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.

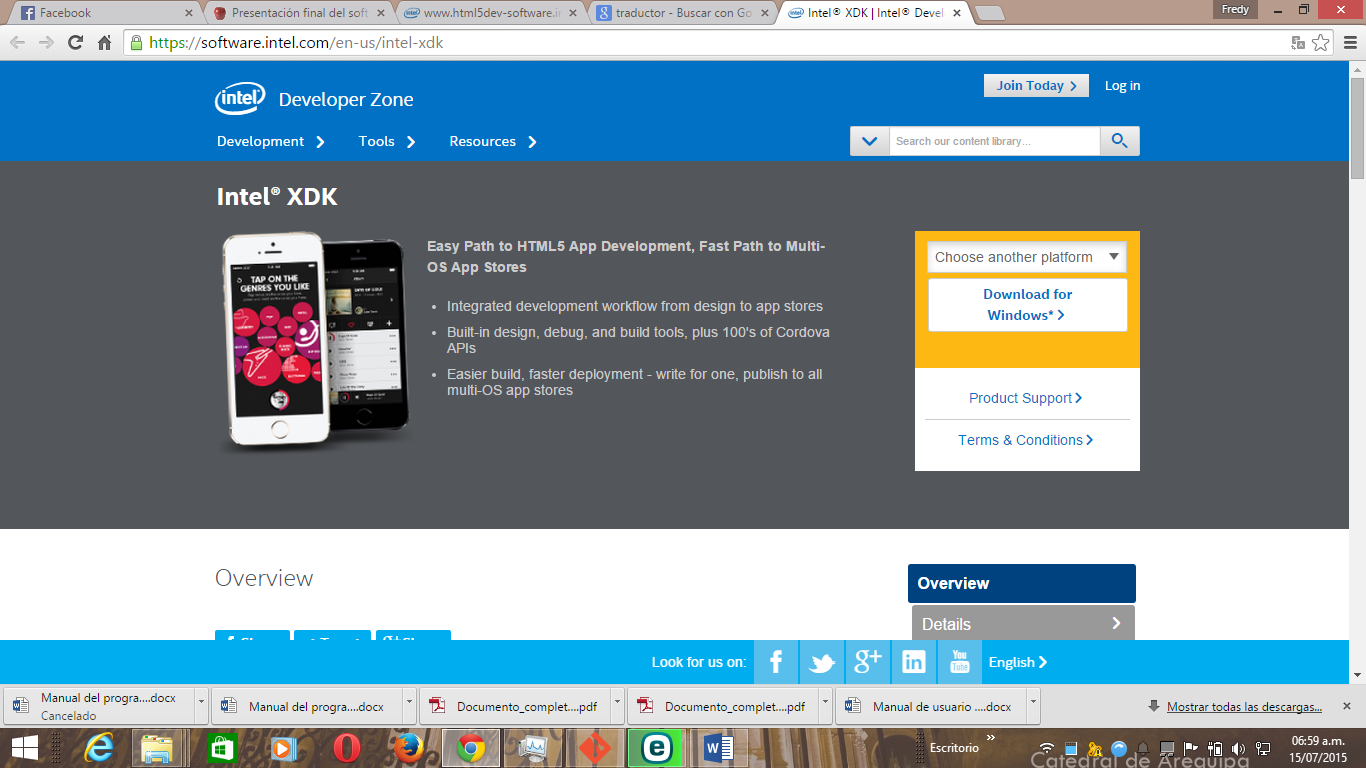
## IV.- ¿Es esta la guía definitiva para crear apps en Android?

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.

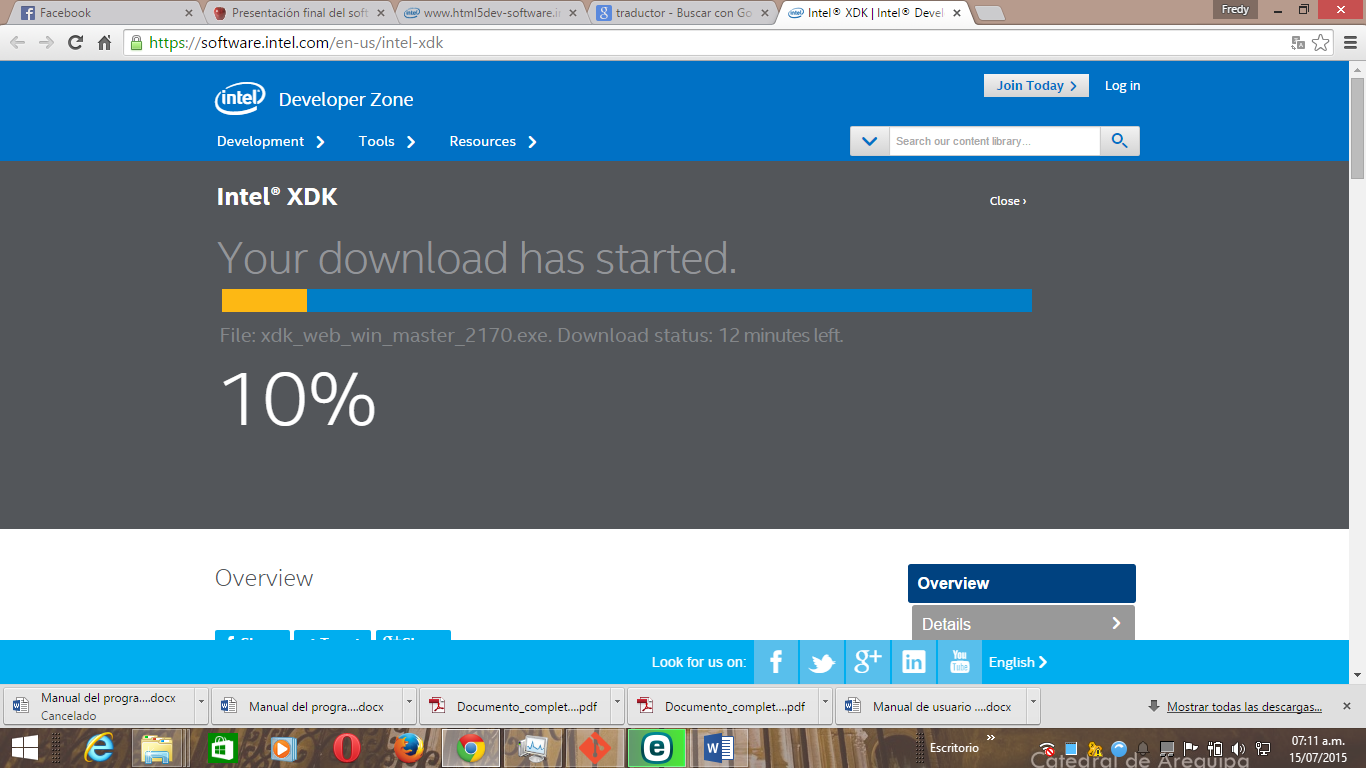
## 1.- Descarga del Intel XDK

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.

