

Introduccion a la Plataforma Android.



Manual del participante

2015



Índice

Índice	2
Formación de la comunidad de aprendizaje	4
Objetivo del curso	5
Objetivo	5
Introducción	5
¿Qué es Android?	5
Historia de Android	5
Estructura	6
Capa del Kernel (Roja)	7
Capa Librerías (Verde)	7
Android Runtime (Capa Amarilla)	7
Application Framework (Capa azul)	7
Aplicaciones (Capa Azul última)	8
REPASO DE JAVA	8
¿Qué es Java?	8
Tipos de Java	8
Pros de usar Java	9
Herramientas para su desarrollo	9
¿Cómo instalar Java en nuestras máquinas?	9
Ajustes de instalación	10
Hello Word Hola Mundo	10
¿Cómo compilar?	11
¿Cómo ejecutar?	11
Sintaxis básica	11
Modificadores	12
Comentarios	15
Herencia	16
Interfaces	16
Clases	16
Acceso a variables	18
Ejemplo	19
Tipos de datos primitivos	20
Short	21
Int	21
Long	21
Float	21
Double	22
Boolean	22
Char	22
Tipos de datos de referencia	22
Variables Locales	23
Variables de estáticas	25
Operadores aritméticos	27

Operadores relacionales	29
Operadores lógicos	30
La sentencia if	33
La sentencia if..else	34
La sentencia if..else if..else	35
La sentencia anidada if..else	36
La sentencia switch	37
El bucle while	39
El bucle do...while	40
El bucle for	41
Herencia	44
Relación ES-UN	44
¿Qué es una aplicación multihilo?	47
¿Cómo creamos una aplicación multihilo en Java? - Modo 1	47
.....	50
¿Cómo creamos una aplicación multihilo en Java? - Modo 2	50
Polimorfismo en Java	52
INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO	54
Asistente para la creación de una aplicación	55
Panorama general del proyecto	59
Previsualización del aspecto grafico	60
INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO (II)	62
La interfaz grafica	62
Archivo XML con la definición de la Interfaz Grafica	63
Recursos string	64
Código Java	66
INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO (III)	68
Creación de un emulador	69
Configuración de un equipo físico	69
Ejecución de la aplicación	71
INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO (IV)	75
Mejorando el desempeño	75
Intel x86 Emulator Accelerator	76
Activación de la virtualización desde el BIOS	77
Instalación del acelerador Intel	78
Instalación directa de la Aplicación	78
Bibliografía	80
Cibergrafía	80

Formación de la comunidad de aprendizaje

La formación de la comunidad de aprendizaje es un proceso que debe llevarse a cabo para iniciar cada uno de nuestros cursos.

Su finalidad es crear un clima propicio para la celebración de la actividad instruccional, es decir, generar un entendimiento previo entre el instructor y los participantes sobre los temas que se desarrollarán durante ésta, así como las estrategias educativas que se llevarán a cabo para lograr un mejor aprendizaje.

Un adecuado manejo de la comunidad de aprendizaje es un elemento fundamental para garantizar la satisfacción de uno de los clientes involucrados en la impartición de los cursos: **los participantes**.

Presentación del Instructor:

- ★ Nombre, profesión, años de experiencia como instructor, experiencia en la impartición del curso, o cursos similares o relacionados.

Alineación de expectativas:

- ★ El instructor recabará las expectativas de los participantes respecto al curso, con el fin de dejarles claro el objetivo del mismo.
- ★ En caso de que alguna expectativa no coincida con los temas que el curso contiene, el instructor dejará claro cuáles de las expectativas expresadas no serán cubiertas con el curso y porqué.
- ★ Las expectativas alineadas serán anotadas en hojas de rotafolio para su revisión al término del curso.
- ★ Durante el desarrollo del curso el instructor deberá cubrir las expectativas alineadas.

Presentación del objetivo del curso:

- ★ El instructor presentará a los participantes el objetivo del curso, aclarando dudas al respecto si las hubiese.

Reglas de oro:

- ★ El instructor promoverá el establecimiento de reglas por parte de los participantes que se observarán a través del curso; por lo que puede proponer: tiempo de tolerancia para iniciar las sesiones, respeto hacia los compañeros, participación de todos en técnicas y ejercicios grupales, etc.; se incluirán todos los puntos que los participantes consideren pertinentes.
- ★ Se anotarán los acuerdos en hojas de rotafolio y se colocarán en un espacio en el que sean visibles a lo largo de todo el curso.

Cumplimiento de expectativas

- ★ Al finalizar el curso el instructor deberá llevar a cabo una revisión de las expectativas alineadas que se anotaron en hojas de rotafolio al inicio del curso
- ★ Se revisará cada una de las expectativas alineadas palomeando las que hayan sido cumplidas, y el instructor explicará de qué manera se llevó a cabo tal cumplimiento.

Objetivo del curso

Objetivo

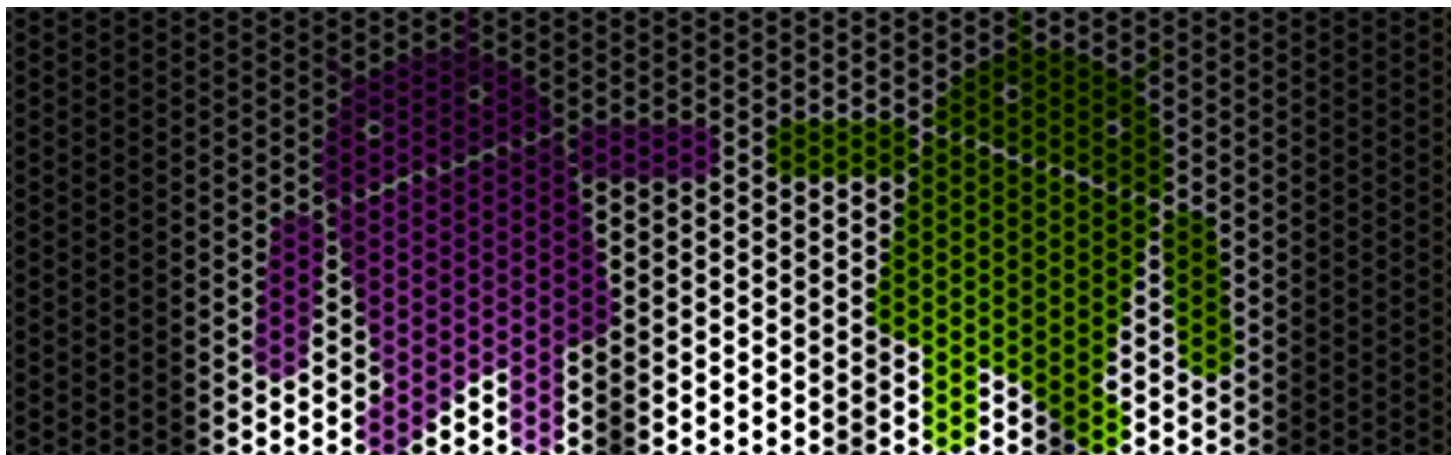
Al término del curso el participante tendrá nociones básicas de:

- Lenguaje de programación Java
- El IDE y sus extensiones para Android
- El SDK Android Studio

Introducción

¿Qué es Android?

La evolución de la tecnología va a paso veloz, Android es de las tecnologías que esta alcanzado a todos por el simple motivo de que se encuentra en los móviles. Android es un sistema operativo basado en Linux. La diferencia principal es que tiene módulos que responden a la pantalla táctil, eventos nativos del móvil. Se desarrolló por una compañía llamada Android, Inc. En 2005 Google adquiere la empresa para seguir trabajando en el mismo proyecto que después conociera la luz como un S.O. para móviles denominado finalmente como Android.



A finales de 2008 Septiembre-Octubre, sale a la venta el primer dispositivo móvil con Android.

Historia de Android

Android tiene una característica peculiar: las versiones tienen nombre de postres en inglés y cada versión que cambia, continúa de forma incremental en el alfabeto, es decir que si el primer nombre inicio con A, el siguiente con B, el siguiente C y así sucesivamente; ya veremos qué sucede cuando lleguen a la Z.

Hasta el día de hoy, que comienzo a escribir el manual Android para Desarrolloweb.com, tenemos la versión 4.4 KitKat.

Demos un repaso a las Versiones.

Versión 1.0 Apple Pie - Salió en septiembre del 2008.

Versión 1.1 Banana Bread - Salió en febrero 2009.

Versión 1.5 Cup Cake - Salió en abril 2009

Versión 1.6 Donut - Salió en septiembre 2009

Versión 2.0 Eclair - Salio en octubre 2009

Versión 2.2 Froyo - Salió en mayo 2010

Versión 2.3 Gingerbread - Salió en diciembre 2010

Versión 3.0 Honeycomb - Salió en febrero 2011

Versión 4.0 Ice Cream Sandwich - Salió en octubre 2011

Versión 4.1 Jelly Bean - Salió en julio 2012

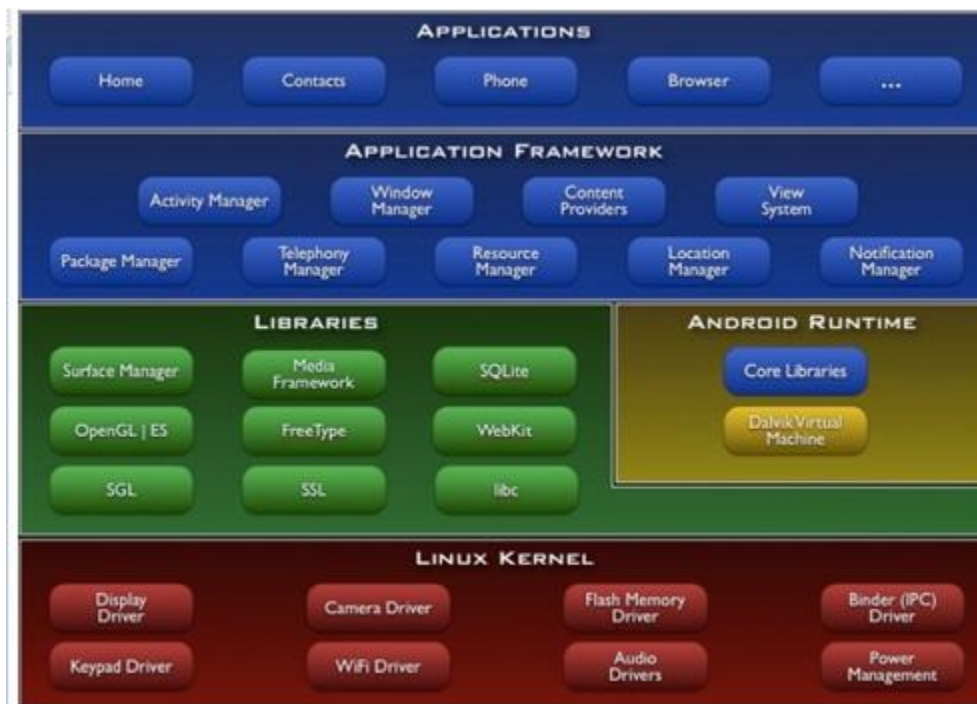
Versión 4.4 KitKat - Salió en octubre 2013

Para más información, puedes consultar este enlace: <http://www.android.com/versions/kit-kat-4-4/>

Estructura

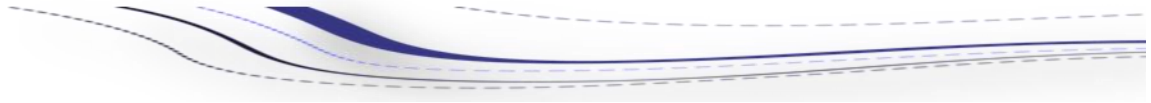
Ya mencionamos que Android está basado en Linux. Para ser más específicos, hablamos del kernel. Android utiliza como base el kernel de Linux. Esto no significa que por estar basado en el algo que se desarrolló en Linux funcione para Android, por ejemplo Android no tiene soporte glibc.

Ahora vamos a darle un vistazo a la estructura:



Tenemos esta estructura:

- Capa Roja, Kernel.



- Capa Verde, Librerías.
- Capa Amarilla, Android runtime.
- Capa Azul, application Framework
- Capa Azul Última, Application.

Vamos a conocer cada una de ellas:

Capa del Kernel (Roja)

Aquí tenemos el corazón de Android: el manejo de memoria, procesos, *drivers*, etc. Aquí es donde se da la comunicación con el *hardware*. Esto nos sirve para no estar peleando con los fabricantes de cada móvil, nos ayuda a solo usar la "cámara" y no tener que saber cómo funciona la cámara del fabricante X, fabricante Y; solamente hacemos lo que nos interesa, que sería usar la cámara y listo. Además de eso, aquí se administran los recursos del celular, memoria, energía...

Capa Librerías (Verde)

Esta capa tiene las librerías nativas de Android, están escritas en C o C++ y tienen tareas específicas.

- **Surface manager:** Gestión del acceso a la pantalla.
- **Media Framework:** Reproducción de imágenes, audio y vídeo.
- **SQLite:** BD
- **Webkit,** Navegador.
- **SGL:** Gráficos 2D.
- **OpenGL:** Gráficos 3D.
- **Freetype:** Renderizar vectores o imágenes.

Android Runtime (Capa Amarilla)

Esta capa amarilla no se considera al 100% una capa. Lo que es muy importante comentar es que aquí se encuentra Dalvik, la máquina virtual de Android, que no es lo mismo que la Java Virtual Machine. Esto quiere decir que cuando compilamos en Java lo que se genera solamente va a funcionar en la JVM, porque Dalvik es una máquina virtual, pero de Android, así que el ByteCode que genera Java es inservible para Dalvik.

Algunas de las características de Dalvik son:

- Trabaja en entorno con restricción de memoria y procesador.
- Ejecuta el formato .dex.
- Convierte .class en .dx.

Application Framework (Capa azul)

Esta capa es la más visible para el desarrollador, ya que la mayoría de los componentes que forman parte del desarrollo los vamos a encontrar aquí.

- **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**- Administra dependiendo de cómo le indiquemos algunos contenidos, puede ser información que necesitamos la encapsule para enviar o compartir.
- **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 Manger**- Administra los paquetes y nos permite el uso de archivos en otros paquetes.
- **Telephony Manager**- Administra lo que tiene que ver con la telefonía, llamadas, mensajes.
- **Resource Manager**- Administra recursos de la aplicación, como los xml, imágenes, sonido.
- **Location Manager**- Gestiona la posición geográfica.
- **Sensor Manager**- Gestiona los sensores que tenga el dispositivo.
- **Cámara**- Administra la cámara.
- **Multimedia**- Administra lo referente a audio, video y fotos.

Aplicaciones (Capa Azul última)

Aquí tenemos las aplicaciones que vienen en el dispositivo, por ejemplo: el gestor de correos, los mensajes, *elmarket*, etc.

REPASO DE JAVA

¿Qué es Java?

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva mucho de C y C++, pero tiene menos facilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente. Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

Tipos de Java

Podemos encontrar tres tipos de ediciones de Java:

- **Java SE o Standard Edition:** es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java.
- **Java EE o Enterprise Edition:** es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java que permite utilizar arquitecturas

de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

- **Java ME o Micro Edition:** es una especificación de un subconjunto de la plataforma Java orientada a proveer una colección certificada de APIs de desarrollo de software para dispositivos con recursos restringidos. Está orientado a productos de consumo como PDAs, teléfonos móviles o electrodomésticos.

El curso se va a enfocar en Java SE.

Pros de usar Java

- Es orientado a objetos.
- Es simple de usar como lenguaje de programación.
- Es portable, corre en cualquier plataforma, independientemente de cual sea.
- Es seguro.
- Es robusto, ya que enfatiza los errores en el proceso de compilación.
- Es multi-hilo, es decir, ejecuta varios procesos a la vez en un solo programa.
- Posee un alto desempeño y capacidad..
- Está diseñado para ser usado como sistemas distribuidos.
- Es dinámico, ya que está diseñado para adaptarse a cualquier ambiente de desarrollo.

Herramientas para su desarrollo

Puedes usar cualquier editor de texto que sea de tu agrado pero te recomiendo el uso de IDEs muy poderosos como lo son:

- [Eclipse.](#)
- [Netbeans.](#)
- [BlueJ.](#)
- [JCreator.](#)
- [IntelliJ.](#)

A mí en particular me gusta netbeans y es el que voy a estar usando en el curso.

¿Cómo instalar Java en nuestras maquinas?

Lo primero que tenemos que hacer es descargar gratis el Java SE Development Kit (JDK) de la página de [Descargas de Oracle.](#) En esta vas a tener que descargar la última versión del JDK disponible para tu sistema operativo. En el tiempo de este tutorial es la versión *7u40*. Una vez que hayas descargado el JDK, deberás ejecutarlo y el instalador es muy lineal y sencillo de usar. No existen muchas configuraciones necesarias en este punto.

Ajustes de instalación

Una vez que hayas instalado Java en tú máquina es necesario hacer unos ajustes a las variables de ambiente disponible en el sistema, todo va a depender de que sistema operativo poseas:

Windows 2000/XP:

Asumiendo que instalaste Java en el directorio c:\Program Files\java\jdk debes hacer lo siguiente:

- Click derecho en "Equipo" o "Mi Computador" y seleccionar "Propiedades".
- Click en "Variables de ambiente" o "Environment variables" bajo la pestaña de "Avanzados".
- Ahora, altera la variable "Path" para que contenga la dirección de Java, es decir, si actualmente esta seteado para 'C:\WINDOWS\SYSTEM32', deberás cambiarlo por 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

Linux, UNIX, Solaris, FreeBSD:

Tu variable de ambiente PATH debe estar asignada hacia donde los binarios de Java fueron instalados.

Ejemplo, si tu estas usando bash como tu shell, deberás agregar la siguiente linea al final de tu archivo `.bashrc`: 'export PATH=/path/to/java:\$PATH'.

Hello Word | Hola Mundo

Ya que tenemos instalado Java en nuestras maquinas es hora de que desarrollemos nuestro primer programa y nada más fácil que realizar un "Hola Mundo" para probar la instalación.

La manera más fácil para hacerlo es haciendo uso de un editor de texto de tu preferencia (para este ejemplo tan sencillo no es necesario usar un IDE. No te preocupes que más adelante si lo vamos a estar utilizando).

Con el editor de texto vamos a crear un archivo el cual denominaremos `HolaMundo.java`.

