity\_main)

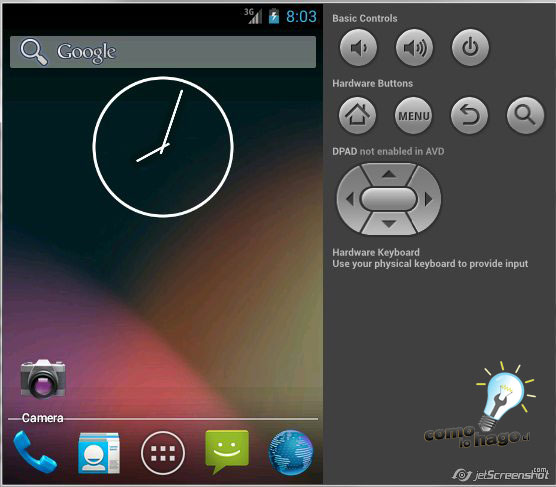
La cual se encarga de definir la vista que será cargada al momento de iniciar la actividad. Esto se hace mediante el método **setContentView**, al cual se le pasa como parámetro el identificador de la vista y le entregamos una referencia de la clase R *(tal como lo vimos unas líneas atrás)*. Este identificador llama primero a la clase *(R)*, luego al tipo de recurso *(layout)* que representa a la vista *(las cuales como archivos siempre quedan almacenadas en* ***res/layout****)*y finalmente el nombre de la vista *(activity\_main)* sin la extensión xml.

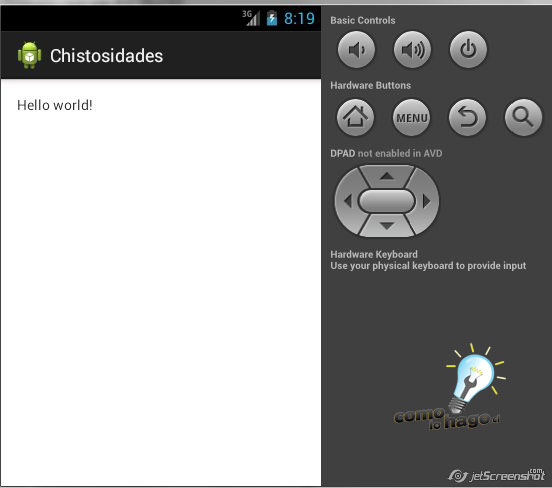
Teniendo esto en cuenta, al ejecutar esta app, nuestra primera aplicación, se va a cargar la vista correspondiente a activity\_main.xml y nos va a mostrar el texto que vimos hace un par de imágenes atrás. Así que para eso hacemos click derecho en el proyecto *(en el lado izquierdo, en el explorador de archivos)* y vamos a **Run as > Android Application.**Si nos pide elegir emulador, seleccionamos el que creamos recién, y si no, esperamos a que el emulador por defecto se abra y nos muestre la app *(si se demora en iniciar, paciencia, el emulador de Android tiende a ser así)*, o si por alguna razón no carga el emulador, vamos primero a **Run as > Run configurations** y seleccionamos la segunda pestaña del lado derecho **"Target"**, en la cual aparecerá un listado de todos los emuladores que tengamos y seleccionamos una de las tres opciones, según lo que más nos acomode:

1. **Always prompt to pick device:**Que siempre nos pregunte en que emulador lanzar.
2. **Launch all compatible devices/AVD:** Que la app se lance simultáneamente en todos los emuladores y dispositivos compatibles.
3. **Automatically pick compatible device:** Que automáticamente seleccione un emulador o dispositivo compatible.

Por lo general, como uno tiene a disponer de varios emuladores, y prefiere probar en uno por defecto, lo mejor es seleccionar la primera opción, y que nos pregunte al momento de lanzar en cual queremos. Con eso, lanzamos la app.