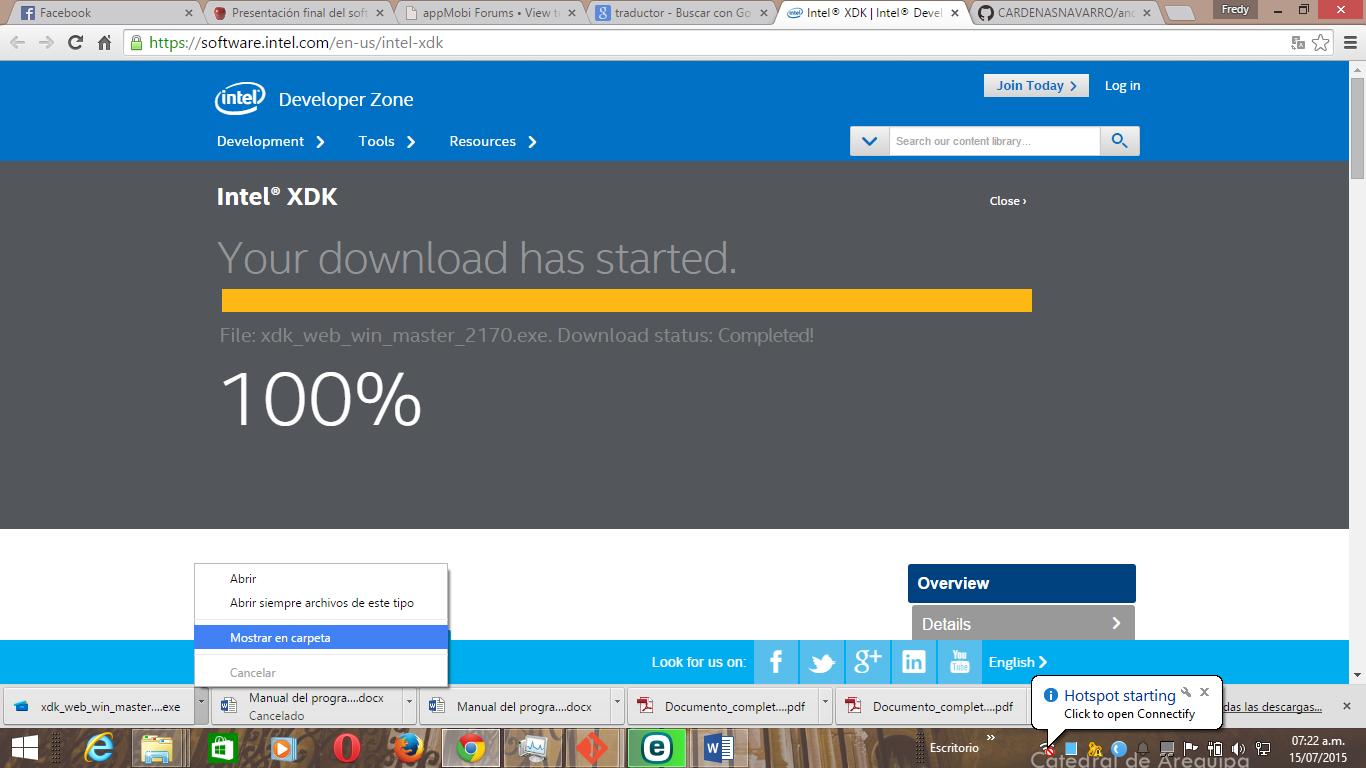
Para descargar, vamos a [https://software.intel.com/en-us/intel-xdk](https://software.intel.com/en-us/intel-xdk ) y verán en los textos de la parte derecho, un vínculo que dice **"Download for Windows"**, la plataforma se puede seleccionar **“Choose another platform”** según requiere el usuario. Hacemos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



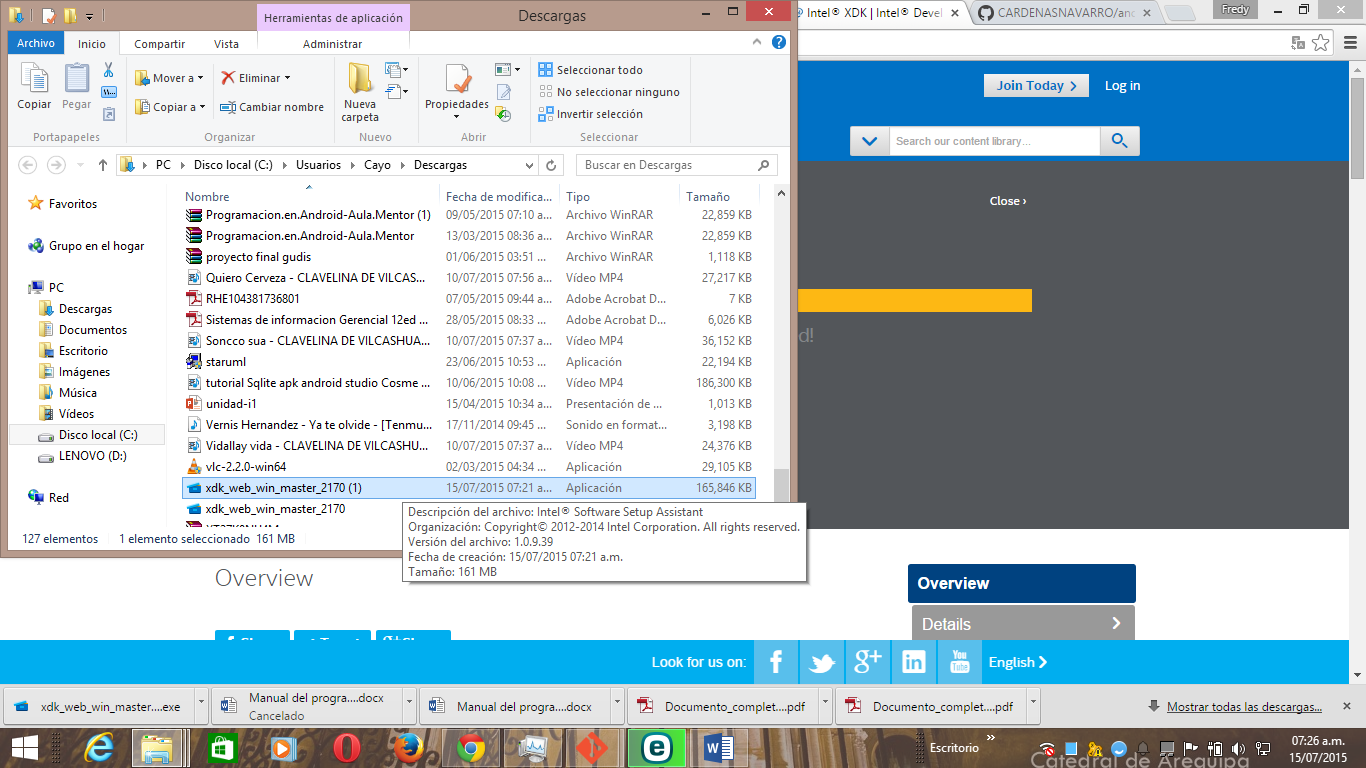
Luego de ello, nos llevará a la pantalla de descargas, en donde automáticamente reconocerá nuestro Sistema operativo y empieza descargar para nuestra plataforma, la descarga puede demorar según la velocidad del internet.



Cuando haya terminado se busca el instalador en nuestra PC , por defecto se guarda en la carpeta de descargas de nuestra pc. En peor de los casos se vera en la parte inferior y hacemos anticlik para ver el origen en donde esta almacenado.