Una vez que lo hayamos creado agregamos el siguiente código:

```
public class HolaMundo {

    /* This is my first java program.
    * This will print 'Hello World' as the output
    */

    public static void main(String []args) {

        System.out.println("Hola Mundo"); // imprime Hola Mundo
    }
}
```

No te preocupes si no entiendes muy bien cómo funciona en los próximos capítulos cubriremos todos los aspectos del lenguaje.

¿Cómo compilar?

Para compilar el código que acabamos de hacer es necesario:

- Abrir la consola o el terminal.
- Navegar hasta el directorio donde creamos `HolaMundo.java`.
- Tipeamos lo siguiente para compilar:

```
$ javac HolaMundo.java
```

¿Cómo ejecutar?

Para probar el `HolaMundo.java` que creamos anteriormente vamos a necesitar hacer uso de la consola o terminal.

Abre una consola y tipea:

```
$ java HolaMundo
```

Debería aparecerte **"Hola Mundo"** en la pantalla.

Sintaxis básica

Cuando desarrollamos un programa en Java es muy importante tener en cuenta los siguientes puntos:

- **Sensibilidad a mayúsculas** - Java es sensible a mayúsculas, lo que significa que si poseemos un identificador llamado *Hola* y *hola* tendrían valores diferentes.
- **Nombres de las clases** - Para todos los nombres de clases la primera letra debe estar en mayúsculas. Si quieres usar varias palabras para formar un nombre, la primera letra de cada palabra interior debe estar en mayúsculas.

```
class miPrimeraClase
```

- **Nombres de los métodos** - Todos los nombres de los métodos deben comenzar con una letra minúscula. Si se quiere usar varias palabras para formar el nombre de un método, la primera letra de cada palabra interior debe estar en mayúsculas.

```
public void miPrimerMetodo()
```

- **Nombre de archivo del programa** - El nombre del archivo de programa debe coincidir exactamente con el nombre de la clase.

Al guardar un archivo, debemos guardarlo con el nombre de clase (Recuerda que Java distingue entre mayúsculas y minúsculas) y añadir '.java' al final del nombre (si el nombre del archivo y el nombre de clase no coinciden el programa no compilará).

Ejemplo: Supongamos que 'MiPrimerPrograma' es el nombre de la clase. Luego el archivo debe ser guardado como 'MiPrimerPrograma.java'

- **public static void main(String args[])** - la ejecución de un programa en Java se inicia desde el método `main()`, por lo cual es una parte obligatoria del desarrollo.

Identificadores

Todos los componentes en Java requieren nombres. Los nombres usados para las clases, variables y métodos se denominan identificadores.

Existen varios puntos que debemos recordar acerca de los identificadores:

- Todos los identificadores deben comenzar con una letra (A a Z o de la a a z), el carácter de moneda (\$) o un guión bajo (_).
- Una palabra clave no se puede utilizar como un identificador .
- Los identificadores distinguen entre mayúsculas y minúsculas.
- Ejemplos de identificadores legales: edad, \$salario, _valor, __1_valor.
- Ejemplos de identificadores ilegales : 123abc , -salario.

Modificadores

Al igual que otros lenguajes de programación, en Java es posible modificar las clases, métodos, etc..., mediante el uso de modificadores. Existen dos categorías de modificadores:

- **De acceso:** default, public , protected, private.
- **De no acceso:** final, abstract.

Tranquilo si estas un poco perdido mas adelante vamos a ver estos puntos a mayor detalle.

Variables

En Java existen los siguientes tipos de variables:

- Variables locales.
- Variables de clase (variables estáticas).
- Variables de instancia (variables no estáticas).

Tranquilo si estas un poco perdido más adelante vamos a ver estos puntos a mayor detalle.

Arrays

Los arrays son objetos que almacenan múltiples variables del mismo tipo. Sin embargo, un array en sí es un objeto.

Tranquilo en los próximos capítulos veremos cómo declararlo, construirlo e inicializarlo.

Enum

Las enumeraciones se introdujeron en java 5.0. Estas restringen una variable a tener solo unos valores predefinidos. Con el uso de enumeraciones que es posible reducir el número de errores en el código.

Por ejemplo, si tenemos queremos hacer una aplicación en donde podamos ordenar tipos de jugos frescos, sería posible limitar el tamaño del jugo en pequeño, mediano y grande con la ayuda de los enum.

```
class Jugo {  
  
    enum JugoTamano { PEQUENO, MEDIANO, GRANDE }  
  
    JugoTamano tamano;  
}  
  
public class JugoPrueba {  
  
    public static void main(String args[]){  
  
        Jugo jugo = new Jugo();  
  
        jugo.tamano = Jugo.JugoTamano.MEDIANO ;  
  
        System.out.println("Tamaño del jugo: " + jugo.tamano);  
    }  
}
```

Si compilamos y ejecutamos el programa anterior debería devolvernos:

```
Tamaño del jugo: MEDIANO
```

Los enum se pueden declarar dentro o fuera de una clase.

Tranquilo si estas un poco perdido más adelante vamos a ver estos puntos a mayor detalle.

Palabras clave

La siguiente lista muestra las palabras reservadas de Java. Estas palabras reservadas no se pueden utilizar como constante o variable o cualquier otro nombre de identificador.

- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char

- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static

- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

Comentarios

Java soporta una o varias líneas de comentarios. Es muy similar a los comentarios que podemos encontrar en C y C++. Todos los caracteres disponibles dentro de cualquier comentario son ignorados por el compilador.

```
public class MiPrimerPrograma{  
  
    /* Este es mi primer programa  
    * Esto va a imprimir "Hola Mundo"  
    * Un ejemplo de un comentario multi-linea  
    */  
  
    public static void main(String []args){  
        // Un ejemplo de un comentario de una linea  
        /* Este también es un ejemplo de un comentario de una linea */  
        System.out.println("Hola Mundo");  
    }  
}
```

```
}
```

Herencia

En Java, las clases pueden ser derivados de clases. Básicamente, si necesitamos crear una nueva clase y tenemos una clase que tiene una parte del código que necesitamos, entonces es posible derivar la nueva clase a partir del código ya existente.

Este concepto permite reutilizar los campos y métodos de la clase existente sin tener que volver a escribir el código de una nueva clase. En este escenario la clase existente se llama la superclase y la clase derivada se llama la subclase.

Interfaces

En Java, una interfaz se puede definir como un contrato entre los objetos sobre la forma como se van a comunicar entre sí. Las interfaces juegan un papel fundamental cuando se trata de el concepto de herencia.

Una interfaz define los métodos que una clase derivada (subclase) debe utilizar. Pero la puesta en práctica de los métodos es totalmente de la subclase.

Clases

Una clase es una plantilla de la que se crean los objetos individuales.

A continuación vamos a ver un ejemplo de una clase en Java:

```
public class Carro{  
    String marca;  
    int kilometraje;  
    String color;  
  
    void encender(){  
    }  
  
    void acelerar(){  
    }  
  
    void apagar(){  
    }  
}
```

Una clase puede contener cualquiera de los siguientes tipos de variables:

- **Variables locales:** Las variables definidas dentro de los métodos, los constructores o los bloques se denominan variables locales. Se declara la variable y se inicializa en el método y la variable será destruida cuando el método se ha completado.
- **Variables de instancia:** Las variables de instancia son variables dentro de una clase, pero fuera de cualquier método. Estas variables se crean instancias cuando se carga la clase. Las variables de instancia se puede acceder desde el interior de cualquier método, constructor o bloques de esa clase en particular.
- **Variables de clase:** Las variables de clase son variables declaradas dentro una clase y fuera de cualquier método. En el ejemplo anterior **marca**, **kilometraje** y **color** son variables de clase Carro.

Una clase en Java puede tener cualquier número de métodos para acceder o modificar el comportamiento de dicha clase. En el ejemplo anterior **encender**, **acelerar** y **apagar** son métodos de clase Carro.

Una vez que tenemos una idea general de que es una clase en Java y cuales son sus características es importante revisar los siguientes aspectos:

Constructores

Cuando se discute acerca de las clases, uno de los temas más importantes tópicos serian los constructores. Cada clase tiene un constructor. Si no escribimos explícitamente un constructor para una clase el compilador de Java genera un constructor predeterminado para esa clase.

Cada vez que se crea un nuevo objeto, se invocará al menos un constructor. La regla principal de los constructores es que ellos deben tener el mismo nombre que la clase. Como dato importante una clase puede tener más de un constructor.

Vemos un ejemplo:

```
public class Carro{  
    public carro(){  
    }  
  
    public carro(String marca){  
        // El constructor tiene solo un parametro, en este caso marca  
    }  
}
```

Observemos que en el ejemplo anterior tenemos dos constructores el primero que es un constructor sencillo en el cual podemos inicializar variables de la clase con los valores que nosotros queramos y el segundo es un constructor el cual acepta un parámetro, es decir, para poder instanciar un objeto de esta clase con este constructor siempre vamos a tener que pasarle el nombre de la marca.

¿Cómo crear un objeto?

Como se mencionó anteriormente, una clase proporciona los planos de objetos. Así que, básicamente, un objeto se crea de una clase. En Java, la palabra clave **new** se utiliza para crear nuevos objetos. Existen tres pasos al crear un objeto de una clase:

- **Declarar:** Debemos declarar una variable con su nombre y con el tipo de objeto que va a contener.
- **Instanciar:** La palabra clave **new** se utiliza para crear el objeto.
- **Inicialización:** La palabra clave **new** va seguida de una llamada a un constructor. Esta llamada inicializa el nuevo objeto.

Si seguimos el modelo del ejemplo anterior:

```
public class Carro{

    public carro(String marca){

        // El constructor tiene solo un parametro, en este caso marca

        System.out.println("La marca es : " + marca );

    }

    public static void main(String []args){

        // Creamos la variable carro

        Carro miCarro = new Carro( "Ford" );

    }

}
```

Observemos que poseemos un constructor el cual recibe un parametro, en este caso la marca del carro, el cual va a imprimir el nombre de la marca cada vez que inicializemos un objeto de la clase carro.

A su vez con `Carro miCarro = new Carro("Ford")` estamos cumpliendo los pasos que explicamos anteriormente ya que declaramos una variable llamada `miCarro` de la clase `Carro`, la instanciamos al hacerle **new** y la inicializamos al llamar al constructor con `Carro("Ford")`.

Si compilamos el condigo anterior obtenemos:

```
La marca es : Ford
```

Acceso a variables

Se accede a las variables y métodos de instancia a través de los objetos creados.

Para acceder a la instancia de una variable la ruta de acceso completa debe ser el siguiente:



```
/* Primero creamos un objeto */
```

```
Objeto = new Constructor();
```

```
/* Ahora llamamos a la variable de la clase de la siguiente manera */
```

```
Objeto.nombreDeLaVariable;
```

```
/* También podemos acceder al método de la clase */
```

```
Objeto.nombreDelMetodo();
```

Ejemplo

Veamos un ejemplo que recopile todo lo que vimos hasta ahora:

```
public class Carro{
```

```
    int kilometraje;
```

```
    public Carro(String marca){
```

```
        // El constructor tiene solo un parametro, en este caso marca
```

```
        System.out.println("La marca es : " + marca );
```

```
    }
```

```
    public void setKilometraje( int kilometraje ){
```

```
        this.kilometraje = kilometraje;
```

```
    }
```

```
    public int getKilometraje( ){
```

```
        System.out.println("El kilometraje es : " + kilometraje );
```

```
        return this.kilometraje;
```

```
    }
```



```
public static void main(String []args){  
    /* Creación */  
    Carro miCarro = new Carro( "Ford" );  
  
    /* Seteamos el kilometraje del carro */  
    miCarro.setKilometraje( 2000 );  
  
    /* Obtenemos el kilometraje del carro */  
    miCarro.getKilometraje( );  
  
    /* También podemos acceder a la variable de la clase */  
    System.out.println("Valor variable : " + miCarro.kilometraje );  
}  
}
```

Observemos lo siguiente: tenemos una clase llamada **Carro**, la cual posee un constructor, dos métodos para modificar los valores de las variables de la clase (`getKilometraje` y `setKilometraje`). Y por último tenemos un programa el cual instancia una variable de la clase `Carro`, le asigna el kilometraje de 2000 a ese objeto y luego lo imprime en la consola.

Si compilamos el ejemplo obtenemos:

La marca es : Ford

El kilometraje es : 2000

Valor variable : 2000

Tipos de datos primitivos

Hay ocho tipos de datos primitivos soportados por Java. Los tipos de datos primitivos están predefinidos por el lenguaje y nombrados por una palabra clave. Veamos ahora en detalle acerca de los ocho tipos de datos primitivos.

Byte

- Tipo de datos Byte es un entero de 8 bits.
- El valor mínimo es -128 (-2^7).
- El valor máximo es 127 (inclusive) ($2^7 - 1$).
- El valor por defecto es 0.

- Tipo de datos Byte se utiliza para ahorrar espacio en grandes conjuntos, sobre todo en el lugar de los números enteros, ya que un byte es cuatro veces más pequeño que un **int**.
- Ejemplo: `byte a = 100`, `byte b = -50`

Short

- Tipo de datos Short es un entero de 16 bits.
- El valor mínimo es -32,768 (-2^{15}).
- El valor máximo es de 32.767 (inclusive) ($2^{15} - 1$).
- Tipo de datos Short también se puede utilizar para ahorrar memoria como tipo de datos byte. Un tipo de dato **short** es 2 veces más pequeño que un **int**.
- El valor por defecto es 0.
- Ejemplo: `short s = 10000`, `short r = -20000`

Int

- Tipo de datos int es un entero de 32 bits.
- El valor mínimo es -2147483648 (-2^{31}).
- El valor máximo es 2147483647 (inclusive) ($2^{31} - 1$).
- Int. se utiliza generalmente como el tipo de datos predeterminado para los valores enteros a menos que exista una preocupación acerca de la memoria.
- El valor por defecto es 0.
- Ejemplo: `int a = 100000`, `int b = -200000`

Long

- Tipo de datos Long es un entero de 64 bits.
- El valor mínimo es -9223372036854775808 (-2^{63}).
- El valor máximo es 9223372036854775807 (inclusive) ($2^{63} - 1$).
- Este tipo se utiliza cuando se necesita una gama más amplia que int.
- El valor por defecto es 0.
- Ejemplo: `long a = 100000`, `int b = -200000`

Float

- El Float es un dato de coma flotante de precisión simple de 32 bits.
- Float se utiliza principalmente para ahorrar memoria en grandes arrays de números.
- El valor por defecto es 0,0 f.
- Ejemplo: `float f1 = 234.5f`

Double

- El doble es un dato de coma flotante de doble precisión de 64 bits.
- Este tipo de datos se utiliza generalmente como el tipo de datos predeterminado para valores decimales, en general, la opción por defecto.
- El valor por defecto es 0.0 D.
- Ejemplo: `doble d1 = 123,4`

Boolean

- Boolean representa un bit de información.
- Sólo hay dos posibles valores: true y false.
- Este tipo de datos se utiliza para indicadores simples que hacen un seguimiento de condiciones.
- El valor predeterminado es falso.
- Ejemplo: `boolean a = true`

Char

- Char es un carácter Unicode de 16 bits.
- El valor mínimo es '\u0000' (o 0).
- El valor máximo es '\uffff' (o 65.535 inclusive).
- Tipo de datos char se utiliza para almacenar cualquier carácter.
- Ejemplo: `char letra = 'A'`

Tipos de datos de referencia

- Las variables de referencia se crean mediante constructores definidos de las clases. Se utilizan para acceder a los objetos. Estas variables se declaran de un tipo específico que no se puede cambiar.
- Objetos de la Clase, y varios tipos de variables de array están bajo tipo de datos de referencia.
- El valor predeterminado de cualquier variable de referencia es nulo.
- Una variable de referencia se puede utilizar para referirse a cualquier objeto del tipo declarado o cualquier tipo compatible.
- Ejemplo: `Carro unCarro = new Carro ("Ford");`

En Java, todas las variables deben ser declaradas antes de que puedan ser utilizados. La forma básica de una declaración de variable es la siguiente:

```
type identificador [ = valor][, identificador [= valor] ...] ;
```

- `type`, va a ser el tipo de dato que queremos declarar.
- `identificador`, es el nombre que le queremos dar a la variable.
- `valor`, es la cantidad o frase que le quieras asignar a la variable que estamos declarando.

Veamos unos ejemplos de cómo declarar variables en Java:

```
int a, b, c;    // declaramos tres variables de tipos int a, b, c.

int d = 3, e, f = 5; // declaramos dos variables de tipos int d y f, pero esta vez la instanciamos. A 'd' le
asignamos 3 y a 'f' le asignamos 5.

double pi = 3.14159; // declaramos un double denominado pi.

char x = 'x';    // la variable x posee un caracter 'x'.
```

Variables Locales

- Las variables locales se declaran en los métodos, constructores, o bloques.
- Se crean cuando se introduce el método, constructor o bloque y la variable serán destruidos una vez que sale el método, constructor o bloque.
- Son visibles sólo dentro del método declarado, constructor o bloque.
- Se implementan a nivel de pila interna.

Veamos un ejemplo:

```
public class Curso{

    public void kilometrajeCarro(){
        int kilometraje = 0;
        kilometraje = kilometraje + 7;
        System.out.println("El kilometraje del carro es: " + kilometraje);
    }

    public static void main(String args[]){
        Curso curso = new Curso();
        curso.kilometrajeCarro();
    }
}
```

```
}

```

- Aquí, el kilometraje es una variable local. Está definida dentro `kilometrajeCarro()` y su ámbito se limita sólo a este método.
- Si no inicializamos la variable kilometraje al ejecutar el código este nos daría un error.