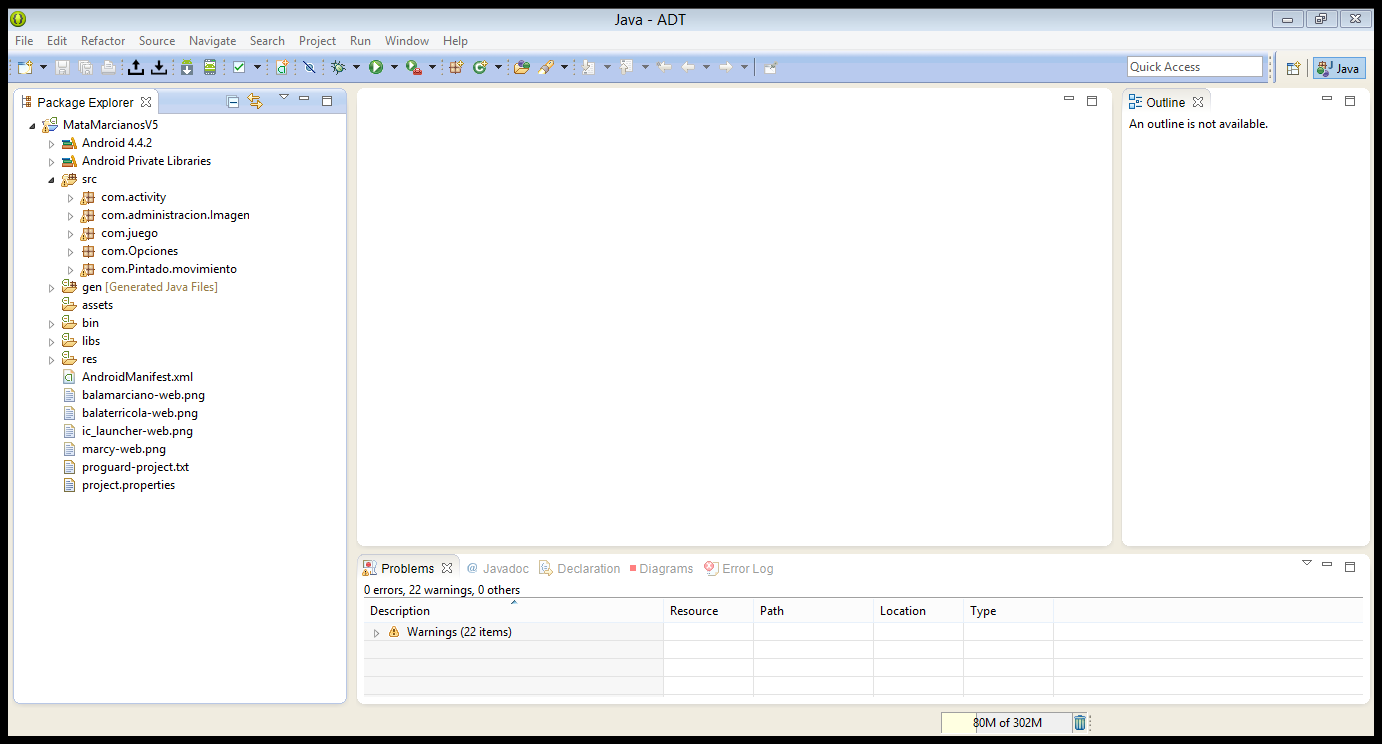




  Y así de simple tenemos ya en el emulador nuestra primera App.