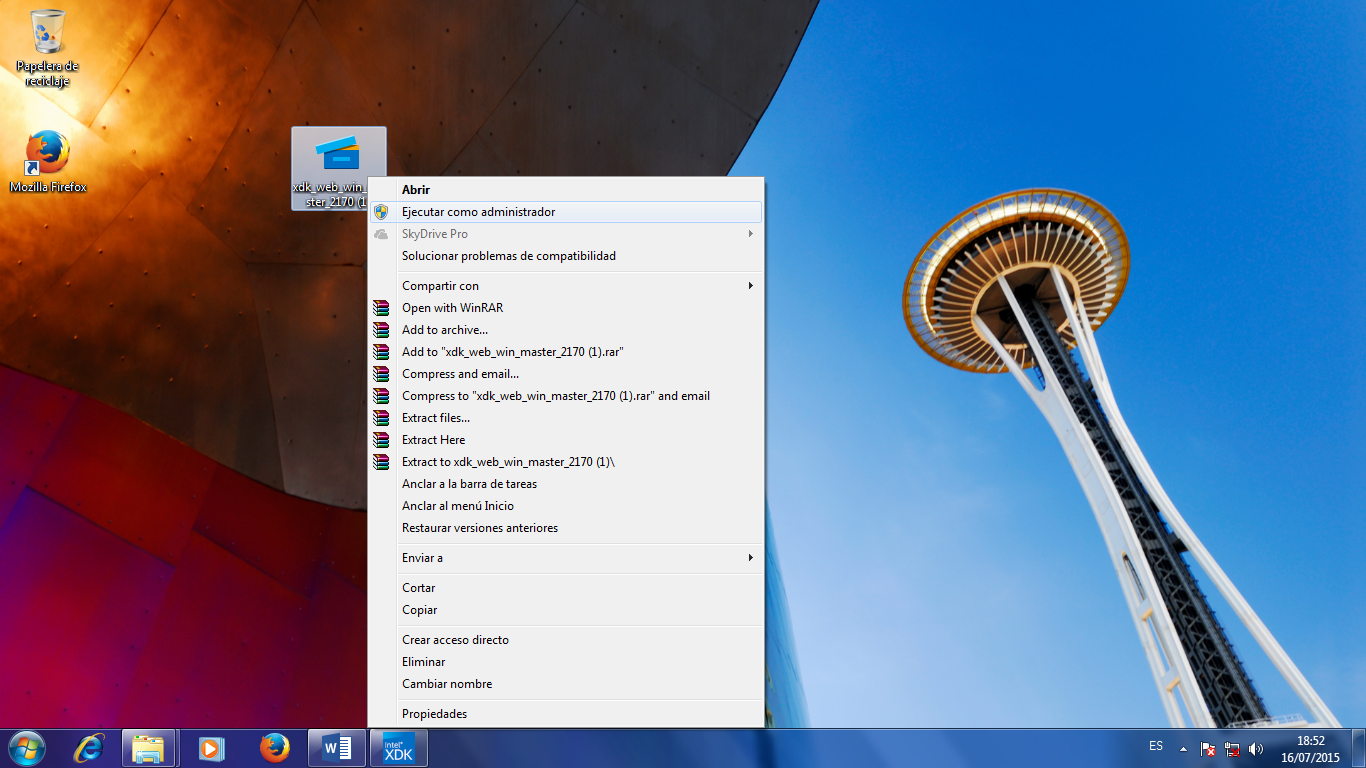


Allí tenemos la descarga.



**2.- Instalación**

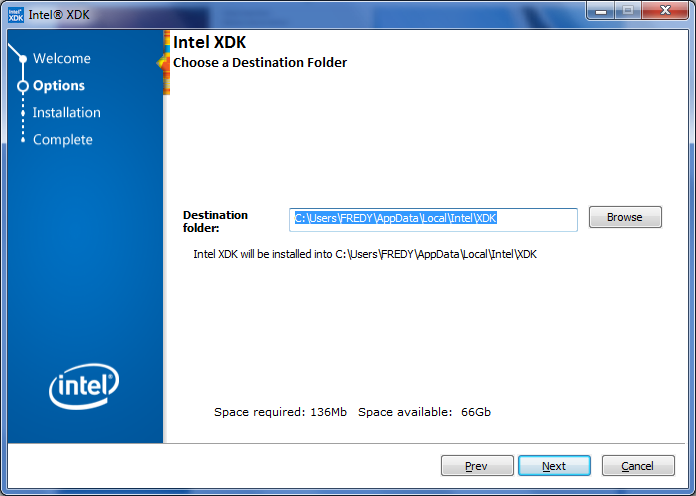
La instalación del SDK de Android es bastante sencilla, ya que para empezar a desarrollar, solo debemos descomprimir el archivo que acabamos de descargar y estamos listos. Así que doble click y descompriman el contenido en la ubicación que más les acomode *(su directorio personal es un buen lugar)* y tendrán un directorio ahí de nombre parecido a **"xdk\_web\_win\_master\_2170 "**, el cual se ejecutara como administrador.



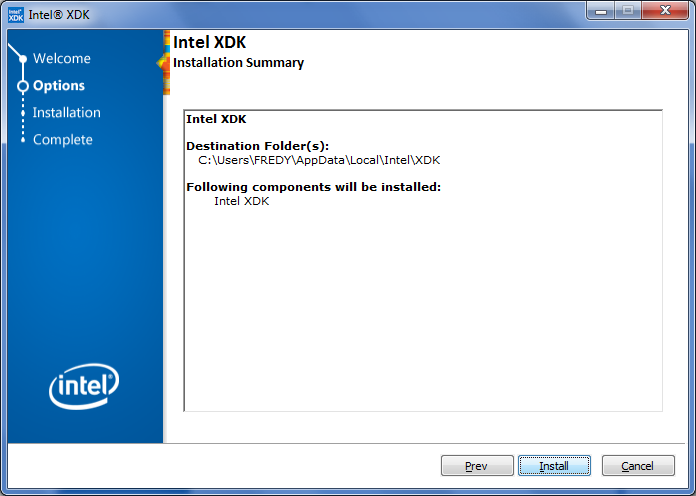
**Importante:** Si bien el archivo descargado pueden descomprimirlo y ubicarlo en cualquier parte de su equipo, es muy importante **no mover los directorios ni archivos que están en su interior**, ya que para futuras actualizaciones se requiere respetar esa estructura para poder ubicar las preferencias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