Si ejecutamos ese código debería aparecernos lo siguiente:

```
El kilometraje del carro es: 7

```

Variables de instancia

- Las variables de instancia se declaran en una clase, pero fuera de un método, constructor o cualquier bloque.
- Se crean cuando se crea un objeto con el uso de la palabra `new` y se destruye cuando se destruye el objeto.
- Tienen valores que pueden ser referenciados por más de un método, constructor o bloque.
- Los modificadores de acceso se pueden dar para variables de instancia.
- Poseen valores por defecto. Para los números el valor por defecto es 0 , por Booleans es falso y por referencias de objeto es nulo. Los valores pueden ser asignados en la declaración o en el constructor.
- Se puede acceder directamente mediante una llamada al nombre de la variable dentro de la clase. Sin embargo dentro de los métodos estáticos debe ser llamado con el nombre completo. `Objeto.NombreVariable`.

```
import java.io.*;

public class Persona{
    // la variable de instancia nombre puede ser vista por todos los hijos de la clase
    public String nombre;

    // peso es una variable solo visible por la clase Persona
    private double peso;

    // La variable nombre es asignada en el constructor
    public Persona (String nombre){

```

```
this.nombre = nombre;
```

```
}
```

```
// Este metodo asigna un peso a la variable peso
```

```
public void setPeso(double peso){
```

```
    this.peso = peso;
```

```
}
```

```
// Este metodo imprime los datos de la persona
```

```
public void imprimirPersona(){
```

```
    System.out.println("Nombre : " + this.nombre );
```

```
    System.out.println("Peso : " + this.peso);
```

```
}
```

```
public static void main(String args[]){
```

```
    Persona alguien = new Persona("Carlos");
```

```
    alguien.setPeso(80);
```

```
    alguien.imprimirPersona();
```

```
}
```

```
}
```

Si ejecutamos ese codigo deberia aparecernos lo siguiente:

```
Nombre : Carlos
```

```
Peso :80.0
```

Variables de estáticas

- Las variables de clase se declaran con la palabra clave static en una clase, pero fuera de un método, constructor o un bloque.
- Sólo habría una copia de cada variable por clase, independientemente del número de objetos se crean de la misma.

- Las variables estáticas se usan muy poco aparte de ser declarado como constantes.
- Se almacenan en la memoria estática.
- Se crean cuando se inicia el programa y se destruyen cuando el programa se detiene.
- La visibilidad es similar a las variables de instancia. Sin embargo, las variables estáticas se declaran normalmente *public*, para que estén disponibles para los usuarios de la clase.
- Las variables estáticas se puede acceder llamando con el nombre de la clase. `NombreClase.NombreVariable`.

```
import java.io.*;

public class Empleado{

    // salario es una variable estatica privada de la clase empleado

    private static double salario;

    // DEPARTAMENTO es una constante

    public static final String DEPARTAMENTO = "Desarrollo";

    public static void main(String args[]){

        salario = 2000;

        System.out.println(DEPARTAMENTO + " posee un salario promedio de: " + salary);

    }

}
```

Si ejecutamos ese código debería aparecernos lo siguiente:

```
Desarrollo posee un salario promedio de: 2000
```

Java proporciona un conjunto de operadores para manipular variables. Hoy vamos a ver los operadores más importantes:

- Operadores aritméticos.
- Operadores relacionales.
- Operadores lógicos.
- Operadores de asignación.

Operadores aritméticos

Los operadores aritméticos se utilizan en expresiones matemáticas de la misma manera que se utilizan en el álgebra. En la siguiente tabla se muestran los operadores aritméticos:

Operador	Descripción
+	Adición - Suma los valores de los operadores
-	Resta - Resta el operando de la derecha del operador del lado izquierdo
*	Multiplicación - Multiplica los valores de ambos lados del operador
/	División - Divide el operador del lado izquierdo por el operando de la derecha
%	Módulo - Divide el operando de la izquierda por el operador del lado derecho y devuelve el resto
++	Incremento - Aumenta el valor del operando en 1
--	Decremento - Disminuye el valor del operando por 1

Veamos un ejemplo:

```
public class Curso {

    public static void main(String args[]) {

        int a = 10;
        int b = 20;
        int c = 25;
        int d = 25;

        System.out.println("a + b = " + (a + b) );
        System.out.println("a - b = " + (a - b) );
        System.out.println("a * b = " + (a * b) );
        System.out.println("b / a = " + (b / a) );
        System.out.println("b % a = " + (b % a) );
        System.out.println("c % a = " + (c % a) );
        System.out.println("a++ = " + (a++) );
        System.out.println("b-- = " + (b--) );
    }
}
```

```
// Mira la diferencia entre d++ and ++d
```

```
System.out.println("d++  = " + (d++) );
```

```
System.out.println("++d  = " + (++d) );
```

```
}
```

```
}
```

Se ejecutamos este código deberá producir:

```
a + b = 30
```

```
a - b = -10
```

```
a * b = 200
```

```
b / a = 2
```

```
b % a = 0
```

Operador	Descripción
==	Comprueba si los valores de dos operandos son iguales o no, si sí, entonces condición sea verdadera.
!=	Comprueba si los valores de dos operandos son iguales o no, si los valores no son iguales, entonces la condición se convierte en realidad.
>	Comprueba si el valor del operando de la izquierda es mayor que el valor del operando derecho, si sí, entonces condición sea verdadera.
<	Comprueba si el valor del operando de la izquierda es menor que el valor del operando derecho, si es así, entonces la condición sea verdadera.
>=	Comprueba si el valor del operando de la izquierda es mayor o igual que el valor del operando derecho, si sí, entonces condición sea verdadera.
<=	Comprueba si el valor del operando de la izquierda es menor o igual que el valor del operando derecho, si es así, entonces la condición sea verdadera.

```
c % a = 5
```

```
a++ = 10
```

```
b-- = 11
```

```
d++ = 25
```

```
++d = 27
```

Operadores relacionales

En Java existen las siguientes operadores relacionales:

Veamos un ejemplo:

```
public class Curso {  
  
    public static void main(String args[]) {  
  
        int a = 10;  
        int b = 20;  
  
        System.out.println("a == b = " + (a == b) );  
        System.out.println("a != b = " + (a != b) );  
        System.out.println("a > b = " + (a > b) );  
        System.out.println("a < b = " + (a < b) );  
        System.out.println("b >= a = " + (b >= a) );  
        System.out.println("b <= a = " + (b <= a) );  
    }  
}
```

Se ejecutamos este código deberá producir:

```
a == b = false
```

```
a != b = true
```

```
a > b = false
```

```
a < b = true
```

```
b >= a = true
```

```
b <= a = false
```

Operadores lógicos

En la siguiente tabla se muestran los operadores lógicos:

Operador	Descripción
&&	Llamado lógico AND. Si ambos operandos son distintos a cero, entonces la condición sea verdadera.
	Llamado operador lógico OR. Si alguno de los dos operandos son no cero, entonces la condición sea verdadera.
!	Llamado operador lógico NOT. Utilizado para invertir el estado lógico de su operando. Si una condición es verdadera, entonces el operador NOT será falso.

Veamos un ejemplo:

```
public class Curso {

    public static void main(String args[]) {

        boolean a = true;
        boolean b = false;

        System.out.println("a && b = " + (a&&b));

        System.out.println("a || b = " + (a||b));

        System.out.println("!(a && b) = " + !(a && b));

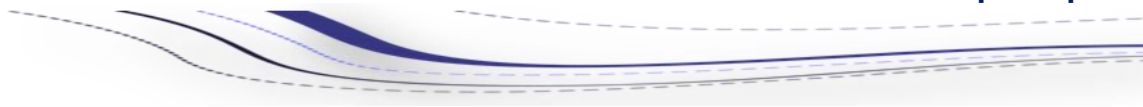
    }

}
```

Se ejecutamos este código deberá producir:

```
a && b = false
a || b = true
!(a && b) = true
```

Operadores de asignación



Operador	Descripción
=	Operador de asignación simple. Asigna valores de operados del lado derecho al operando del lado izquierdo.
+=	Añadir y operador de asignación, Añade operando derecho al operando izquierdo y asigna el resultado al operando de la izquierda.
-=	Restar y operador de asignación, se resta el operando derecho del operando de la izquierda y asigna el resultado al operando de la izquierda.
*=	Multiplicar y operador de asignación, se multiplica el operando derecho al operando de la izquierda y asigna el resultado a la izquierda del operando.
/=	Divide y operador de asignación, se divide el operando izquierdo con el operando derecho y asigna el resultado a la izquierda del operando.


En Java existen las siguientes operadores de asignación:


%=

Módulo y operado de asignación, se saca el módulo el operando izquierdo con el operando derecho y asignan el resultado al operando de la izquierda.

Veamos un ejemplo:

```
public class Curso {  
  
    public static void main(String args[]) {  
  
        int a = 10;  
        int b = 20;  
        int c = 0;  
  
        c = a + b;  
        System.out.println("c = a + b = " + c);  
  
        c += a;  
        System.out.println("c += a = " + c);  
  
        c -= a;  
        System.out.println("c -= a = " + c);  
  
        c *= a;  
        System.out.println("c *= a = " + c);  
  
        a = 10;  
        c = 15;  
        c /= a;  
        System.out.println("c /= a = " + c);  
    }  
}
```




```

a = 10;
c = 15;
c %= a;
System.out.println("c %= a = " + c);
}
}

```

Se ejecutamos este código deberá producir:

```

c = a + b = 30
c += a = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a = 5

```

Hoy vamos a ver que existen dos tipos de sentencias de decisiones en Java. Veamos cuales son:

- La sentencia **if**.
- La sentencia **switch**.

La sentencia if

Una sentencia if consiste en una expresión booleana seguida de una o más sentencias. Veamos la sintaxis:

```

if(expresion_Booleana)
{
    //Sentencias se ejecutan si la expresión booleana es verdadera
}

```

Si la expresión booleana es verdadera, entonces el bloque de código dentro del if se ejecuta. Si no se ejecutará el primer grupo de código después del final de la instrucción if.

Veamos un ejemplo:

```

public class Curso {

```

```
public static void main(String args[]){  
    int x = 15;  
  
    if( x < 30 ){  
        System.out.print("Esto es una sentencia if");  
    }  
}  
}
```

Si ejecutamos el código anterior obtendremos lo siguiente:

```
Esto es una sentencia if
```

La sentencia if..else

Una sentencia **if** puede ser seguido por una sentencia **else**, que se ejecuta cuando la expresión booleana es falsa.

Veamos la sintaxis:

```
if(expresion_Booleana)  
    // Se ejecuta cuando la expresión booleana es verdadera  
} else {  
    // Se ejecuta cuando la expresión booleana es falsa  
}
```

Veamos un ejemplo:

```
public class Curso {  
  
    public static void main(String args[]){  
        int x = 40;  
  
        if( x < 10 ){  
            System.out.print("Esto es una sentencia if");  
        }else{
```

```
System.out.print("Esto es una sentencia else");
```

```
}
```

```
}
```

```
}
```

Si ejecutamos el código anterior obtendremos lo siguiente:

```
Esto es una sentencia else
```

La sentencia if...else if...else

Una sentencia if puede ser seguida por un opcional else if ... else, que es muy útil para comprobar varias condiciones.

Veamos la sintaxis:

```
if(expresion_Booleana_1)
```

```
// Se ejecuta cuando la expresión booleana 1 es verdadera
```

```
} else if (expresion_Booleana_2) {
```

```
// Se ejecuta cuando la expresión booleana 2 es verdadera
```

```
} else if (expresion_Booleana_3) {
```

```
// Se ejecuta cuando la expresión booleana 3 es cierto
```

```
} else {
```

```
// Se ejecuta cuando ninguna condición anterior es verdadera.
```

```
}
```

Veamos un ejemplo:

```
public class Curso {
```

```
public static void main(String args[]){
```

```
int x = 45;
```

```
if( x == 15 ){
```

```
System.out.print("El valor de X es15");
```

```
}else if( x == 30 ){
```

```

System.out.print("El valor de X es 30");
}else if( x == 45 ){
    System.out.print("El valor de X es 45");
}else{
    System.out.print("X no cumple ninguna de las condiciones anteriores");
}
}
}
}

```

Si ejecutamos el código anterior obtendremos lo siguiente:

```
El valor de X es 45
```

La sentencia anidada if...else

También es posible usar una sentencia **if** o **if...else** dentro de otro **if** o **if..else**. Veamos la sintaxis:

```

if(expresion_Booleana_1)
    // Se ejecuta cuando la expresión booleana 1 es verdadera
    if (expresion_Booleana_2) {
        // Se ejecuta cuando la expresión booleana 2 es verdadera
    }
}

```

Veamos un ejemplo:

```

public class Curso {

    public static void main(String args[]){

        int x = 40;
        int y = 5;

        if( x == 40 ){
            if( y == 5 ){

```

```
System.out.print("X = 30 y Y = 5");
```

```
}
```

```
}
```

```
}
```

```
}
```

Si ejecutamos el código anterior obtendremos lo siguiente:

```
X = 40 y Y = 5
```

La sentencia switch

Una sentencia **switch** permite a una variable ser probada por una lista de condiciones. Cada condición se llama **case**. Veamos la sintaxis:

```
switch (expresion) {
```

```
    case valor1:
```

```
        // Declaraciones
```

```
        break; // opcional
```

```
    case valor2:
```

```
        // Declaraciones
```

```
        break; // opcional
```

```
    // Usted puede tener cualquier número de sentencias case.
```

```
    default: // Opcional
```

```
        // Declaraciones que cumplirá si la variable no entra en ningún caso.
```

```
}
```

Las siguientes reglas se aplican a una sentencia switch :

- La variable que se utiliza en una sentencia switch sólo puede ser un byte, short , int, o char.
- Puedes tener cualquier número de sentencias case dentro de un switch. Cada caso es seguido del valor a ser comparado.
- El valor de un caso debe ser el mismo tipo de datos que la variable en el switch.
- Cuando la variable del switch es igual a un caso, las instrucciones que siguen a ese caso se ejecutará hasta que se alcanza una sentencia break.
- Cuando se llega a una sentencia break, el caso termina, y el flujo de control pasa a la siguiente línea después de la sentencia switch.

- No todos los casos tiene que contener un break.
- Una sentencia switch puede tener un caso por defecto (opcional), que debe aparecer al final del switch. El caso por defecto se puede utilizar para realizar una tarea cuando ninguno de los casos es cierto.

Veamos un ejemplo:

```
public class Curso {  
  
    public static void main(String args[]){  
        char departamento = 'B';  
  
        switch(departamento)  
        {  
            case 'A' :  
                System.out.println("Desarrollo");  
                break;  
            case 'B' :  
                System.out.println("Recursos Humanos");  
                break;  
            case 'C' :  
                System.out.println("Finanzas");  
                break;  
            case 'D' :  
                System.out.println("Mercadeo");  
            default :  
                System.out.println("Departamento invalido");  
        }  
        System.out.println("Código para el departamento es " + departamento);  
    }  
}
```

```
}

```

Si ejecutamos el código anterior obtendremos lo siguiente:

```
Recursos Humanos

```

```
Código para el departamento es B

```

Hoy vamos a ver que existen cuatro tipos de bucles (Loop) en Java. Veamos cuales son:

- El bucle **while**.
- El bucle **do...while**.
- El bucle **for**.
- El bucle **for mejorado**.

El bucle while

Un bucle while es una estructura de control que le permite repetir una tarea un número determinado de veces. Veamos su sintaxis:

```
while(expresion_booleana)
{
    //Bloque de código
}

```

Cuando se ejecuta, si el resultado `expresion_booleana` es cierto, entonces se ejecutarán las acciones dentro del bucle. Esto continuará siempre y cuando el resultado de la expresión es verdadera. Cuando la expresión se prueba y el resultado es falso, el cuerpo del bucle se omitirá y la primera sentencia después del bucle while se ejecutará.

Veamos un ejemplo:

```
public class Curso {

    public static void main(String args[]) {

        int x = 20;

        while( x < 30 ) {

            System.out.print("valor de x : " + x );

            x++;

            System.out.print("\n");
        }
    }
}

```

```
}
}
}
```

Si ejecutamos el código anterior debemos esperar el siguiente resultado:

```
valor de x : 20
valor de x : 21
valor de x : 22
valor de x : 23
valor de x : 24
valor de x : 25
valor de x : 26
valor de x : 27
valor de x : 28
valor de x : 29
```

El bucle do...while

Un bucle do...while es similar a un bucle while, excepto que este está garantizando ejecutar al menos una vez el bloque de código. Veamos su sintaxis:

```
do
{
    //Bloque de código
} while(expresion_booleana)
```

Observe que la `expresion_booleana` aparece al final del bucle, por lo que las instrucciones del bucle ejecutar una vez antes de que el booleano es probado. Si la expresión booleana es verdadera, el flujo de control vuelve al **do**, y las instrucciones del bucle se vuelve a ejecutar. Este proceso se repite hasta que la expresión booleana es falsa.

Veamos un ejemplo:

```
public class Curso {
```



```
public static void main(String args[]){  
    int x = 20;  
  
    do{  
        System.out.print("valor de x : " + x );  
        x++;  
        System.out.print("\n");  
    }while( x < 30 );  
}  
}
```

Si ejecutamos el código anterior debemos esperar el siguiente resultado:

```
valor de x : 20  
valor de x : 21  
valor de x : 22  
valor de x : 23  
valor de x : 24  
valor de x : 25  
valor de x : 26  
valor de x : 27  
valor de x : 28  
valor de x : 29
```

El bucle for

Un bucle for es una estructura de control de repetición que permite escribir de manera eficiente un bucle que es necesario ejecutar un número determinado de veces. Un bucle **for** es útil cuando se sabe cuántas veces una tarea se va a repetir. Veamos su sintaxis:

```
for(inicializacion; expresion_booleana; actualizacion)  
{  
    //Bloque de código
```

```
}
```

Veamos un ejemplo:

```
public class Curso {  
  
    public static void main(String args[]) {  
  
        for(int x = 20; x < 30; x++) {  
            System.out.print("valor de x : " + x );  
            System.out.print("\n");  
        }  
    }  
}
```

Si ejecutamos el código anterior debemos esperar el siguiente resultado:

```
valor de x : 20  
valor de x : 21  
valor de x : 22  
valor de x : 23  
valor de x : 24  
valor de x : 25  
valor de x : 26  
valor de x : 27  
valor de x : 28  
valor de x : 29
```

La etapa de **inicialización** se ejecuta en primer lugar, y sólo una vez. Este paso le permite declarar e inicializar las variables de control del bucle.

A continuación, se evalúa la `expresion_booleana`. Si bien es cierto, se ejecuta el cuerpo del bucle. Si es falsa, el cuerpo del bucle no se ejecuta y el flujo de control salta a la siguiente instrucción más allá del bucle `for`.

Después de que el cuerpo del bucle se ejecuta para el flujo de control salta de nuevo a la instrucción de **actualización**. Esta declaración le permite actualizar las variables de control del bucle.