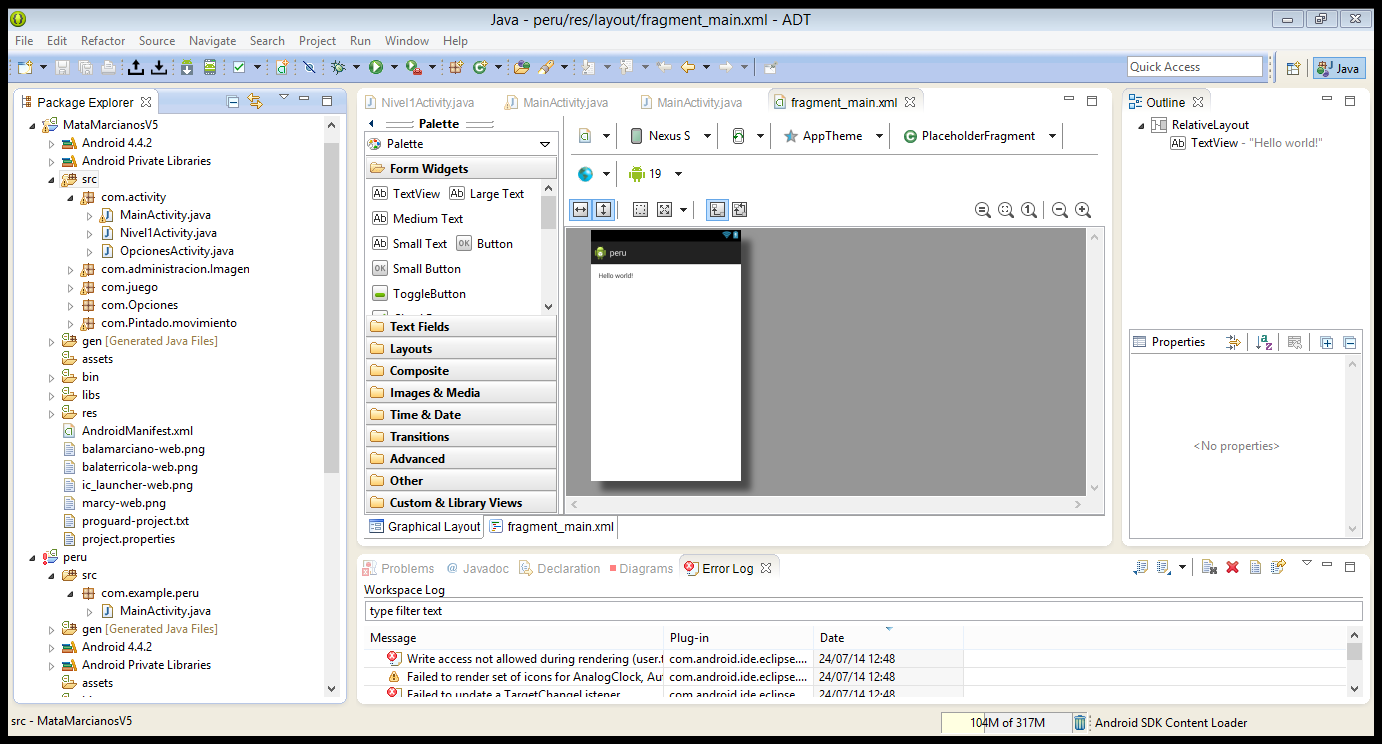
**Desarrollan la aplicación mata marcianos**

**Creación de pates**



**Paquete com.activity**

Es el paquete que crea por defecto al momento de crear nuestro aplicativo.



**Clase MainActivity.java**

La clase principal de proyecto O DA arranque al programa.