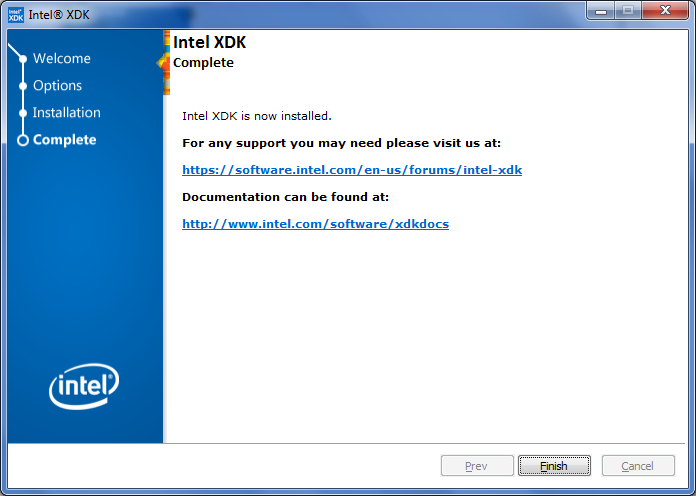


En próximos es solo dar clic en next, a medida que vayamos instalando

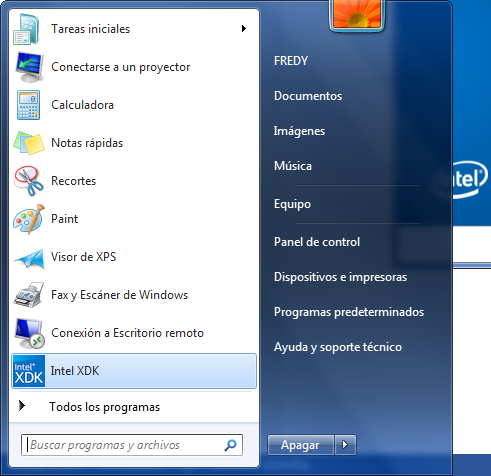


En este directorio que definamos se crearán y guardarán automáticamente todos los contenidos de Intel XDK. Pueden elegir la ruta que quieran y/o la que más les acomode. En nuestro caso, dejamos todo en un mismo directorio para su instalación



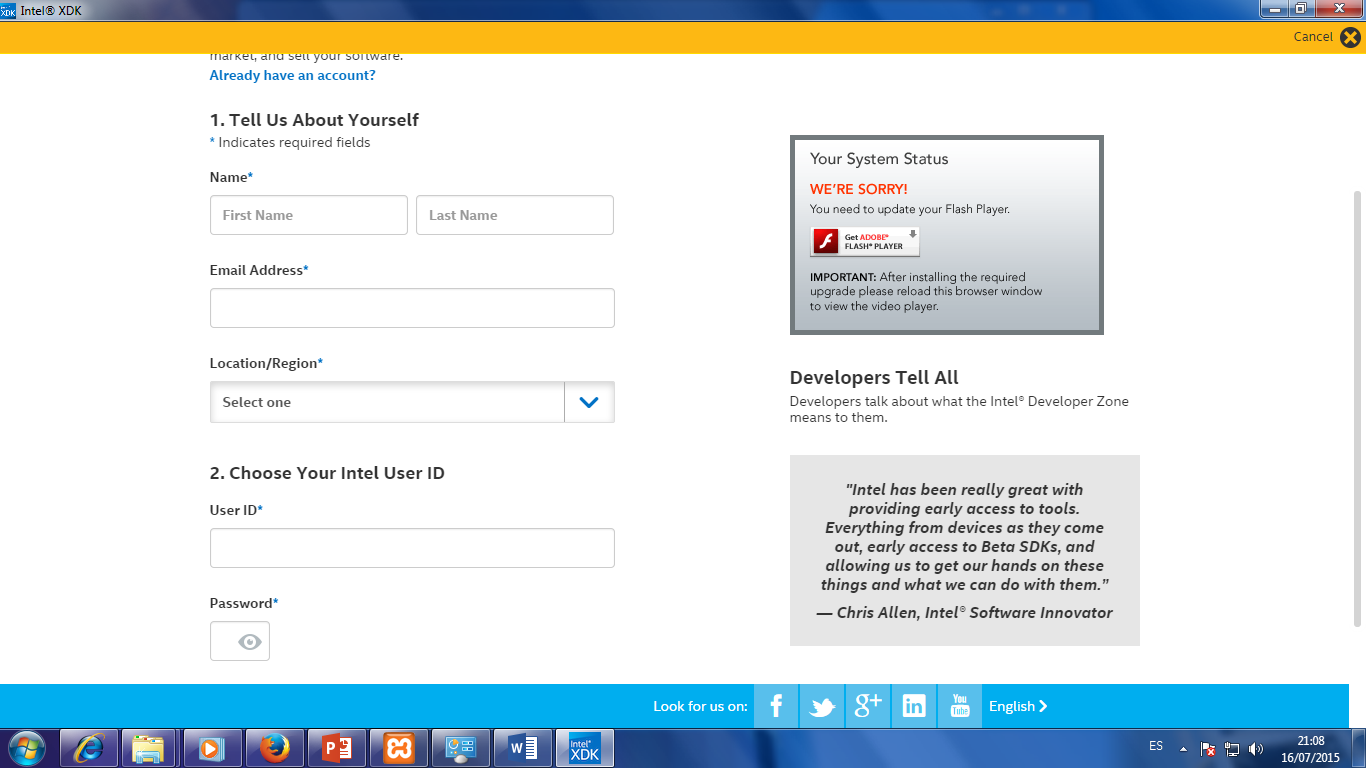


Y al abrir Intel XDK, podrán ver algo como esto, dependiendo de su versión y plataforma:



Y con esto, nos conectamos a internet y luego hacemos clic en sign Up

Aparece una ventana de dialogo para registrarse y crear una cuenta para posterior uso.



**3.- La primera app**

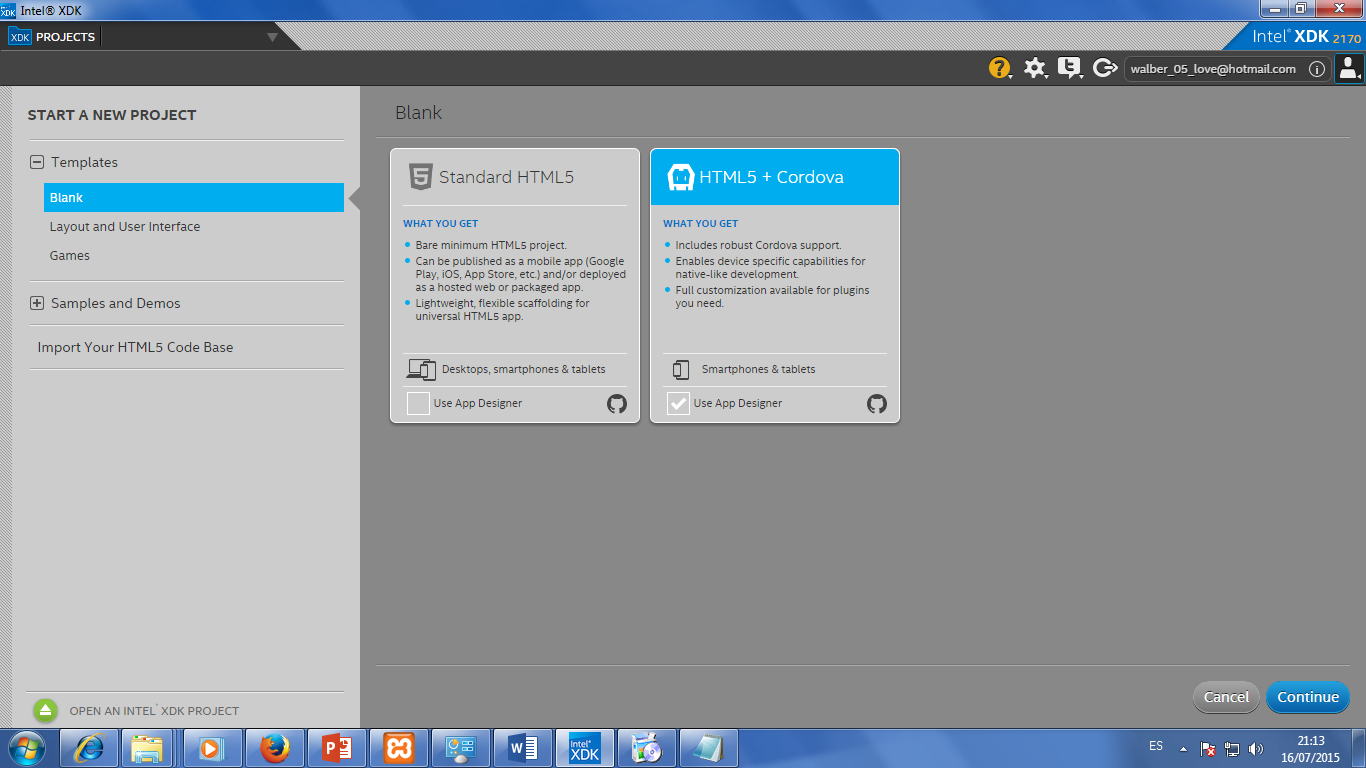
Luego de descargar e instalar, que mejor que meternos de lleno a hacer nuestra primera app, mediante la cual podremos entender como se estructuran los proyectos en Android para futuras referencias.