El bucle for mejorado

El bucle for mejorado se introdujo con la llegada de Java 5. Este se utiliza principalmente para el manejo de arrays. Veamos su sintaxis:

```
for(declaracion : expresion)
{
    //Bloque de código
}
```

- **Declaración:** La variable de bloque recién declarado, que es de un tipo compatible con los elementos del array que está accediendo. La variable estará disponible dentro del bloque para y su valor sería el mismo que el elemento dentro del array.
- **Expresión:** Esta se evalúa como el array que se tiene que recorrer. La expresión puede ser una variable de tipo array o una llamada al método que devuelve un array.

Veamos un ejemplo:

```
public class Curso {

    public static void main(String args[]){

        String [] empleados = {"Carlos", "Oscar", "Jony", "Alberto", "Ramses"};

        for(String nombre : empleados) {

            System.out.print(nombre);

            System.out.print(",");

        }

    }

}
```

Si ejecutamos el código anterior debemos esperar el siguiente resultado:

```
Carlos,Oscar,Jony,Alberto,Ramses,
```

Herencia

La herencia puede ser definida como el proceso en el que un objeto adquiere las propiedades de otro. Con su uso la información del objeto se hace manejable en un orden jerárquico.

Cuando hablamos de herencia, las palabras claves más utilizadas serían **extends** y **implements**. Estas palabras podrían determinar si un objeto es de un tipo o de otro. Mediante el uso de estas palabras claves, podemos hacer que un objeto adquiere las propiedades de otro objeto.

Relación ES-UN

ES-UN es una manera de decir: *Este objeto es un tipo de objeto*. Vamos a ver cómo la palabra clave **extends** se utiliza para conseguir la herencia.

```
public class Carro {  
    public String color;  
    public String motor;  
  
    public void setColor(String color) {  
        this.color = color;  
    }  
}  
  
public class Ford extends Carro {  
    private int codigoRadio;  
}  
  
public class Toyota extends Carro {  
  
    public void poseeECOManejo() {  
        return true;  
    }  
}
```

Ahora, con base en el ejemplo anterior, en términos orientados a objetos, lo siguiente es cierto:

- Carro es la superclase de la clase Ford.
- Carro es la superclase de la clase de Toyota.
- Ford y Toyota son subclases de la clase Carro.

Ahora bien, si tenemos en cuenta la relación es-un, podemos decir:

- Ford ES-UN Carro
- Toyota ES-UN Carro

Observemos que en la herencia, todas las subclases obtienen o heredan todos los métodos y propiedades del padre con excepción de las propiedades privadas de la superclase. En nuestro ejemplo sería lo siguiente: las subclases Ford y Toyota van a tener las propiedades `color` y `motor`, y a su vez heredan la función `setColor`.

La herencia no limita a las subclases a tener sus propias propiedades y métodos. En nuestro ejemplo podemos observar que la subclase **Ford** posee una nueva propiedad llamada `codigoRadio` y a su vez la subclase **Toyota** posee su propio método `poseeECOManejo`.

Veamos un ejemplo:

```
public static void main(String args[]){

    Carro c = new Carro();

    Ford f = new Ford();

    Toyota t = new Toyota();

    System.out.println(f instanceof Carro);

    System.out.println(t instanceof Carro);

}
```

Con `instanceof` podemos asegurar que una clase es de un tipo de clase en específico.

Observemos que **Ford** y **Toyota** es en realidad un Carro con `f instanceof Carro` y `t instanceof Carro` respectivamente.

Si ejecutamos el código anterior, obtenemos:

```
true
true
```

Puesto que tenemos una buena comprensión de la palabra clave **extends** echemos un vistazo a cómo se utiliza la palabra clave **implements** para obtener la relación es-un.

La palabra clave **implements** es utilizado por las clases por heredar de interfaces. Interfaces nunca pueden ser extendidas por las clases.

Veamos un ejemplo:

```
public interface encendidoMotor {  
  
    public void encender();  
    public void apagar();  
}  
  
public class MotorCuatroCilindros implements encendidoMotor{  
  
    @Override  
    public void encender() {  
        System.out.println("Encendido a 4 pulsos");  
    }  
}  
  
public class MotorOchoCilindros implements encendidoMotor{  
  
    @Override  
    public void encender() {  
        System.out.println("Encendido a 18 pulsos");  
    }  
}
```

Observemos que cuando implementamos la interfaz `encendidoMotor` en la clase `MotorCuatroCilindros` estamos creando un esquema de métodos los cuales van a hacer implementados de manera diferente al comportamiento que por ejemplo pudiesen tener en la clase `MotorOchoCilindros`.

Cabe destacar que la interfaces se utilizan para poseer el mismo esquema de métodos pero diferentes implementaciones de los mismos. Así podemos obtener diferentes comportamientos por clase por mas que posean las mismas guías de métodos.

Hoy vamos a ver todo lo relacionado a la creación y uso de hilos dentro del mundo de Java.

¿Qué es una aplicación multihilo?

Java es un lenguaje de programación multihilo. Un programa de multiproceso contiene dos o más partes que se pueden ejecutar al mismo tiempo y cada parte puede manejar diferentes tareas al mismo tiempo, haciendo un uso óptimo de los recursos disponibles, especialmente cuando el equipo tiene varias CPU.

Por definición la multitarea es cuando varios procesos comparten recursos comunes de procesamiento, tales como CPU. Multithreading extiende la idea de la multitarea en aplicaciones donde se puede subdividir operaciones específicas dentro de una sola aplicación en hilos individuales.

Cada uno de los hilos se pueden ejecutar en paralelo. El sistema operativo divide el tiempo de procesamiento, no sólo entre las diferentes aplicaciones, sino también entre cada hilo dentro de una aplicación.

¿Cómo creamos una aplicación multihilo en Java? - Modo 1

Si la clase está destinado a ser ejecutado como un hilo, entonces podemos lograr esto mediante la implementación de interface **Runnable**. Para crear un hilo debemos seguir los siguientes pasos:

Primer paso

Como primer paso, es necesario implementar el método `run()` proporcionado por la interface **Runnable**. Este método proporciona una puerta de entrada para el hilo y en el cual pondremos la lógica de negocio.

```
public void run( )
```

Segundo paso

El segundo paso tendremos que instanciar **Thread** usando el siguiente constructor:

```
Thread(Runnable threadObj, String threadName);
```

Observemos que `threadObj` es una instancia de la interface **Runnable** y `threadName` es el nombre que recibe el nuevo hilo.

Tercer paso

Una vez creado el objeto **Thread**, lo podemos iniciar llamando al método `start()`, que ejecuta una llamada al método `run()`. Veamos como:

```
void start( );
```

Veamos un ejemplo:

```
class Multihilo implements Runnable {
```

```
    private Thread hilo;
```

```
    private String nombreHilo;
```

```
    Multihilo(String nombre){
```

```
        nombreHilo = nombre;
```

```
System.out.println("Creando " + nombreHilo);
```

```
}
```

```
public void run() {
```

```
System.out.println("Ejecutando " + nombreHilo );
```

```
try {
```

```
for(int i = 4; i > 0; i--) {
```

```
System.out.println("Hilo: " + nombreHilo + ", " + i);
```

```
// vamos a dormir el hilo unos 50s
```

```
Thread.sleep(50);
```

```
}
```

```
} catch (InterruptedException e) {
```

```
System.out.println("Hilo " + nombreHilo + " interrumpido.");
```

```
}
```

```
System.out.println("Hilo " + nombreHilo + " termino.");
```

```
}
```

```
public void start () {
```

```
System.out.println("Iniciando " + nombreHilo );
```

```
if (hilo == null) {
```

```
hilo = new Thread (this, nombreHilo);
```

```
hilo.start ();
```

```
}
```

```
}
```

```
}
```



```
public class PruebaHilo {  
  
    public static void main(String args[]) {  
        Multihilo hilo1 = new Multihilo( "Hilo-1");  
        hilo1.start();  
        Multihilo hilo2 = new Multihilo( "Hilo-2");  
        hilo2.start();  
    }  
}
```

Observemos que en el ejemplo anterior implementamos la interfaz Runnable. Además, creamos dos hilos y los iniciamos.

Si ejecutamos el código anterior deberíamos obtener lo siguiente:

```
Creando Hilo-1  
Iniciando Hilo-1  
Creando Hilo-2  
Iniciando Hilo-2  
Ejecutando Hilo-1  
Thread: Thread-1, 4  
Ejecutando Hilo-2  
Hilo: Hilo-2, 4  
Hilo: Hilo-1, 3  
Hilo: Hilo-2, 3  
Hilo: Hilo-1, 2  
Hilo: Hilo-2, 2  
Hilo: Hilo-1, 1  
Hilo: Hilo-2, 1  
Hilo Hilo-1 termino.
```



```
Hilo Hilo-2 termino.
```

¿Cómo creamos una aplicación multihilo en Java? - Modo 2

La segunda forma de crear un hilo es crear una nueva clase que extiende la clase **Thread** utilizando los siguientes dos sencillos pasos. Este enfoque proporciona una mayor flexibilidad en el manejo de múltiples subprocesos creados usando los métodos disponibles en la clase **Thread**.

Primer paso

Tendremos que reemplazar el método `run()` que está disponible en la clase **Thread**. Este método proporciona una puerta de entrada para el hilo y en el cual pondremos la lógica de negocio.

```
public void run( )
```

Segundo paso

Una vez creado el objeto **Thread**, lo podemos iniciar llamando al método `start()`, el cual ejecuta una llamada al método `run()`.

```
void start( );
```

Veamos un ejemplo:

```
class Multihilo extends Thread {
    private Thread hilo;
    private String nombreHilo;

    Multihilo(String nombre){
        nombreHilo = nombre;
        System.out.println("Creando " + nombreHilo);
    }

    public void run() {
        System.out.println("Ejecutando " + nombreHilo );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Hilo: " + nombreHilo + ", " + i);
                // vamos a dormir el hilo unos 50s
                Thread.sleep(50);
            }
        }
    }
}
```

```

    }

    } catch (InterruptedException e) {

        System.out.println("Hilo " + nombreHilo + " interrumpido.");

    }

    System.out.println("Hilo " + nombreHilo + " termino.");

}

public void start () {

    System.out.println("Iniciando " + nombreHilo );

    if (hilo == null) {

        hilo = new Thread (this, nombreHilo);

        hilo.start ();

    }

}

}

}

public class PruebaHilo {

    public static void main(String args[]) {

        Multihilo hilo1 = new Multihilo( "Hilo-1");

        hilo1.start();

        Multihilo hilo2 = new Multihilo( "Hilo-2");

        hilo2.start();

    }

}

```

Observemos que en el ejemplo anterior extendemos de la clase Thread. Además, creamos dos hilos y los iniciamos.

Si ejecutamos el código anterior deberíamos obtener lo siguiente:

Creando Hilo-1

```

Iniciando Hilo-1
Creando Hilo-2
Iniciando Hilo-2
Ejecutando Hilo-1
Thread: Thread-1, 4
Ejecutando Hilo-2
Hilo: Hilo-2, 4
Hilo: Hilo-1, 3
Hilo: Hilo-2, 3
Hilo: Hilo-1, 2
Hilo: Hilo-2, 2
Hilo: Hilo-1, 1
Hilo: Hilo-2, 1
Hilo Hilo-1 termino.
Hilo Hilo-2 termino.

```

Polimorfismo en Java



Al escuchar la palabra **polimorfismo** lo primero que se nos viene a la mente es "Muchas Formas" y bueno, si, básicamente esta es la idea general, pero y que mas gira en torno a esto???.....

en esta entrada veremos algunos ejemplos y puntos a tener en cuenta sobre el **polimorfismo** en la programación orientada a objetos y para eso aplicaremos otros conceptos trabajados con anterioridad..... y más adelante desarrollaremos un **ejemplo** en Java donde pondremos a prueba lo aprendido en esta entrada.....

Algunas Ideas.

Como se mencionó en la entrada sobre Conceptos Básicos, podemos definirlo como la capacidad que

tienen los objetos de comportarse de múltiples formas, programando de manera general en vez de hacerlo de forma específica...

Alguna vez consultando, definían este concepto como la forma en la que podemos tratar una subClase como si fuera una Clase del tipo de su superClase, usando solo los métodos o atributos disponibles para la Clase declarada..... es decir, si tenemos una clase "**X**" podemos decir que "**X**" es de tipo "**Y**" esto se cumple solo si existe una relación de herencia entre ellas, ya sea si "**X**" hereda de "**Y**"(extends) o si "**X**" implementa a "**Y**" (implements)

Retomando (de manera reducida) el ejemplo de la entrada sobre conceptos básicos en el punto sobre **polimorfismo**, se explica mejor lo anterior usando "**X**" como cualquier clase **Cuadrado**, **Triangulo** o **Circulo** y "**Y**" como la clase **FiguraGeometrica**, por lo tanto decimos que por ejemplo la clase **Triangulo** es de tipo **FiguraGeometrica**, en Java esto lo representamos así:

```

1    class FiguraGeometrica{
2
3    }
4
5    class Triangulo extends FiguraGeometrica {
6
7    }
8
9    public class Principal{
10
11    public void metodo(){
12    /**Puedo crear objetos polimorficos*/
13    /**Objeto Triangulo de tipo FiguraGeometrica*/
14    FiguraGeometrica triangulo=new Triangulo();
15    }
16    }
```

Vemos que **FiguraGeometrica** es la **superClase** y **Triangulo** es la clase hija o subClase, y por medio del **polimorfismo** podemos crear una instancia de **Triangulo** de tipo **FiguraGeometrica**...

Algunas Consideraciones.

(Estas consideraciones fueron tomadas de la revista digital JavaWord pues me parecen muy claras a la hora de trabajar con polimorfismo.)

Como en todo, tenemos unas reglas que se deben cumplir y tener en cuenta cuando vamos a trabajar con objetos polimórficos, estas son :

- Una variable de referencia puede ser de un solo tipo, y una vez que fue declarada, ese tipo jamás puede modificarse..... por ejemplo, si declaramos **triangulo** de tipo **FiguraGeometrica**, no podemos decir más adelante que el mismo objeto **triangulo** es de tipo **FiguraPuntiaguda**...
- Una referencia es una variable, de manera que puede ser reasignada a otros objetos (a menos que se declare como final). por ejemplo podemos hacer lo siguiente:

```

1  FiguraGeometrica miFiguraGeometrica = new FiguraGeometrica();
2  Cuadrado miCuadro=new Cuadrado();
3
4  /**Puedo crear objetos polimórficos, asignando su referencia*/
5  miFiguraGeometrica=miCuadro;

```

- Una variable de referencia puede tomar como referencia cualquier objeto del mismo tipo, o un subtipo del mismo..... va muy ligada a la anterior, pero nos dice que si por ejemplo tenemos la clase **cuadroPequeño** que es subClase de **cuadro**, al ser "**nieta**" de **FiguraGeometrica**, si se crean instancias de ella, se pueden reasignar a instancias de **FiguraGeometrica**...
- Un tipo de variable de referencia determina los métodos que pueden ser invocados sobre el objeto que la variable referencia.... **Triangulo** al ser de tipo **FiguraGeometrica** puede acceder no solo a los métodos de la clase **Triangulo** sino también a los de **FiguraGeometrica** (Claro, pues tiene que existir una relación de Herencia entre ellos)
- Una variable de referencia puede tener como tipo el de una interface. Si es declarada de esta manera, podrá contener cualquier clase que implemente la interfaz..... las interfaces también permiten el uso de herencia, por lo tanto podemos crear objetos usando para esto una interfaz, por ejemplo si **FiguraGeometrica** fuera una interface, lo anterior sería igual..... (esto también aplica para las clases Abstractas).

Como mencionamos en entradas pasadas, las clases abstractas y las interfaces (al ser clases completamente abstractas) no pueden ser instanciadas, o no directamente, la única forma de hacerlo es mediante la herencia, ya que así extendemos sus funcionalidades creando objetos referenciando clases pero siendo del tipo de su superClase (abstracta o interface)

INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO

17/10/2014

por [pepe abel](#)

[Comentarios 0](#)

Hace algunos días indicamos como **configurar un entorno de desarrollo con Android Studio** (AS) *Beta*. Ahora, en víspera de la liberación de la versión Lollipop (Android 5.0), toca el turno de mostrar cómo crear una sencilla aplicación utilizando este prometedor entorno, indicando además que este tema constara en 4 post relacionados. De modo que partimos justo después del proceso de instalación, ejecutando AS. De manera predeterminada, el ejecutable se localiza dentro de la carpeta bin de la ruta de instalación (**C:\Program Files (x86)\Android\android-studio\bin**, por ejemplo), con el nombre **studio64.exe** o **studio.exe**, según la arquitectura de nuestro equipo. El proceso es el que a continuación detallamos:



Ventana inicial Android Studio

Asistente para la creación de una aplicación

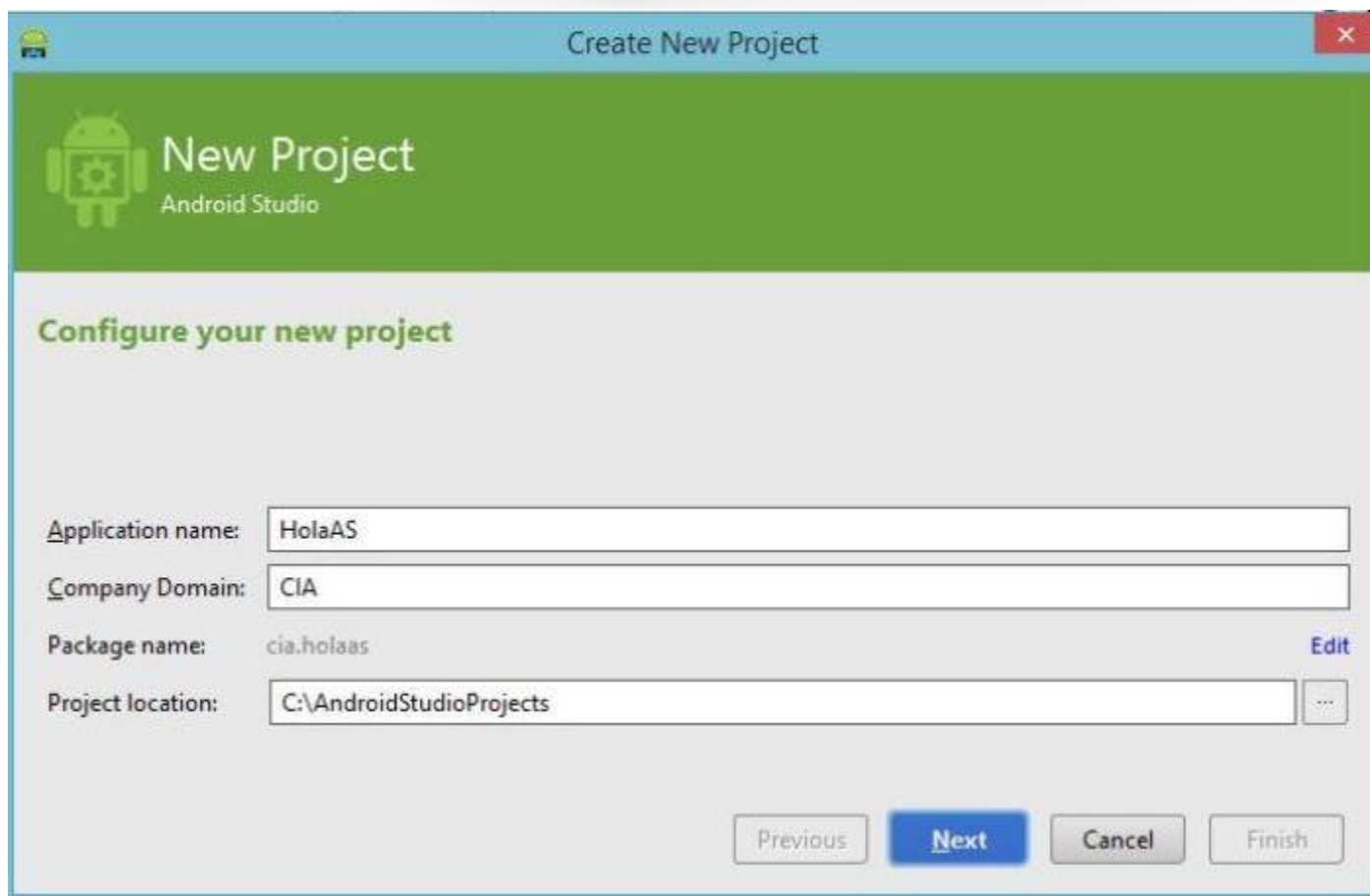
Al iniciar AS (o bien desde el menú **File->New Project...**), nos aparece una venta con las opciones para crear importar o abrir un proyecto.