**Inicio de juego**

**private** **void** empiezaJuego()

**OPCIONES DE JUEGO**

**private** **void** muestraOpciones()

**OPCIONES DE AUDIO**

**private** **void** audio()

**OpcionesActivity**

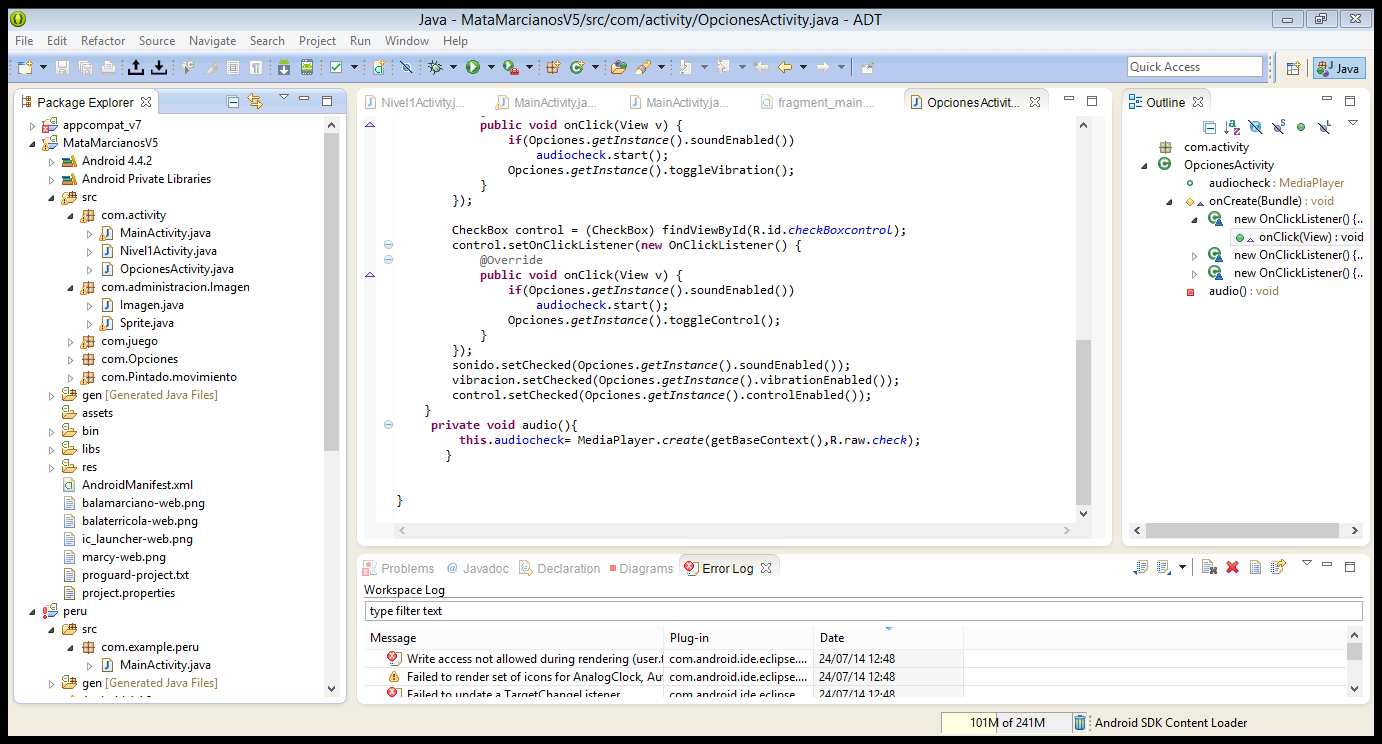
**La clase en donde se programó LAS configuración de juego como son: sonido,** direcciones de flechas.

CONFIGURACION DE JUEGO

**protected** **void** onCreate(Bundle savedInstanceState)

**Paquete com.admistracion.imagen**

Es el paquete en donde se cargan el todo los imágenes utilizadas del juego las dimensiones de cada figura.