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**, o asistente de nuevos proyectos **Project**:

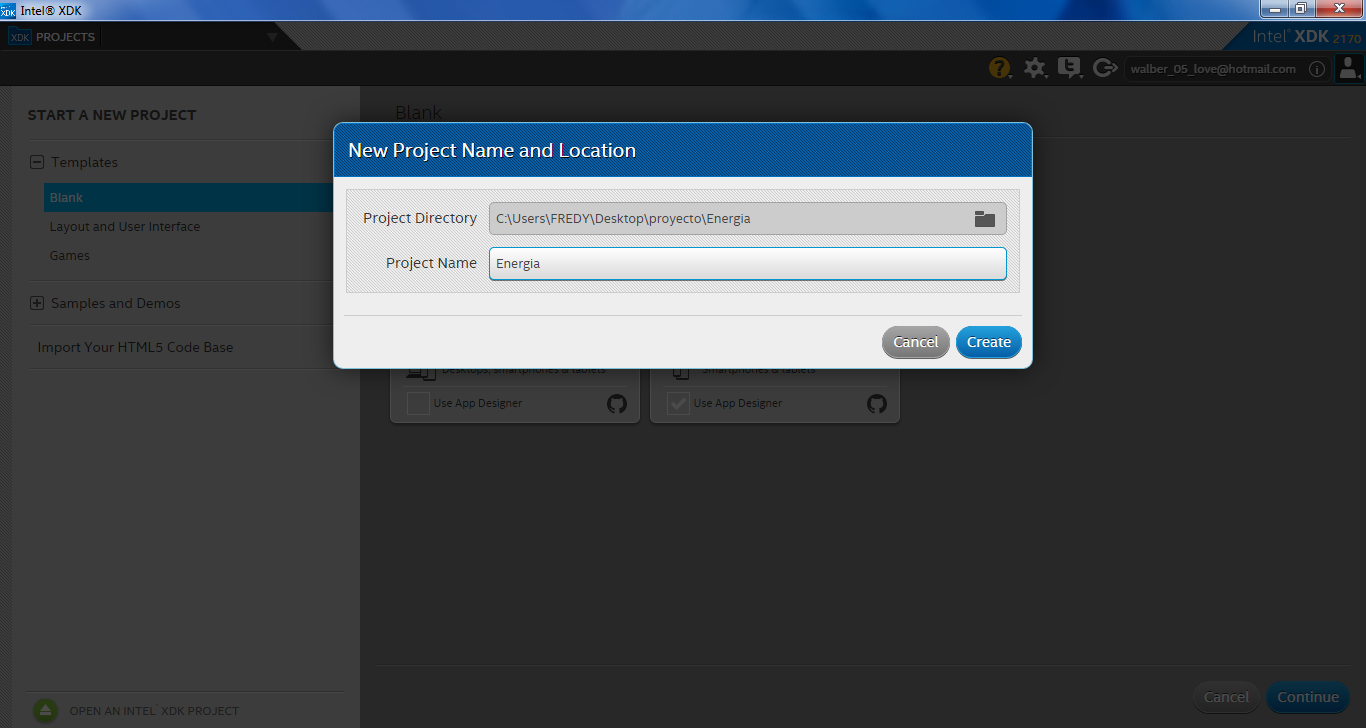


Al hacer click en Siguiente, llegaremos a la pantalla de configuración inicial de la nueva aplicación, en la que veremos algo como esto:



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

* **Project Directory:** Su nombre indica claramente que trata dela dirección 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 Intel XDK*(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.
* **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.



### ¿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).

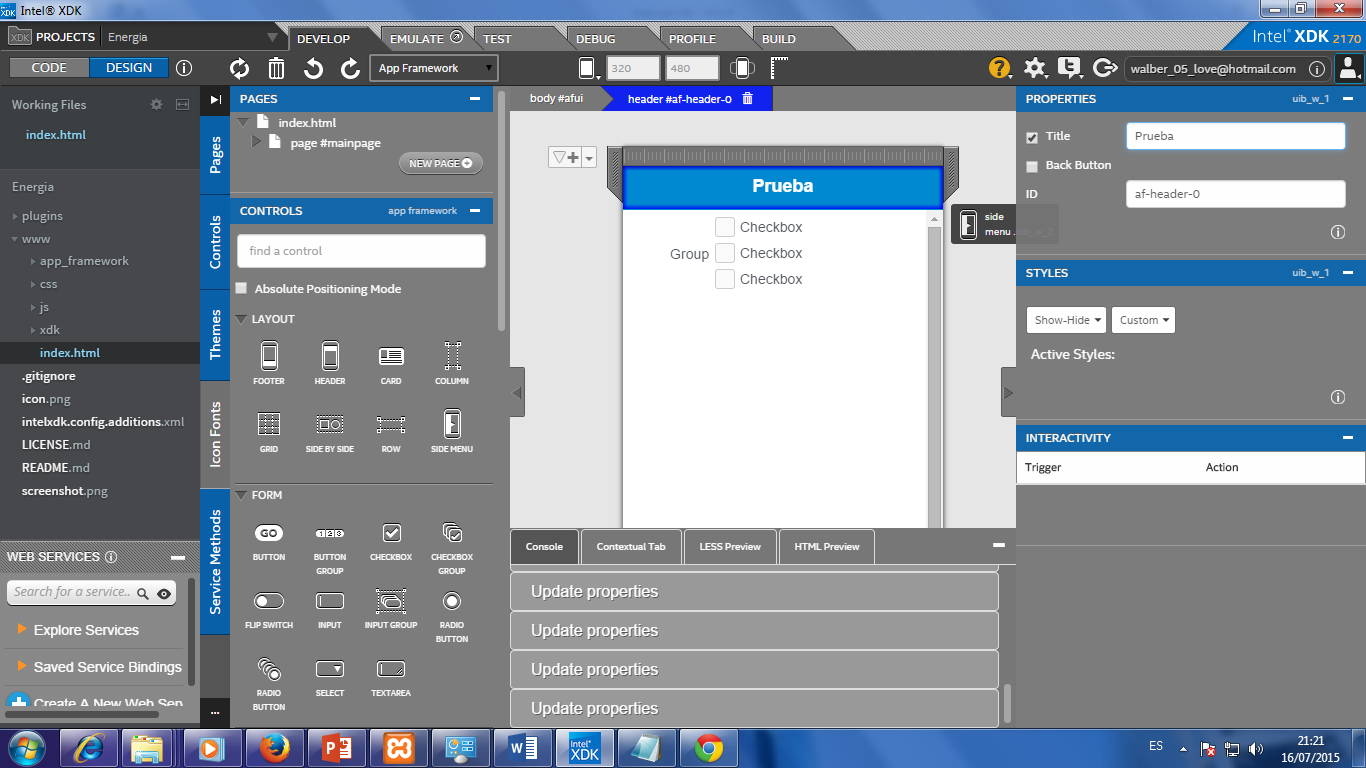
Tecnicamente, una "Activity" es una clase en Java que **hereda** desde la clase Activity de Android y ayuda a definir el comportamiento de una aplicación en distintas instancias. Al heredar desde la clase Activity, es posible manejar el ciclo de vida de la app y definir mediante código como queremos 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.