Seleccionamos la opción **New Project...**, iniciando un asistente que pide los siguientes datos:

- Nombre de la aplicación: **HolaAS**
- Dominio de la compañía: **CIA**
- Localización del proyecto: **C:\AndroidStudioProjects**

Es notoria una sección donde nos muestra el nombre del paquete (*cia.holaas*), que a grandes rasgos define la organización de nuestras clases dentro del proyecto, la cual si se desea, puede modificarse presionando el botón *edit*.

Presionar el botón *Next* para dirigirnos a la siguiente ventana del asistente.



Android Studio

New Project

Configure your new project

Application name: HolaAS

Company Domain: CIA

Package name: cia.holaas [Edit](#)

Project location: C:\AndroidStudioProjects

Previous Next Cancel Finish

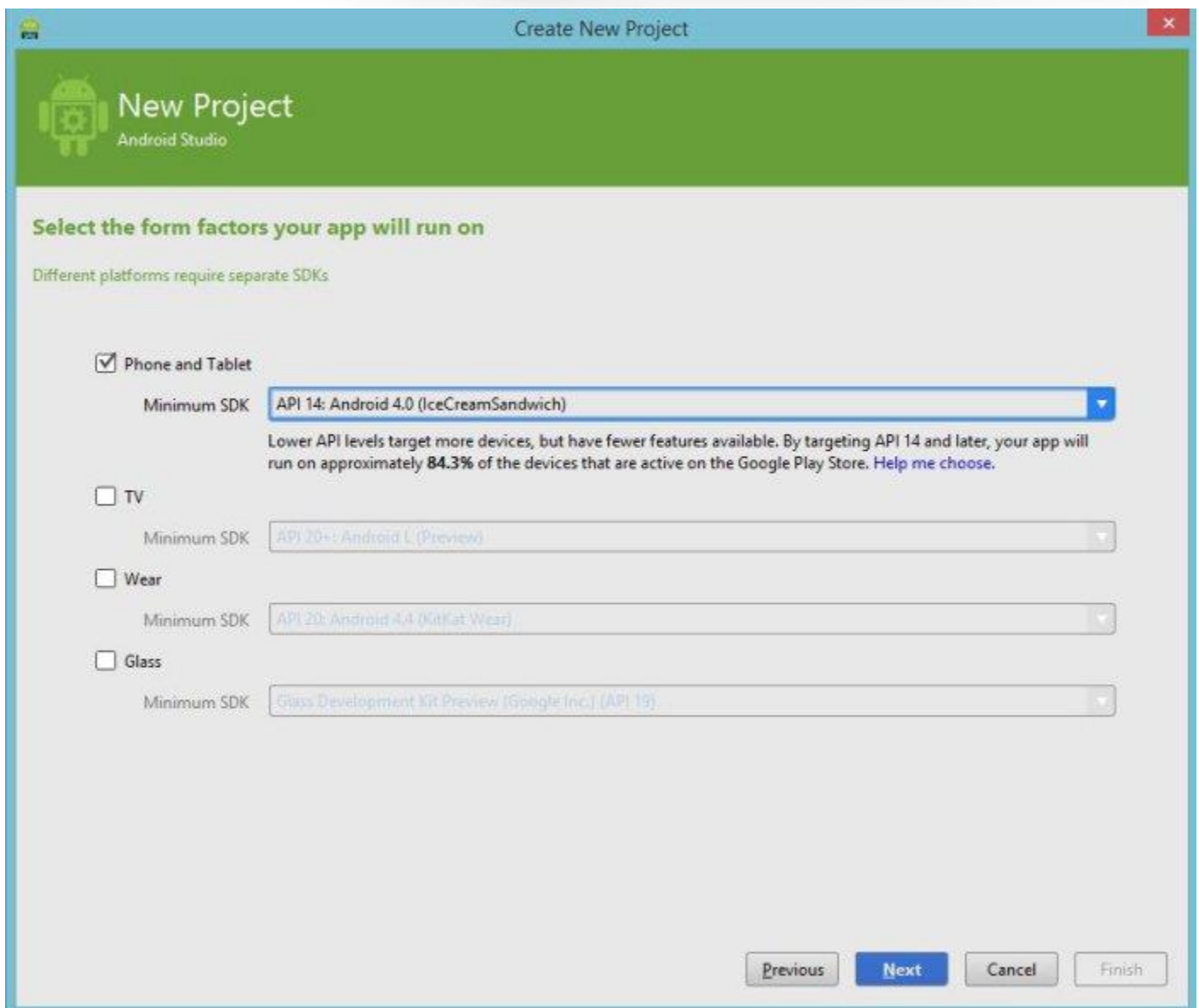
Ventana inicial del asistente

En esta sección del asistente, se solicita el tipo de dispositivos que soportara la aplicación: **teléfonos y tablets, televisores, wereables y/o google glass**. Para cada una de ellas es necesario contar con las **APIS** correspondientes. Para nuestro ejercicio, elegiremos solo la primera opción con el **API 14**. La elección del SDK mínimo define para que versiones de Android correrá nuestra aplicación. En nuestro caso particular, funcionara a partir de la versión 4.0 en adelante, lo cual garantiza su compatibilidad en más del 80% de los dispositivos existentes.

La elección del *SDK* mínimo tiene algunos detalles:

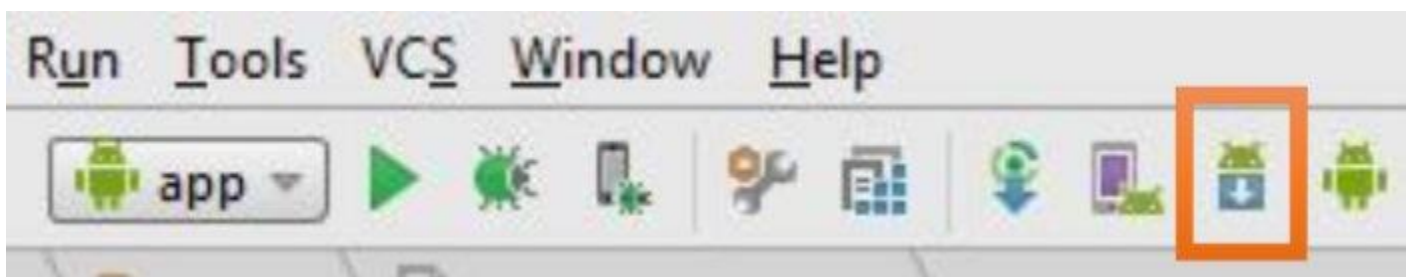
- Entre más bajo sea el *SDK*, garantizaremos la compatibilidad para el mayor número de dispositivos, pero muchas de las características más nuevas para el desarrollo no estarán disponibles
- Entre más alto sea el *SDK*, la compatibilidad será menor con respecto al número de dispositivos de mercado, pero tendremos disponible prácticamente todas las instrucciones y nuevas características de desarrollo

Presionar el botón Next para dirigirnos a la siguiente ventana del asistente



Elección de los tipos de dispositivos y versión del SDK

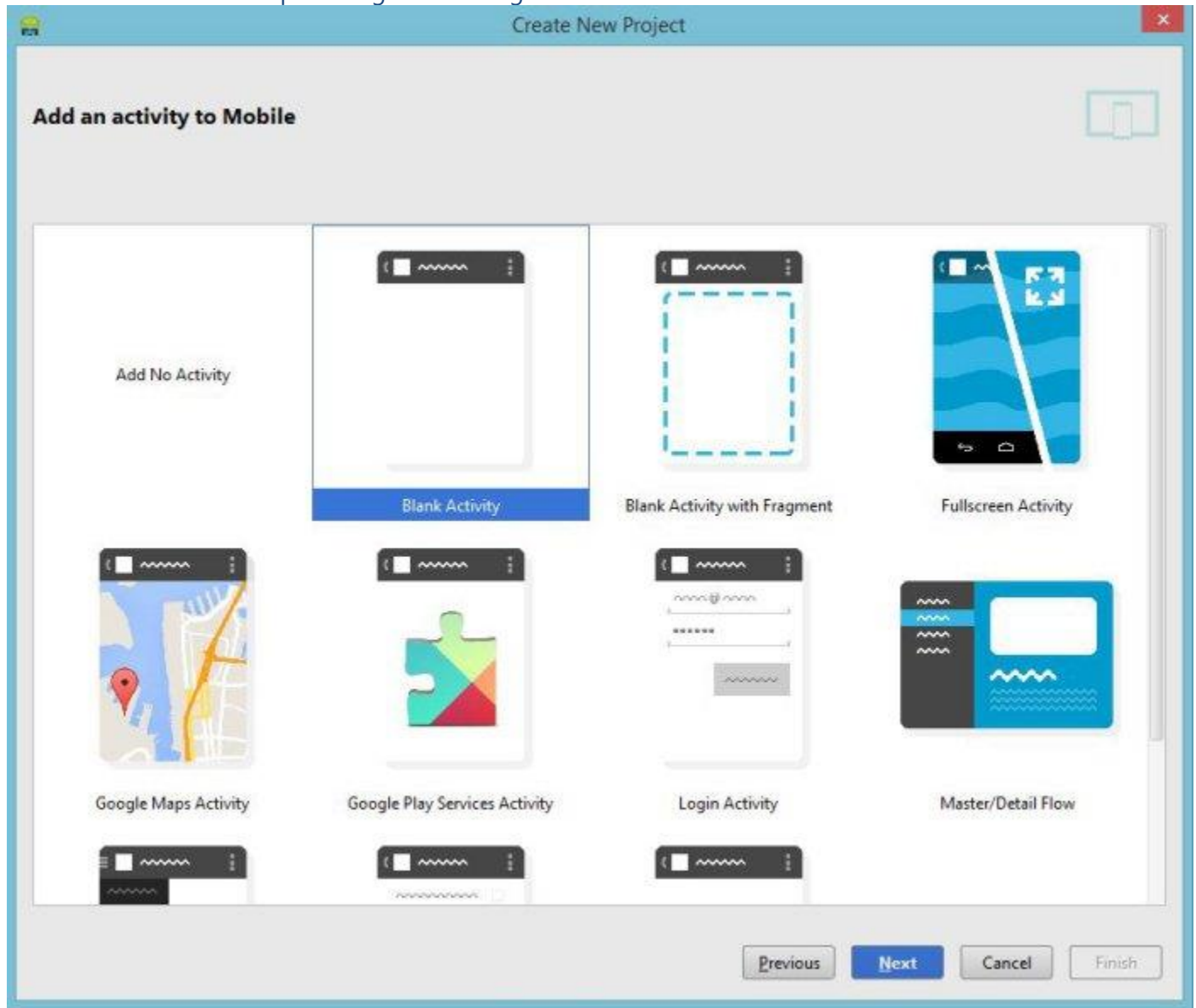
Nota: La configuración inicial de AS solo contiene el último SDK Android (para nuestro caso la versión 19), por lo que si se requiere el uso de un API distinta, es necesario la descarga de archivos adicionales mediante el SDK Manager, acción **perfectamente explicada** en un post anterior.



Ejecución del SDK Manager

En la siguiente sección del asistente, definiremos la venta inicial de la aplicación (**Activity**). Si bien existen varias plantillas a elegir como una de **Google Maps** o de **Inicio de Sesión**, para efectos sencillos elegiremos la plantilla en blanco (**Blank Activity**).

Presionar el botón *Next* para dirigirnos a la siguiente ventana del asistente



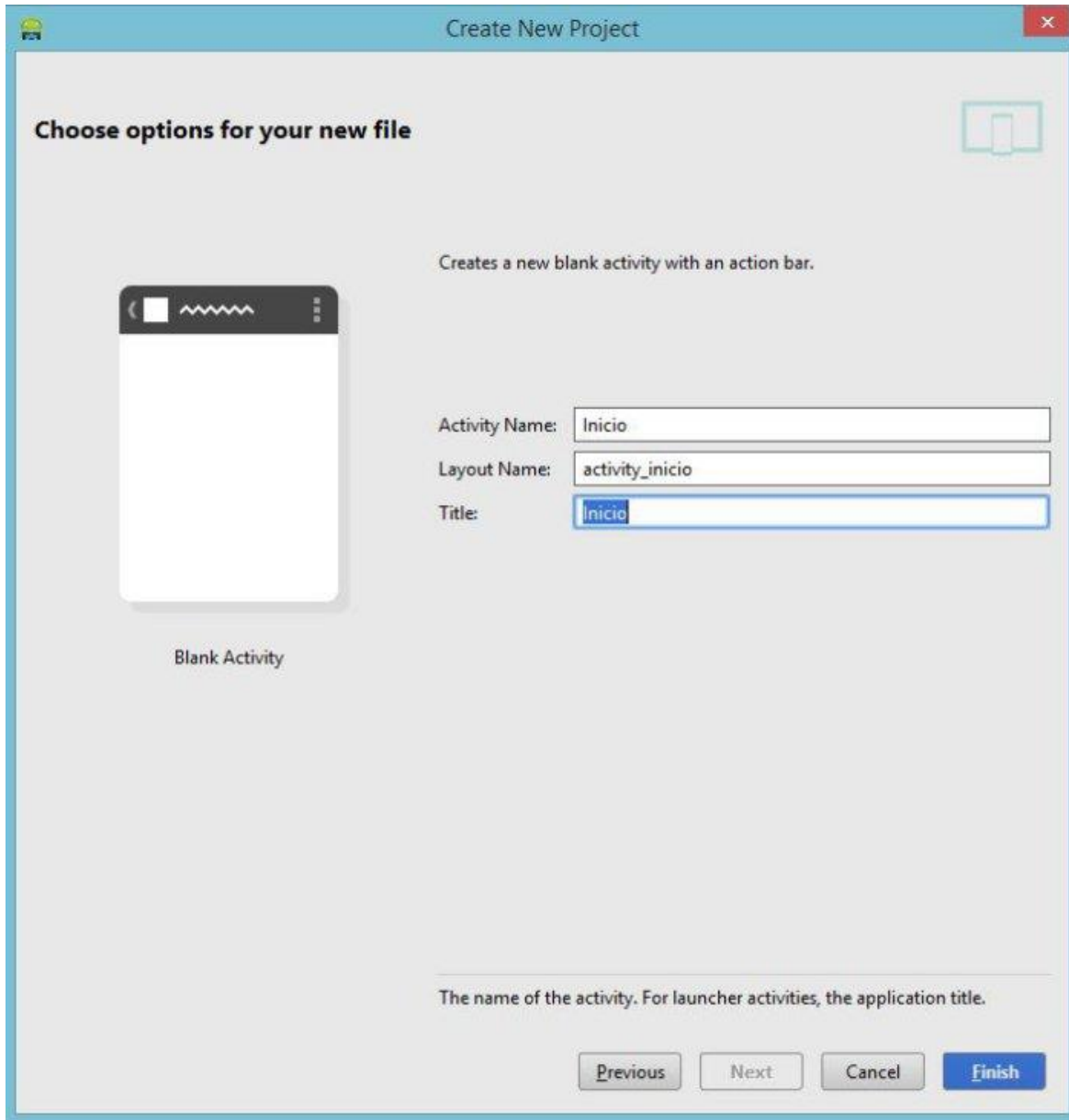
Selección de plantilla

Finalmente es necesario indicar tres datos:

- **Nombre del Activity.** Que finalmente se convertirá en una clase java: **Inicio**
- **Nombre del Layout.** Que contendrá la parte del diseño de la interfaz de usuario en forma de archivo xml: **activity_inicio**

- **Título.** El título que mostrara la ventana inicial: **Inicio**

Presionar el botón Finish, para dar por terminado el asistente. Con ello se realizan las configuraciones necesarias para crear nuestro proyecto.