****

**Clase imagen .java**

**Imagen**

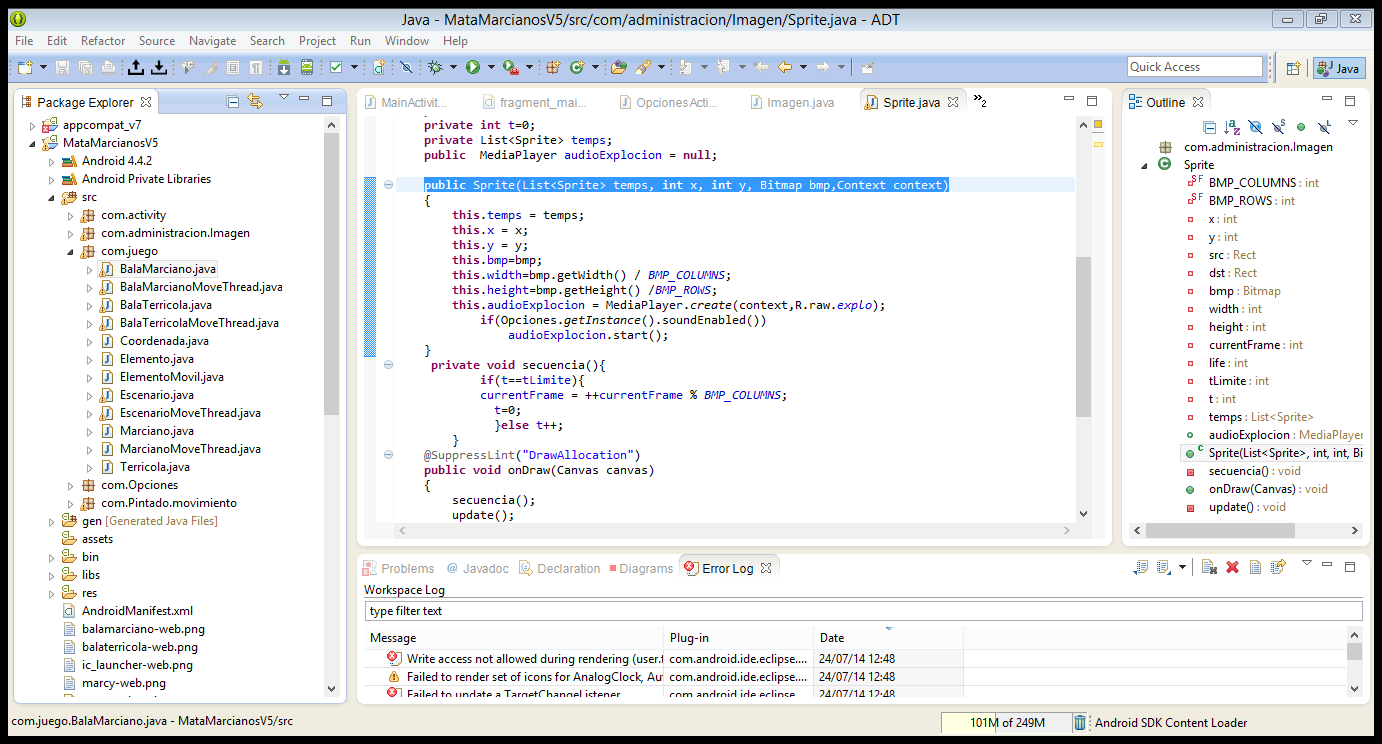
**public** Imagen(Coordenada origen, **int** ancho, **int** alto, Bitmap image)

**Clase Sprite.java**

**public** Sprite(List<Sprite> temps, **int** x, **int** y, Bitmap bmp,Context context)

**Paquete Com.juego**

**El paquete que la programación del juego.**



**Clase Balamarciano.java**

**public** BalaMarciano(Coordenada origen, **int** ancho, **int** alto,Bitmap image,ArrayList<Marciano> marciano,**int** indice)

**ClasemarcianomoveThead.java**

**public** BalaMarcianoMoveThread(BalaMarciano balamarciano ,Rect screen,**int** op)

**Balaterricola.java**

**public** BalaTerricolaMoveThread(BalaTerricola balaterricola ,Rect screen,Context context) {

**BalaTerriculaThead.java**

**public** BalaTerricolaMoveThread(BalaTerricola balaterricola ,Rect screen,Context context) {

**Coordenas.java**

**public** **class** Coordenada

**Elemento.java**

**public** Elemento(Coordenada origen, **int** ancho, **int** alto,Bitmap image)

**public** **boolean** puedoMover(**int** x, **int** y, Rect screen)

**ElementoMovil.com**

**public** **boolean** puedoMover(**int** x, **int** y, Rect screen)

**Escenario.java**

**Escenario**

**public** Escenario(Coordenada origen, **int** ancho, **int** alto,Bitmap image,Elemento[] escenario)

Movimiento

**public** **void** move(**int** x,**int** y)

**EscenarioMoveThread.java**

**public** EscenarioMoveThread(Escenario escenario ,Rect screen)

**Escenario**

**Marciano.java**

**marciano**

**public** Marciano (Coordenada origen, **int** ancho, **int** alto,Bitmap image,Bitmap image1,ArrayList<Elemento> balasMarcianos,ArrayList<BalaMarcianoMoveThread> balasMarcianosMoveThread,**int** columna)

Disparar bala

**public** **void** DispararBala(ArrayList<Marciano> \_marciano,**int** \_indice,Rect screen,Bitmap \_image,**boolean** k)