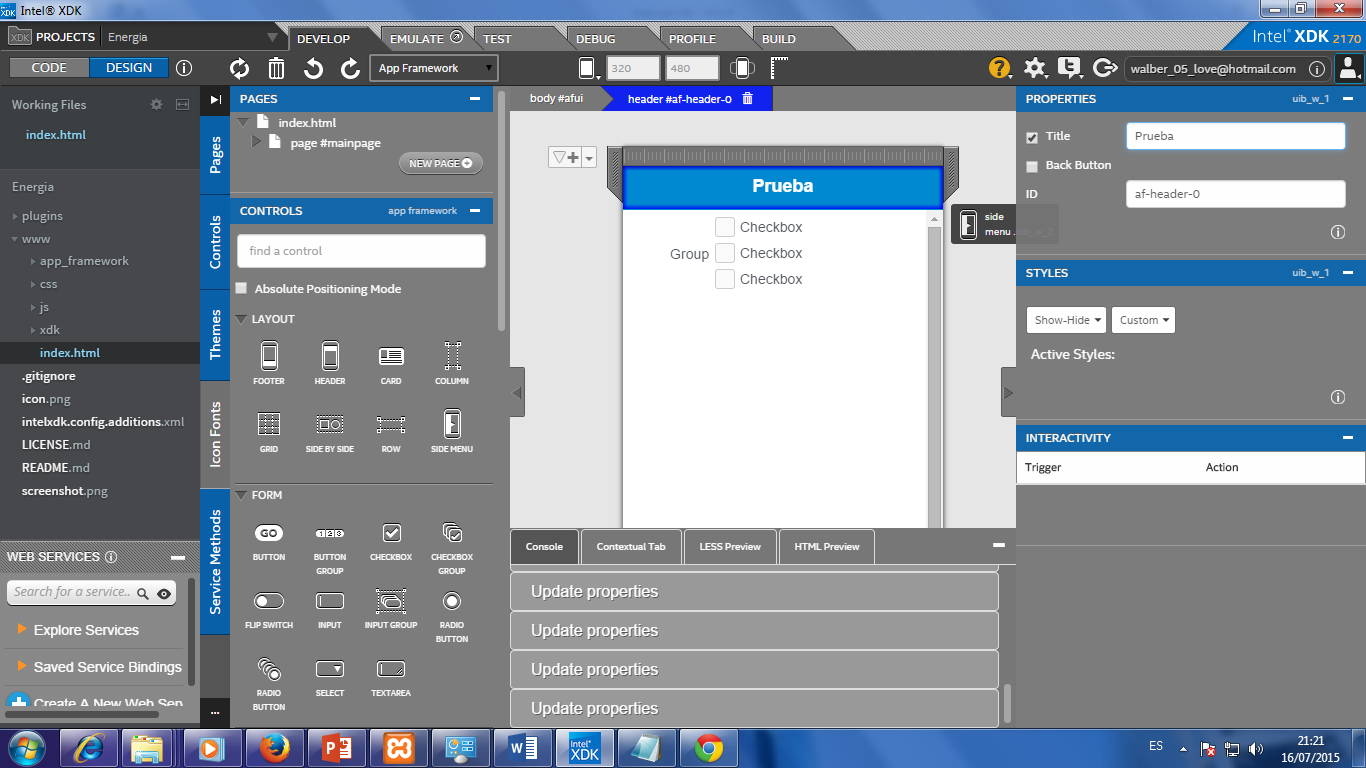
Si bien hay un ciclo de vida definido en cada actividad (desde que se inicia hasta que se cierra), en una actividad es posible crear una infinidad de métodos personalizados para el código de nuestra aplicación y también es factible cargar todas las vistas que queramos dentro de ella (aunque no al mismo 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).

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 pantalla vacía.



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:

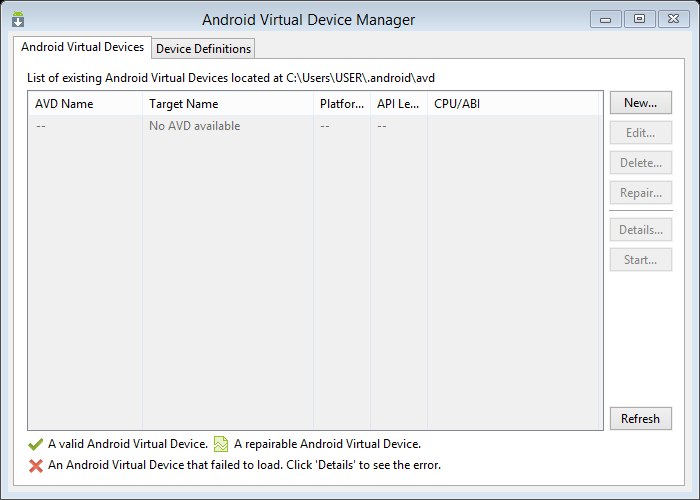
* Actividades de la app (marcando cual es la que se lanza al iniciar la app)
* Permisos solicitados por la app
* Versiones de Android soportadas
* Versión
* Ícono y nombre de la app
* Estilo

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:

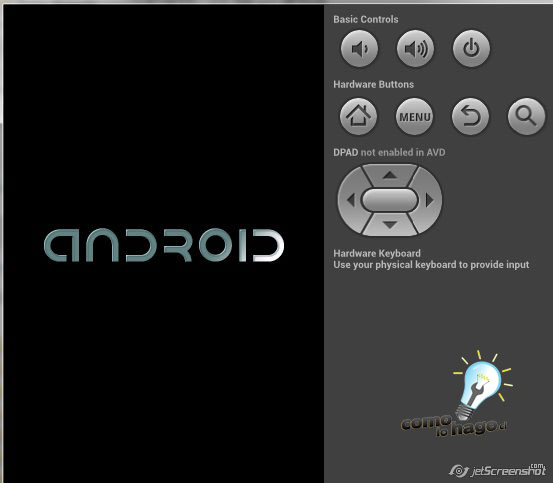
setContentView(R.layout.activity\_main)

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.