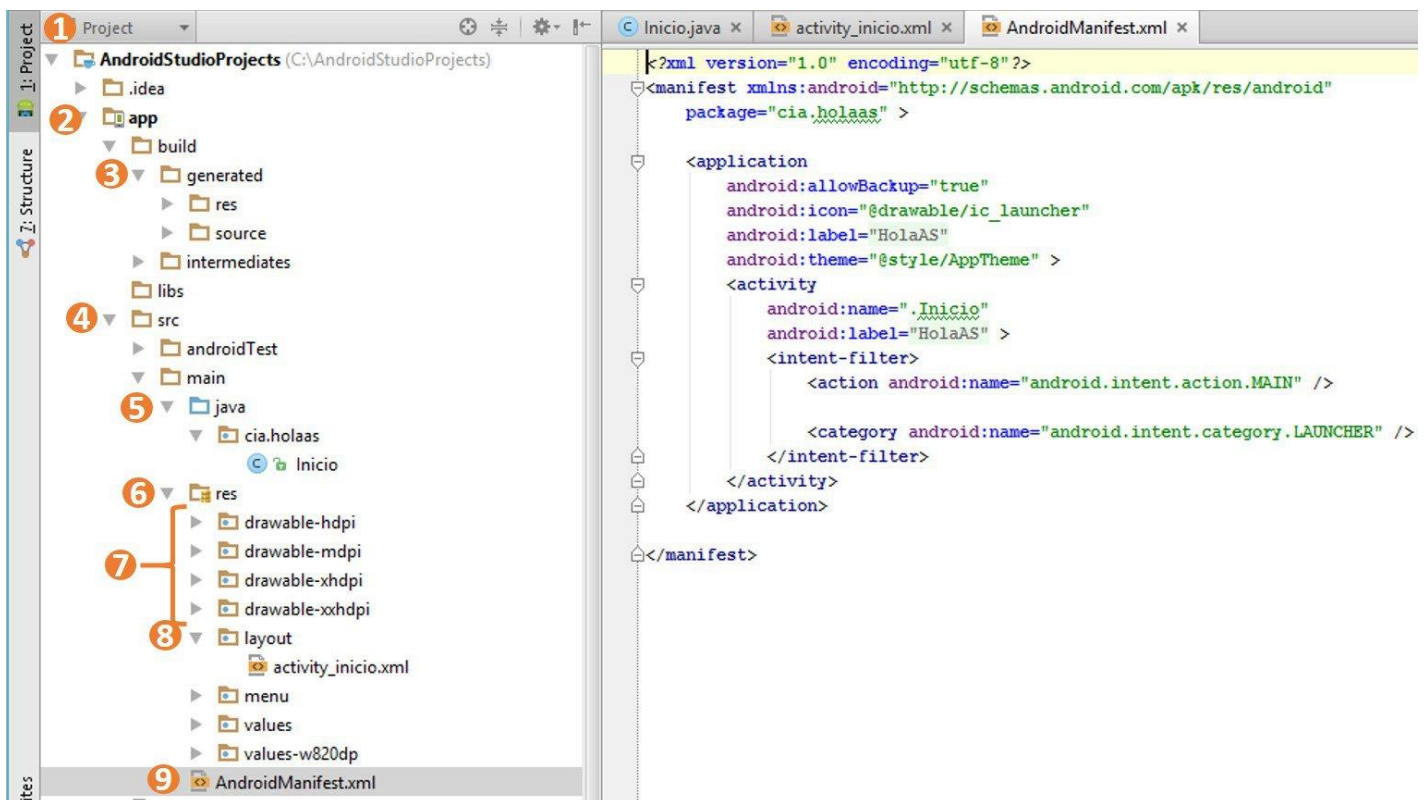


Nombre de los archivos principales

Panorama general del proyecto

1. **Project.** Aquí se visualizan todas las carpetas y archivos involucrados en el proyecto. Entre los mas destacados:
2. **Carpeta app.** Contiene todos los archivos necesarios para el proyecto

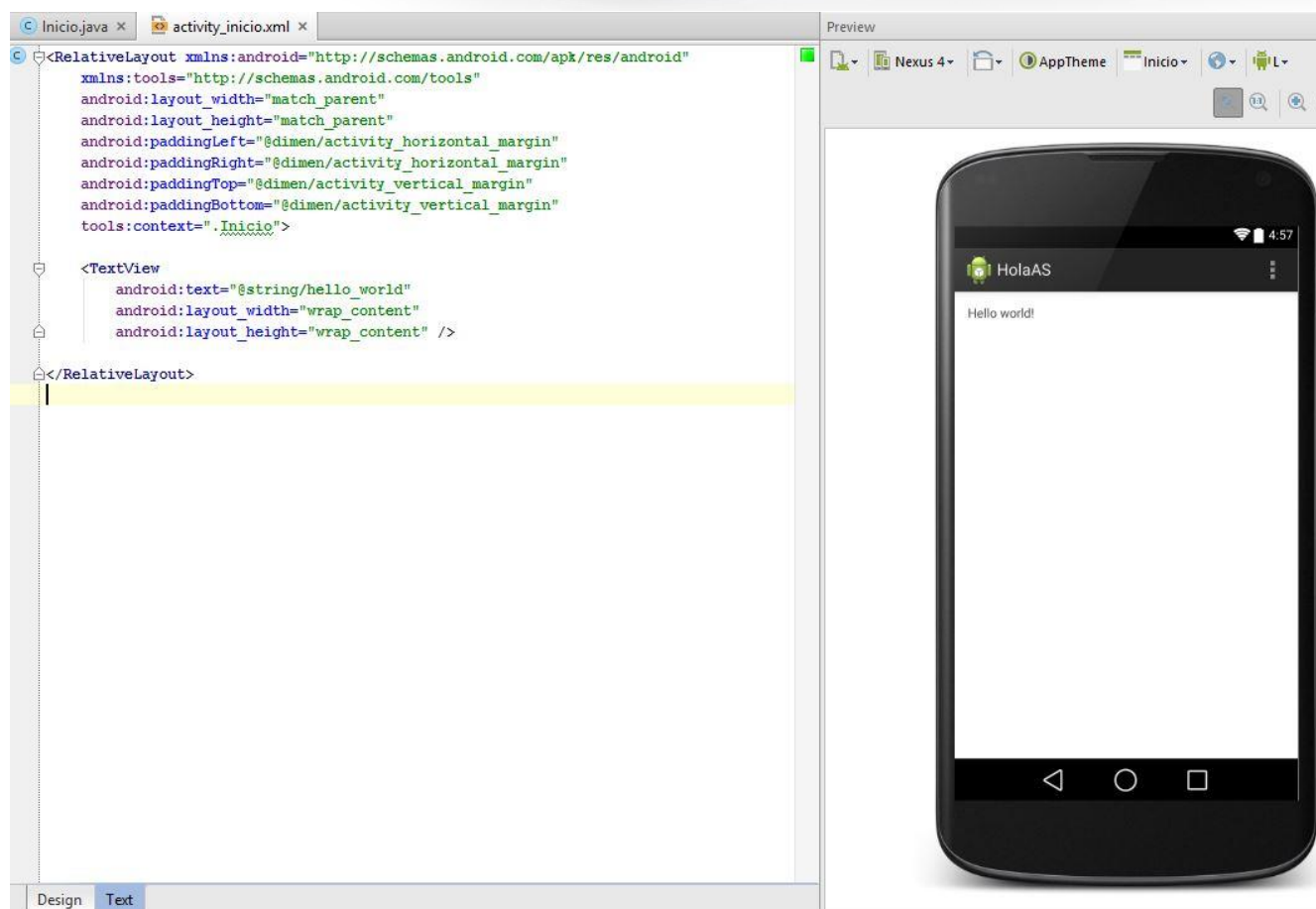
3. **Carpeta *generated*.** Todos los archivos que se generan de manera automática, por lo cual se recomienda mantener estos archivos como de solo lectura
4. **Carpeta *src*.** Todos los archivos fuente, es decir archivos que podemos modificar, tales como .java o xml
5. **Carpeta *java*.** Contiene los archivos .java organizados por paquetes (Como por ejemplo nuestro archivo inicio.java creado con el asistente)
6. **Carpeta *res*.** Contiene los archivos de recursos: iconos, imágenes, archivos de diseño, etc.
7. **Carpetas *drawable*.** Contiene los archivos de recursos para diversas resoluciones de pantalla
8. **Carpeta *layout*.** Contiene los archivos xml que definen la interfaz de usuario (Como por ejemplo nuestro archivo activity_inicio.xml creado con el asistente)
9. ***AndroidManifest.xml*.** Archivo que contiene las configuraciones generales del proyecto, como por ejemplo, el icono de la aplicación o los permisos otorgados a la aplicación.



Archivos y carpetas en un proyecto Android Studio

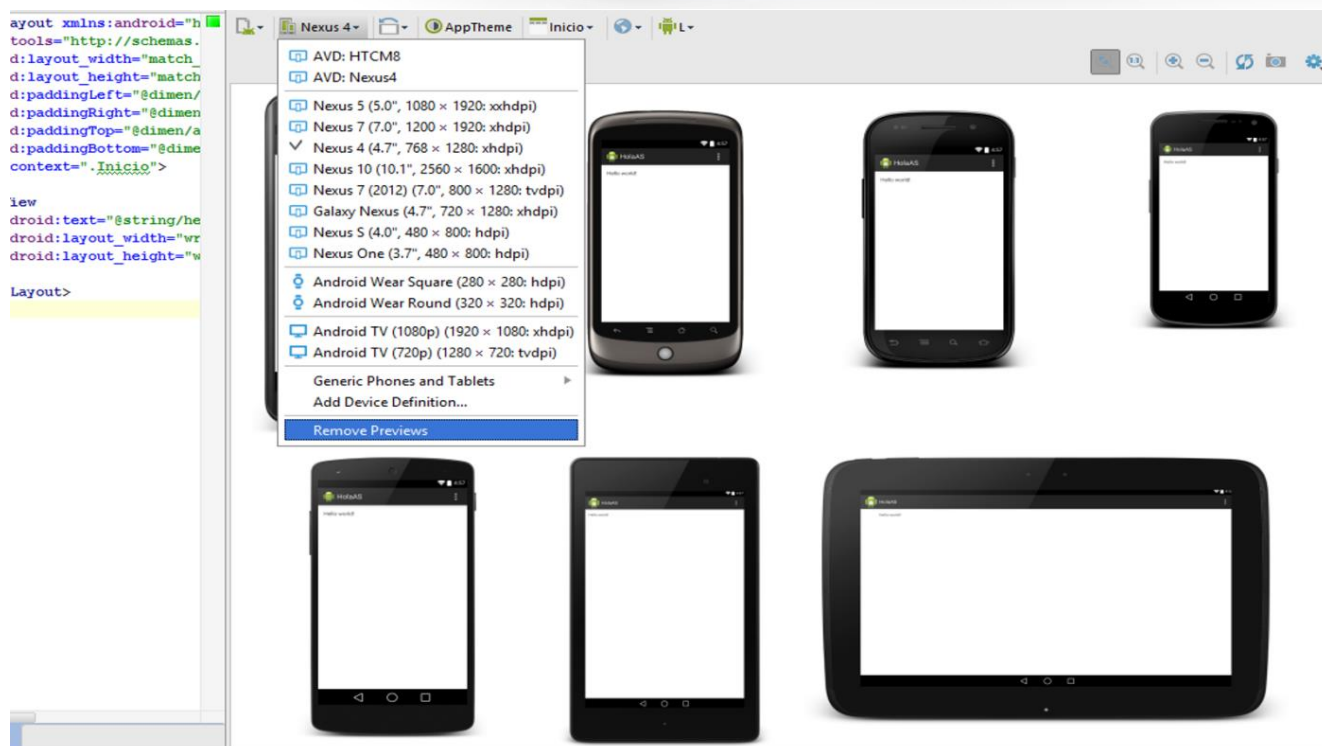
Previsualización del aspecto grafico

Algo que se agradece es la visualización previa de la interfaz gráfica sin necesidad de correr la aplicación. Para ello solo es necesario tener abierto y activo el archivo *xml* que contiene la interfaz de usuario (***activity_inicio.xml***, para nuestro ejercicio), para poder visualizar su aspecto a la derecha, de forma predeterminada con un dispositivo **Nexus 4**.



Previsualización en un Nexus 4

Si se desea, puede visualizarse en todo tipo de pantallas, seleccionando dentro de la lista de dispositivos disponibles la opción **Preview All Screen Sizes**, con la opción de hacer más grande la vista que muestra las pantallas, a fin de contar con una mejor apreciación.



Vista en multiples pantallas

INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO (II)

21/10/2014

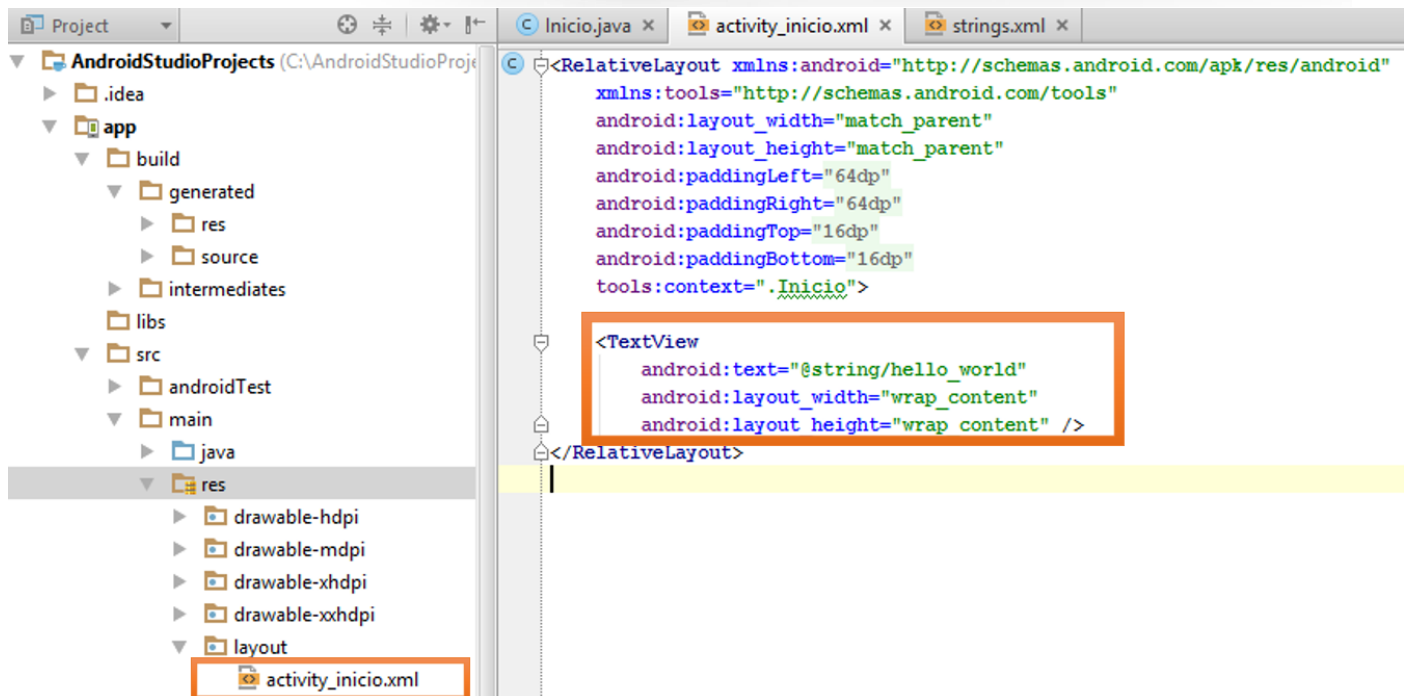
por [pepe_abel](#)

[Comentarios 0](#)

Este post es una continuación de reciente artículo **Introducción al desarrollo de aplicaciones con Android Studio (Parte I)**, en la primer entrega, mostramos paso a paso como crear una aplicación a través de un asistente y analizamos la estructura general de un proyecto; ahora, en la segunda parte realizaremos modificaciones al proyecto recién creado, que si bien son pequeñas, también son significativas, ya que representan un avance para aquellos que recién se inician en este tema, realizando las actividades con poca dificultad y apreciando la ejecución de estos cambios de manera inmediata. Abrimos entonces, el proyecto original para iniciar el presente tutorial.

La interfaz grafica

Los archivos para la interfaz gráfica se localizan dentro de la ruta **src/main/res/layout** y son archivos con extensión **.xml**. Nuestro ejemplo contiene el archivo **activity_inicio.xml**:



Archivo XML con la definición de la Interfaz Gráfica

Se aprecia que el contenido de este archivo define el aspecto visual. Nos detenemos justo en el recuadro naranja donde resaltamos el uso de un **Widget** (control) de tipo **TextView**, el cual permite mostrar etiquetas de texto y define los valores para tres atributos:

- **text:** Texto de la etiqueta
- **layout_width:** El ancho de la etiqueta, cuyo valor *wrap_content* indica que el ancho del control se ajustará al contenido de la etiqueta
- **layout_height:** El largo de la etiqueta, cuyo valor *wrap_content* indica que el largo del control se ajustará al contenido de la etiqueta

Vamos a modificar el texto, el cual en realidad, no es simple texto sino que define el uso de una variable (**recurso string**) llamada *hello_world*, por lo tanto es necesario modificar el texto de dicha variable, la cual se encuentra en el archivo **strings.xml** localizado en la ruta **src/main/res/values**. Tecleamos el nuevo texto **"Hola PoderPDA.com"**



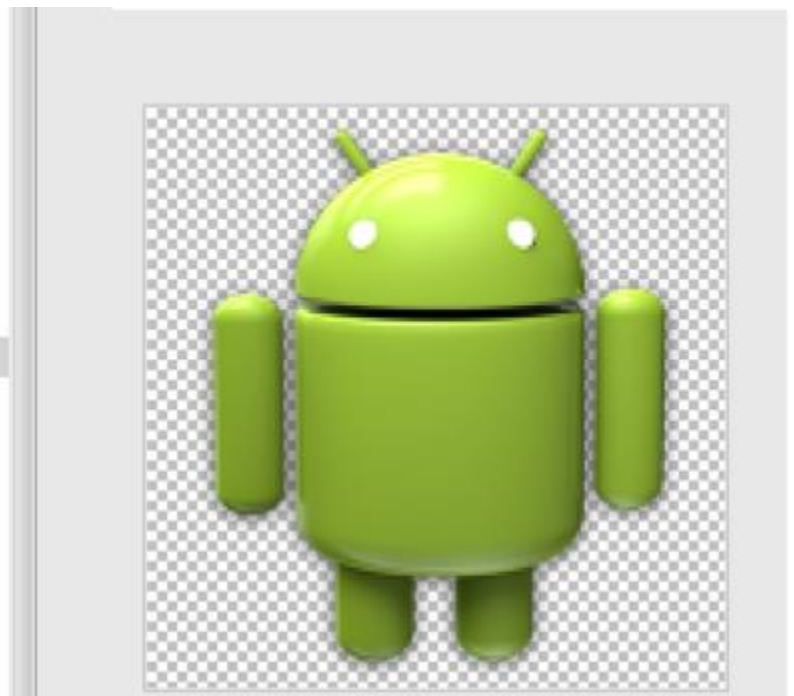
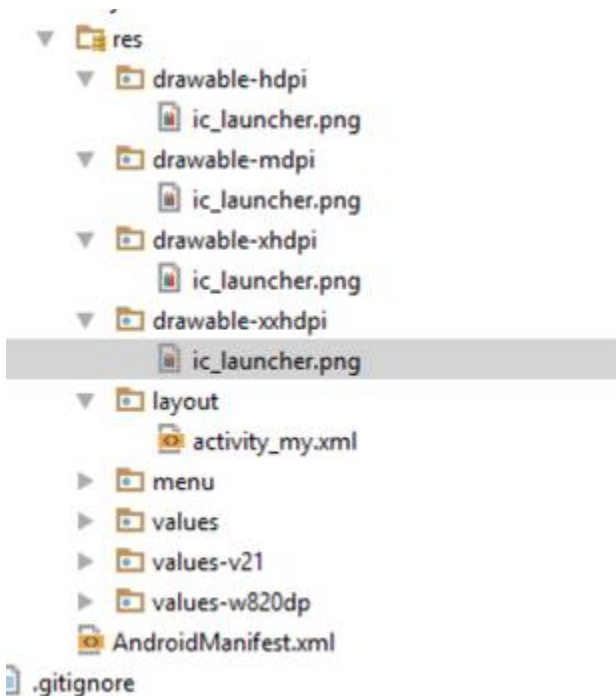
Recursos string

El uso de los recursos **string** no es necesario, pero si recomendable, exigiendo el uso de variables para los valores de texto en lugar de escribir de manera directa el valor de la cadena.