**MarcianoMoveThread.java**

Movimiento Marciano

**public** MarcianoMoveThread(Marciano marciano,Rect screen,Context context)

**Movimiento**

**public** MarcianoMoveThread(Marciano marciano,Rect screen,Context context)

**Terricola.java**

**Terrícola**

**public** Terricola(Coordenada origen, **int** ancho, **int** alto,Bitmap image,ArrayList<BalaTerricola> balaTerricola,ArrayList<BalaTerricolaMoveThread> balasTerricolaMoveThread,Context context)

**Disparar Bala**

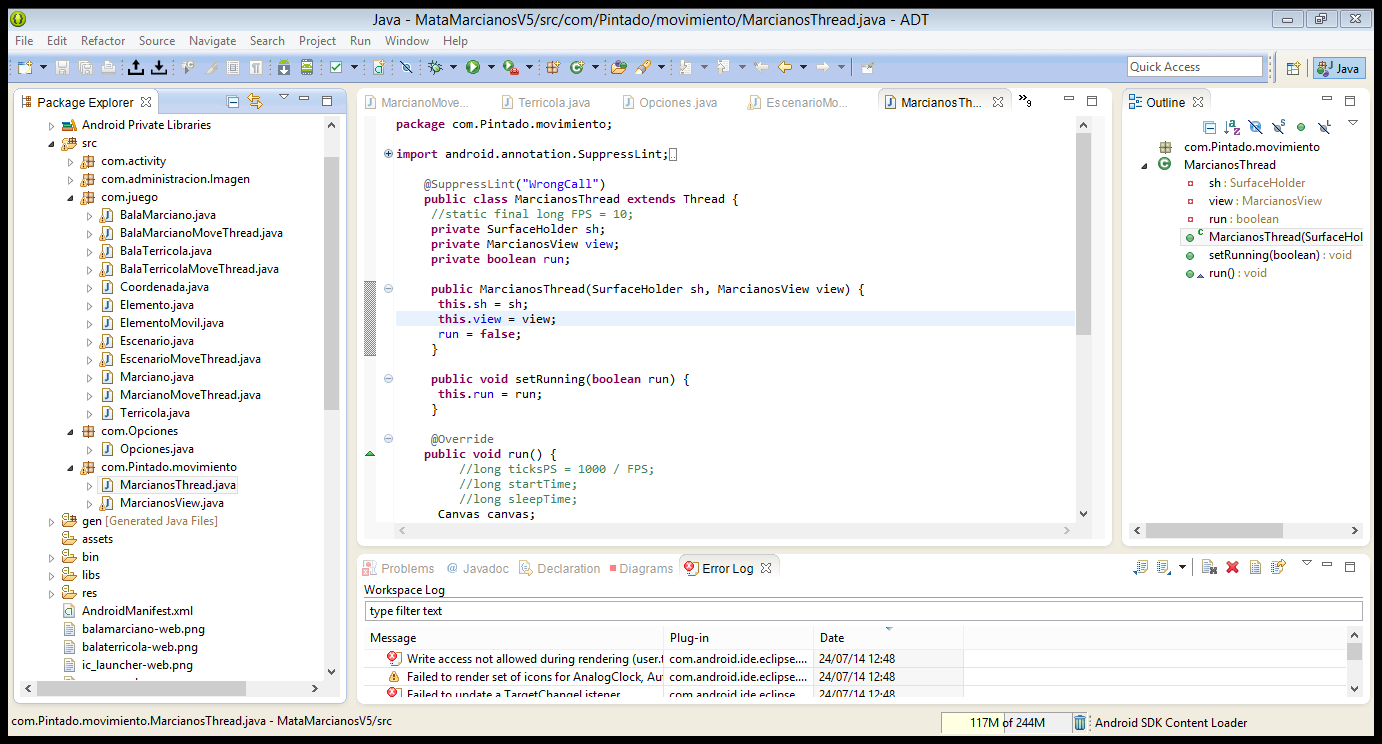
**public** **void** DispararBala(Rect screen,Bitmap \_image,Context context)

**Paquete com.Opciones**

**Clase opciones.java**

**public** **static** **synchronized** Opciones getInstance()

**Paquete com.pintadoMovimiento**

****

**Clase MarcianThead.java**

**MarcianoView.java**