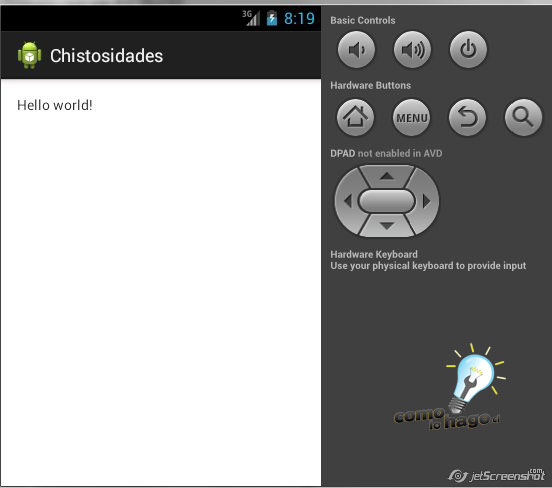
Teniendo esto en cuenta, al ejecutar esta app, nuestra primera aplicación, se va a cargar la vista correspondiente a activity\_main.xml y nos va a mostrar el texto que vimos hace un par de imágenes atrás. Así que para eso hacemos click derecho en el proyecto *(en el lado izquierdo, en el explorador de 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.



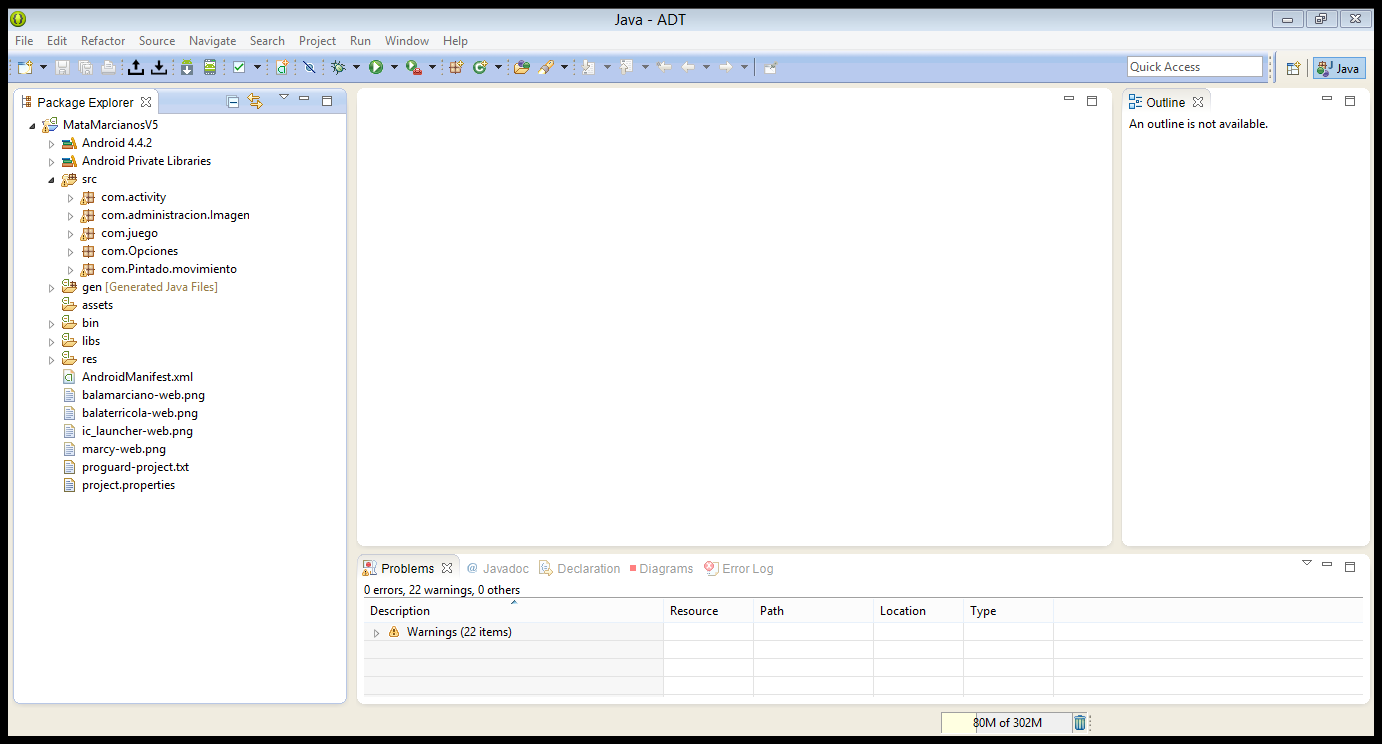




  Y así de simple tenemos ya en el emulador nuestra primera App.

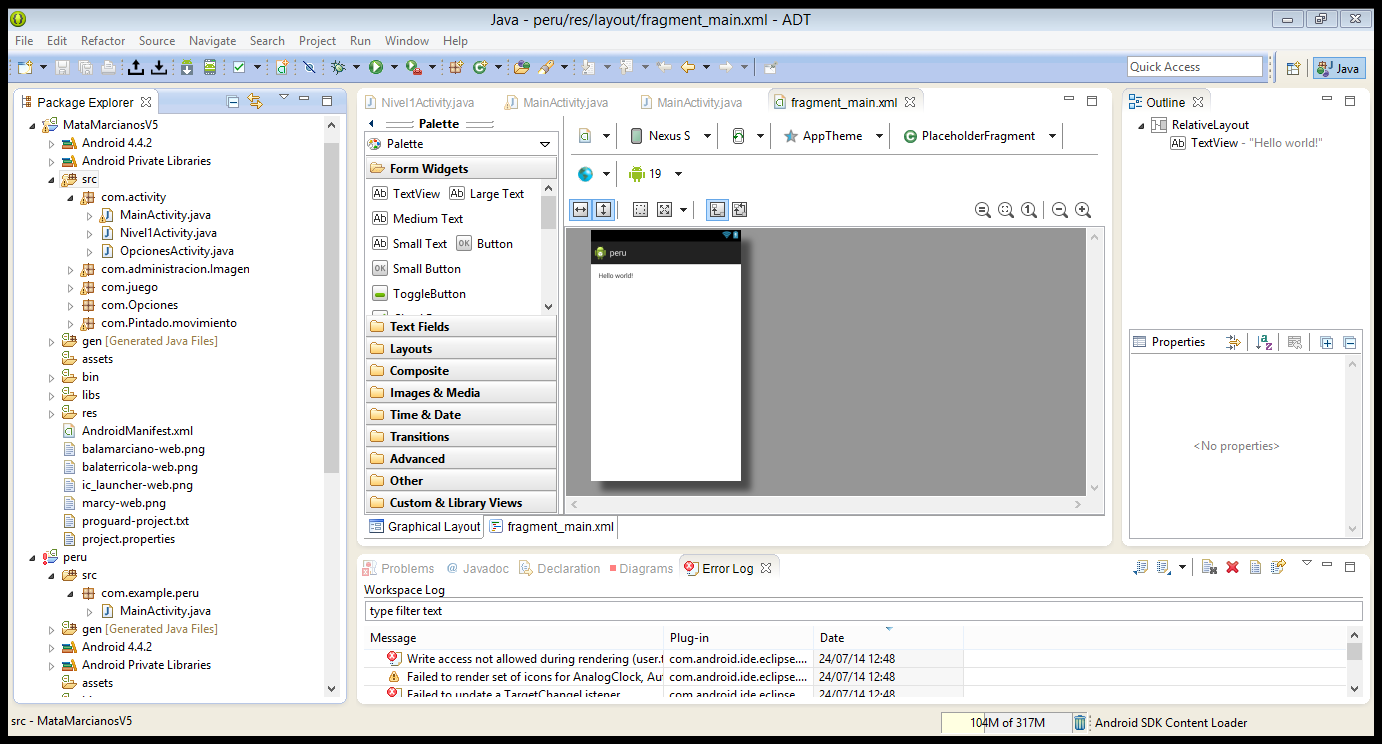
**Desarrollan la aplicación mata marcianos**