El icono de la aplicación

El **icono** se encuentra definido en cada una de las carpetas **drawable** para cada una de las resoluciones con el nombre de **ic_launcher.png**:

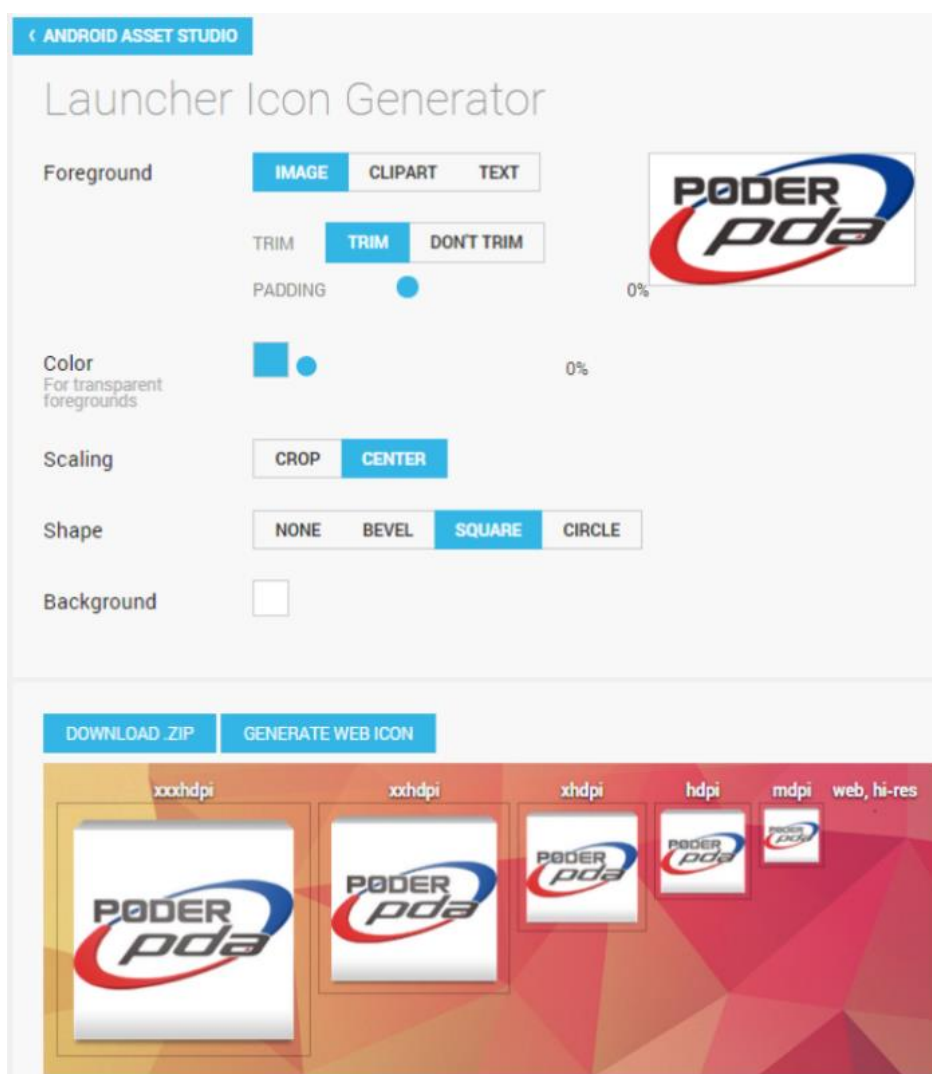
- mdpi (48×48)
- hdpi (72×72)
- xhdpi (96×96)
- xxhdpi (144×144)



Icono original

La siguiente acción consistirá en sustituir este icono, seleccionando nuestra propia imagen y a pesar de que la edición puede realizarse con cualquier editor de imágenes, utilizando una herramienta disponible en la web llamada **Android Asset Studio** para generar los archivos en diferentes resoluciones de una manera sencilla:

- Accedemos a la página [Android Asset Studio](#)
- Seleccionamos la opción **Launcher icons**
- Seleccionamos la opción **Foreground e Image**. Con ello, nos solicitará una imagen, la cual fungirá como icono.
- Después de elegir el archivo, la aplicación generará los iconos necesarios y solo basta descargar el archivo comprimido (zip) con todas las resoluciones y copiar los archivos en sus correspondientes carpetas (carpetas drawable)



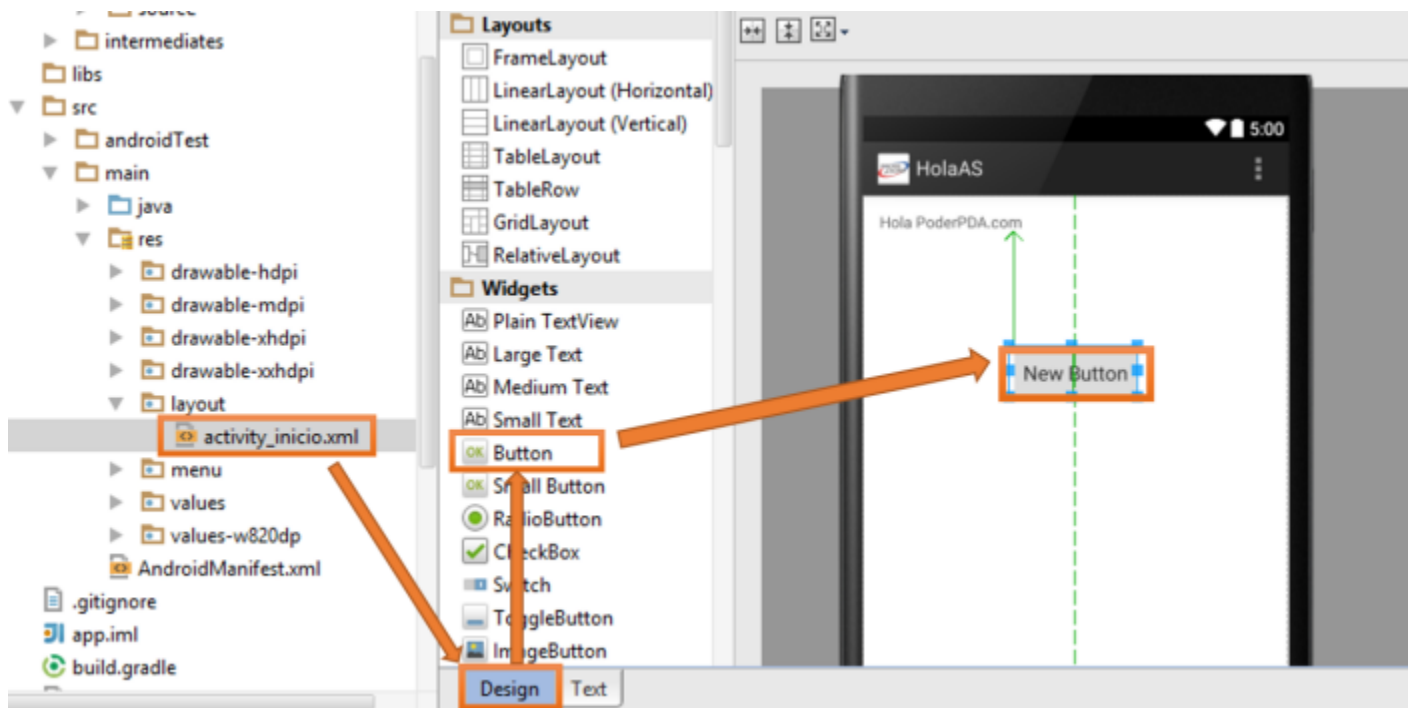
Usando Android Asset Studio

Ahora notamos los pequeños cambios que hemos realizado tanto en la parte del texto de la etiqueta como en el icono de la aplicación. Importante notar que por ahora dejaremos el icono con el mismo nombre, aunque es posible modificar esta propiedad.

Código Java

Es momento de agregarle funcionalidad, insertando un botón y codificando el evento click:

- Abrir el archivo Inicio.java que se encuentra dentro de la carpeta java y el paquete cia.holaas
- Seleccionar la pestaña Design a fin de mostrar el archivo en modo diseño
- Al seleccionar el modo diseño, aparecen una serie de controles para ser insertados en la pantalla de la interfaz
- Seleccionar un botón y arrastrarlo al centro de la pantalla, tal y como se muestra en la figura

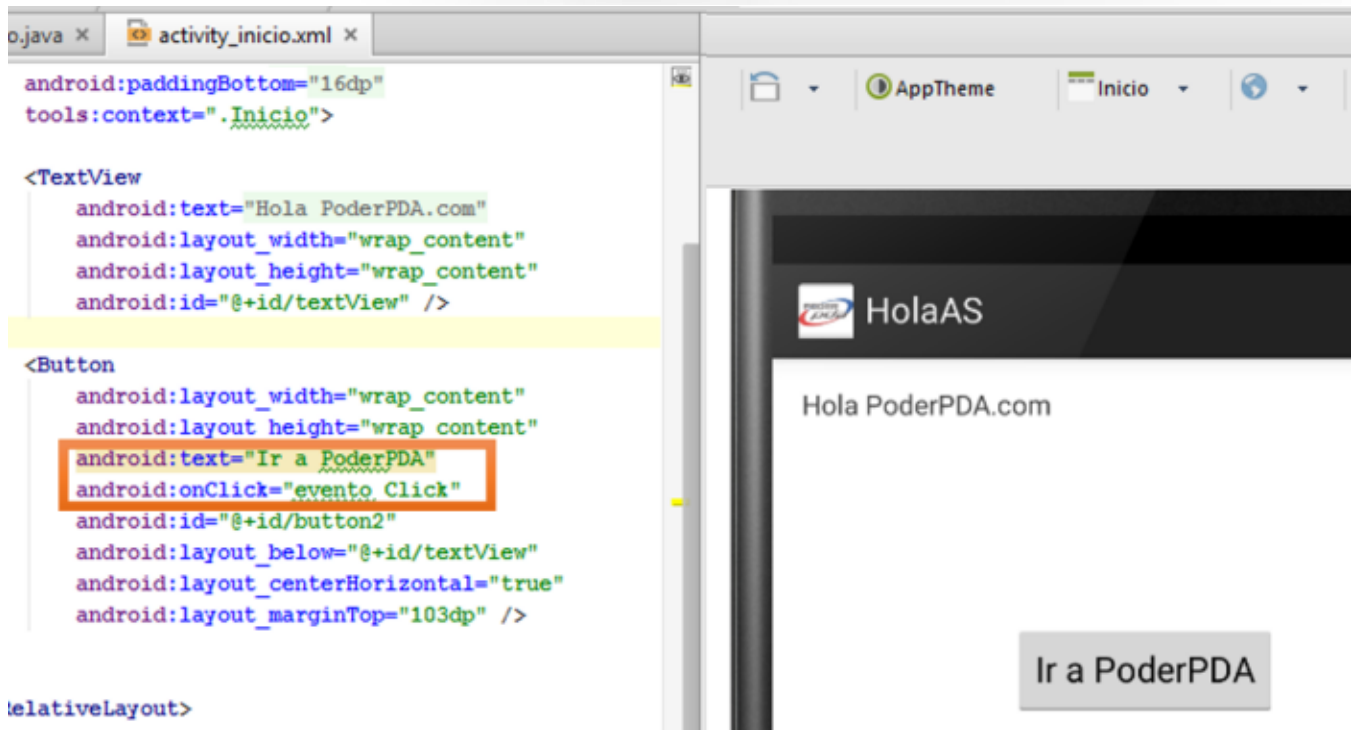


Agregando un Widget (botón)

Ahora cambiaremos el texto que muestra el boton e indicaremos el nombre del metodo que se ejecutara al presionar el boton (evento click)

- Seleccionar la pestaña Text, con ello, se mostrara el codigo XML.
- Buscamos el codigo correspondiente al boton y modificamos la propiedad text con el valor "Ir a PoderPDA"
- Agregamos una propiedad llamada onClick y le asignamos el valor evento_Click

android:text="Ir a PoderPDA"
android:onClick="evento_Click"



Modificando propiedades

Finalmente, es necesario insertar código Java, a fin de que el evento click funcione al presionar el botón.

- Abrirnos el archivo Inicio.java, localizado en la ruta src\main\java\cia.holaas
- Agregamos las referencias a las clases:

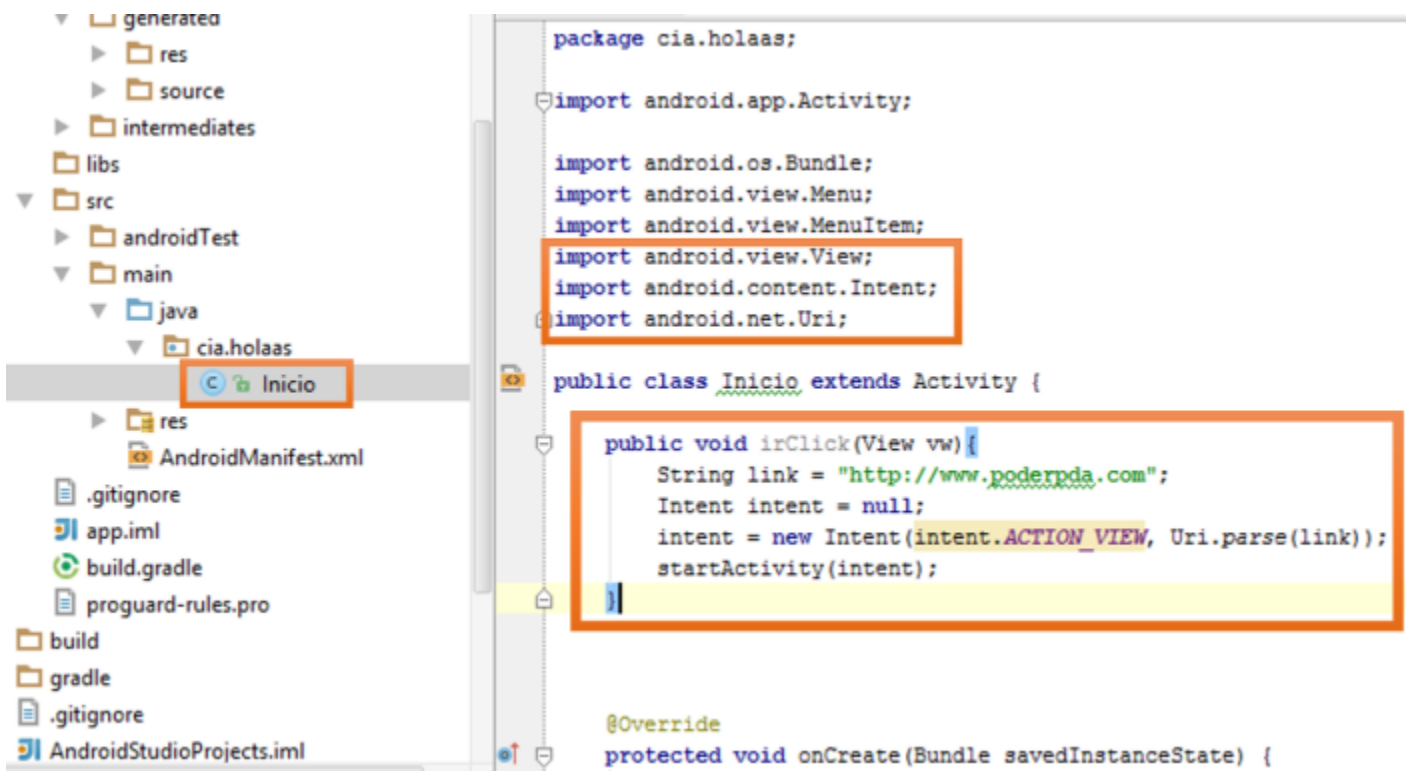
```
import android.view.View;
import android.content.Intent;
import android.net.Uri;
```

- *android.view.View*. Necesaria porque el método del evento click recibe un objeto de tipo View (en este caso un boton)
- *android.content.Intent*. Permitir la ejecución de tareas, en este caso, el redirigirnos a un navegador
- *android.net.Uri*. Para parsear una dirección de internet
- Crear el método para el evento click

```
public void evento_Click(View vw){
String link = "http://www.poderpda.com";
Intent intent = null;
intent = new Intent(intent.ACTION_VIEW, Uri.parse(link));
startActivity(intent);
}
```

donde:

- Declaramos una cadena que contiene la dirección del portal
- Creamos un objeto de tipo Intent
- Iniciamos en intent, redireccionando la ejecución de la aplicación hacia el navegador



Agregando código Java

INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO (III)

23/10/2014

por [pepe abel](#)