**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**

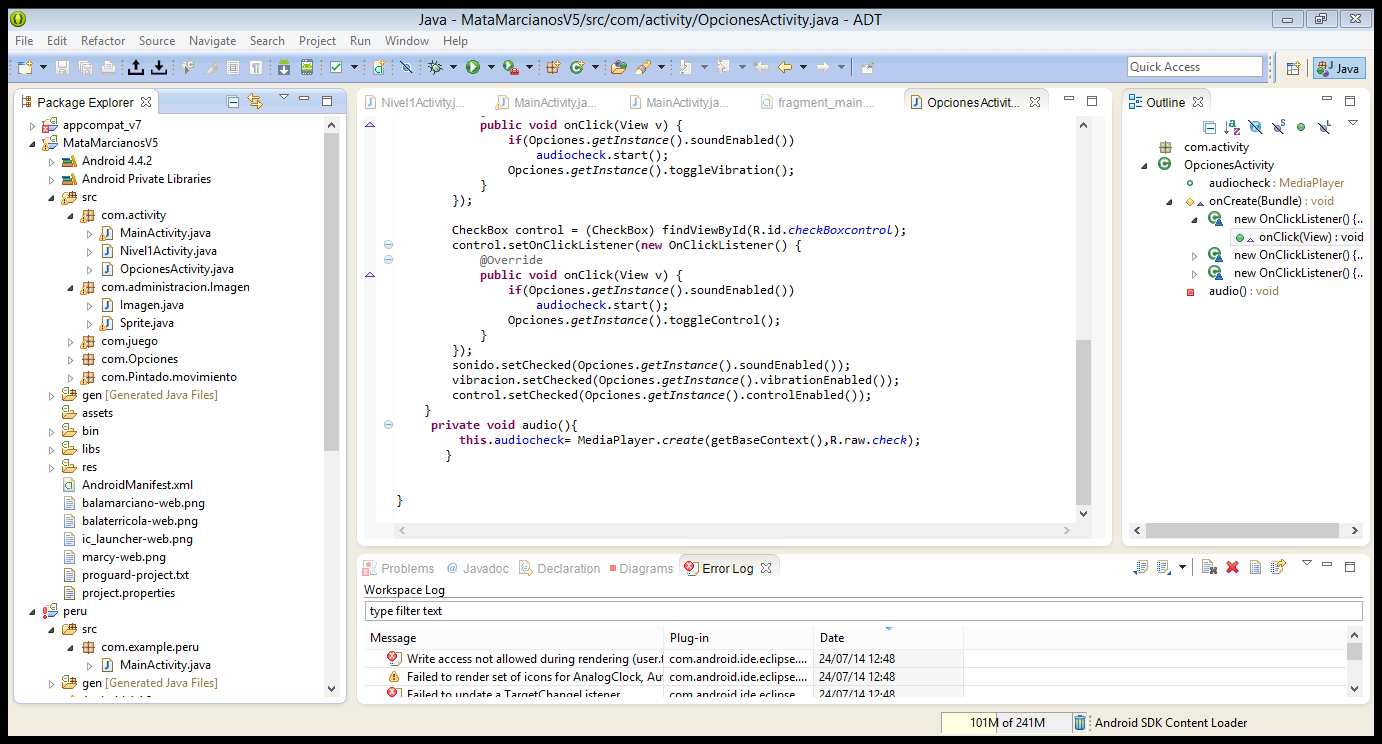
**La clase en donde se programó LAS configuración de juego como son: sonido,** direcciones de flechas.

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**

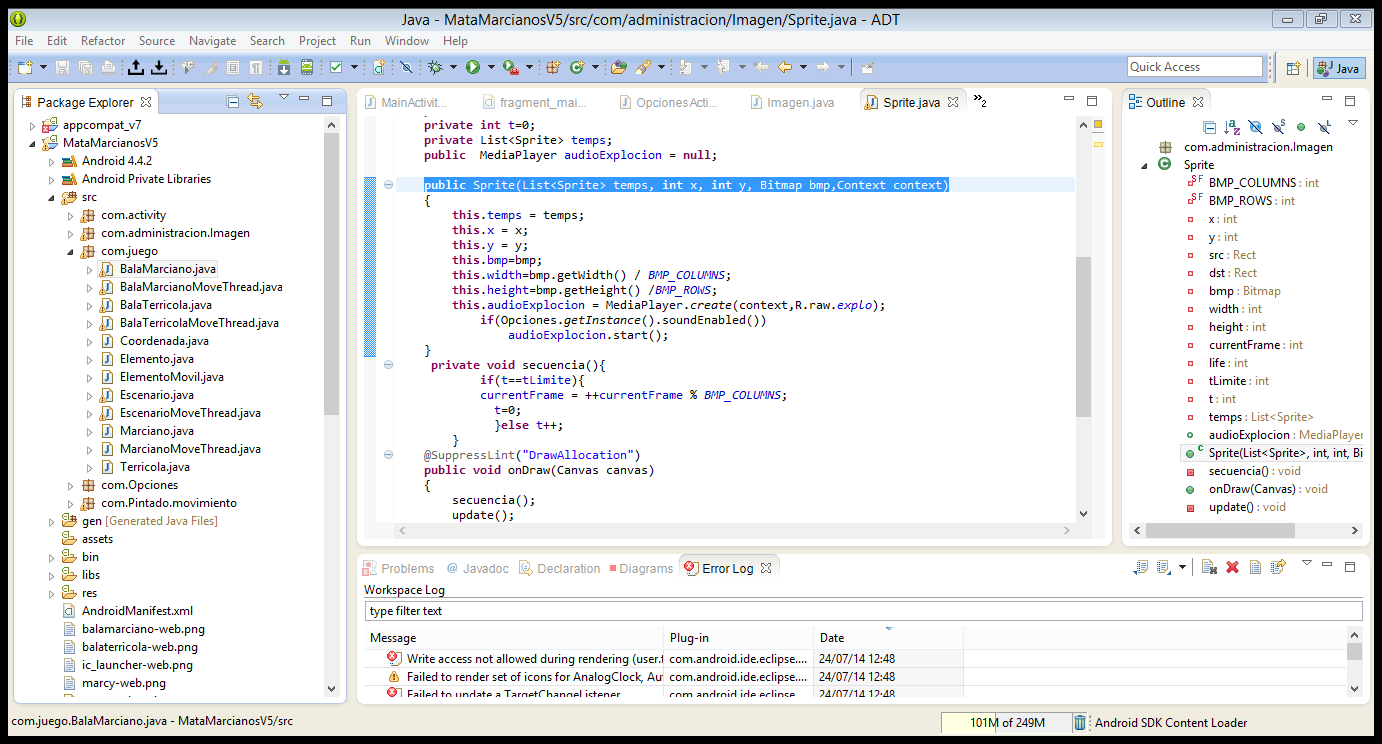
**public** Imagen(Coordenada origen, **int** ancho, **int** alto, Bitmap image)

**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**