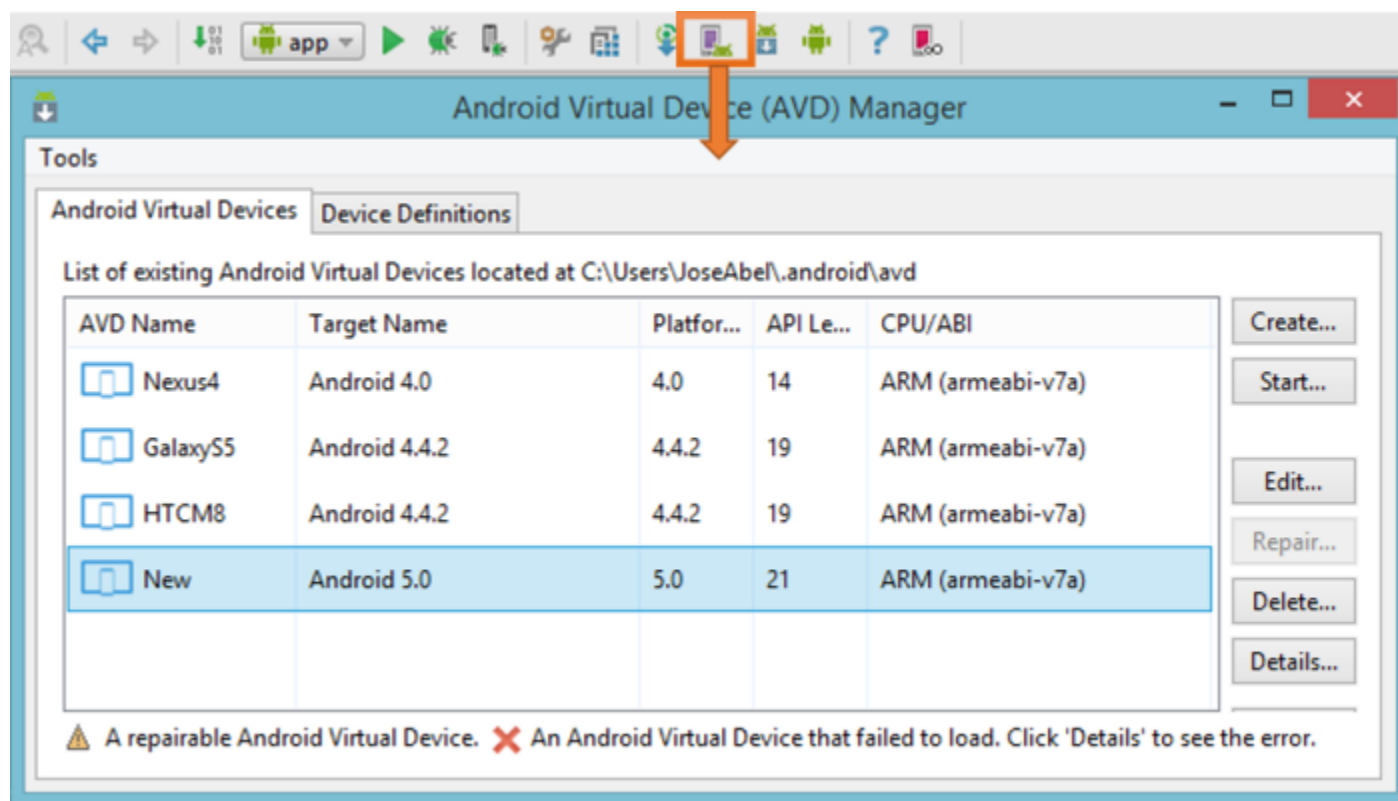
[Comentarios 0](#)

Nos encontramos ya con la tercera entrega del presente tutorial: Hemos creado un proyecto, revisado su estructura, modificado la interfaz gráfica e insertado código Java. Sin embargo, no se le puede llamar aplicación a un programa si este no se visualiza en un dispositivo para el cual fue diseñado, por ello, toca el turno de ejecutar nuestro pequeño proyecto desarrollado en **Android Studio** (AS) en un **emulador** o bien directamente en el **dispositivo físico**. Cada una de las opciones tiene sus ventajas y desventajas, de forma que realizaremos la práctica en ambos medios. Si acaso deseas ver las 2 entregas anteriores, las ligas aparecen justo **al final del post**.

Creación de un emulador

Es la opción más común para la ejecución de aplicaciones, ya que resulta relativamente sencillo crear un **Dispositivo Virtual Android (AVD)** y sobre todo, cuando no se tiene a la mano el dispositivo físico. Para entrar a detalle sobre la creación de los AVDs, puedes ingresar al tutorial **Android Virtual Device: Creando y Configurando Emuladores**, aunque dicho tutorial está centrado en el uso de Eclipse, el proceso es el mismo para AS, solo es necesario:

- Ejecutar el AVD Manager
- El proceso es similar, con algunas ligeras diferencias en el aspecto visual del asistente
- En este ejercicio, utilizare un emulador con un **Skin del Galaxy S5** y Android 4.4.2, pero para efectos prácticos cualquier otro emulador servirá



Accediendo al AVD Manager

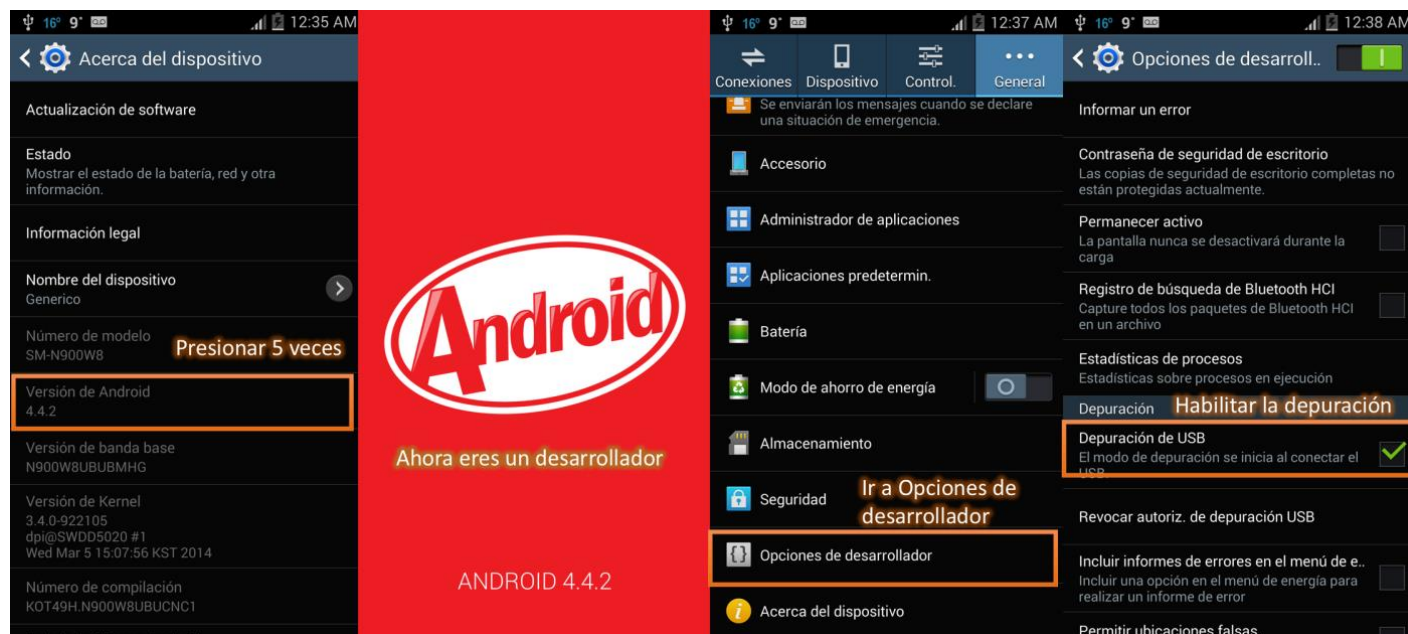
Configuración de un equipo físico

No es suficiente con conectar el dispositivo a través del cable **USB** a la computadora, ya que se tiene que **habilitar el modo desarrollador**:

- Ir a **Configuración > Opciones** de desarrollador
- Si no aparecen las Opciones de desarrollador, es necesario ir a **Configuración > Acerca del dispositivo** y presionar **5 veces** el renglón donde aparece la versión de Android. Enseguida aparecerá un mensaje indicado que **ahora eres un desarrollador**. Google ha realizado este

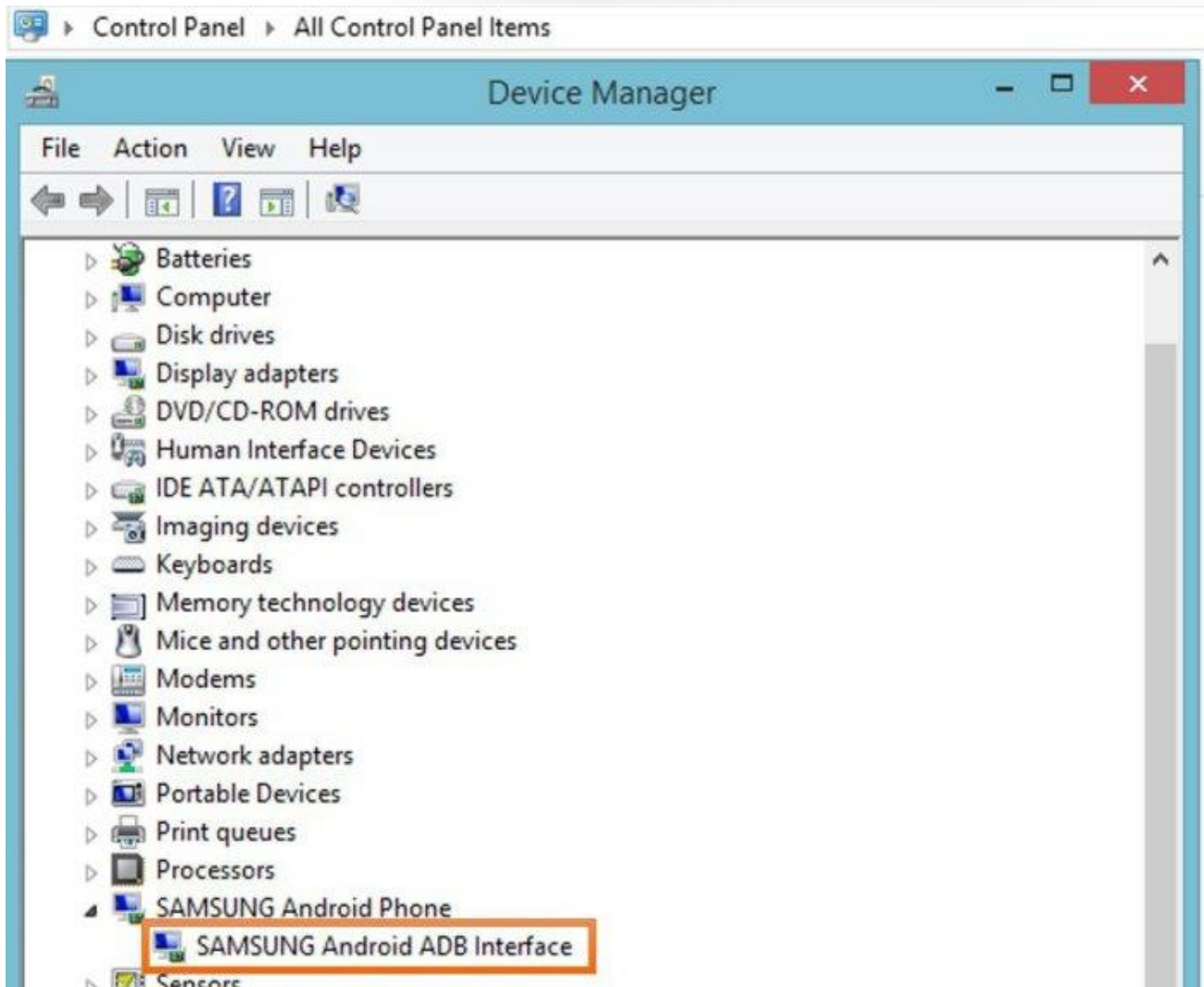
cambio a fin de que un usuario promedio no tenga acceso a estas configuraciones y provoque un mal funcionamiento del sistema.

- Entramos a las **Opciones de desarrollador** y habilitamos la **Depuración de USB**



Habilitar modo Desarrollador

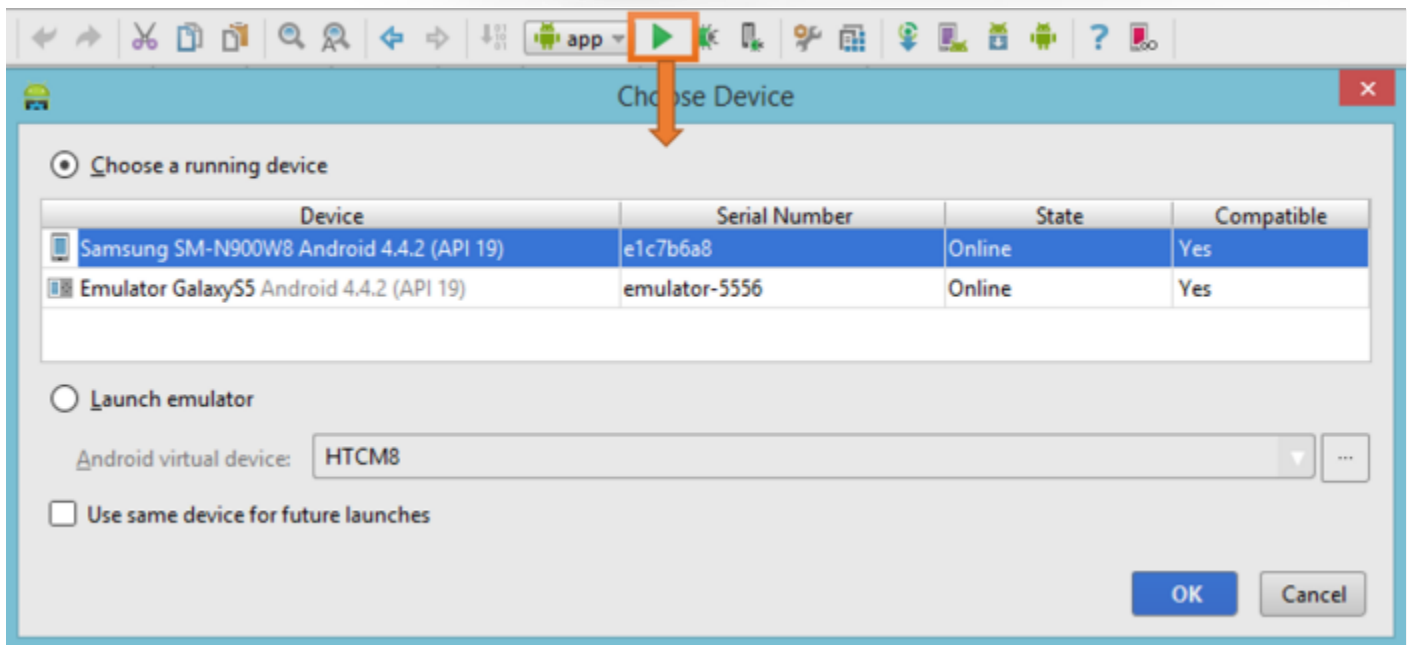
- El dispositivo debe ser reconocido dentro de la lista del sistema, para cerciorarse, es necesario acudir al **Panel de Control > Administrador de dispositivos**. De no ser así, será necesario acudir a la página del fabricante en busca de los drivers necesarios para su correcta configuración.



Driver del dispositivo instalado

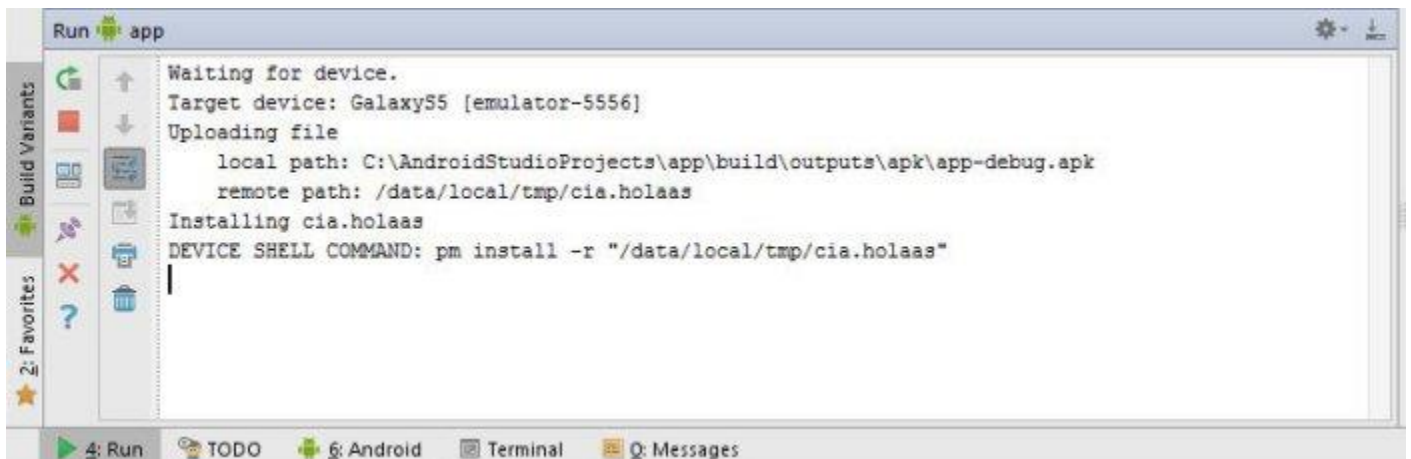
Ejecución de la aplicación

- Dentro de AS, presionar el botón **Run 'app'** (o bien Shift+F10). Aparece una ventana para elegir el dispositivo donde se ejecutará la aplicación
 - **Choose a running device.** Puede seleccionarse un dispositivo virtual que ya se esté ejecutando o un dispositivo físico que se encuentre conectado y configurado. En esta lista también se muestra el estado del dispositivo (en línea o no) y la compatibilidad
 - **Launch emulator.** Lanzar un emulador que previamente ha sido creado
- También es posible ejecutar más de uno a la vez, seleccionando todos los que se deseen
- Presionar el botón **OK**.



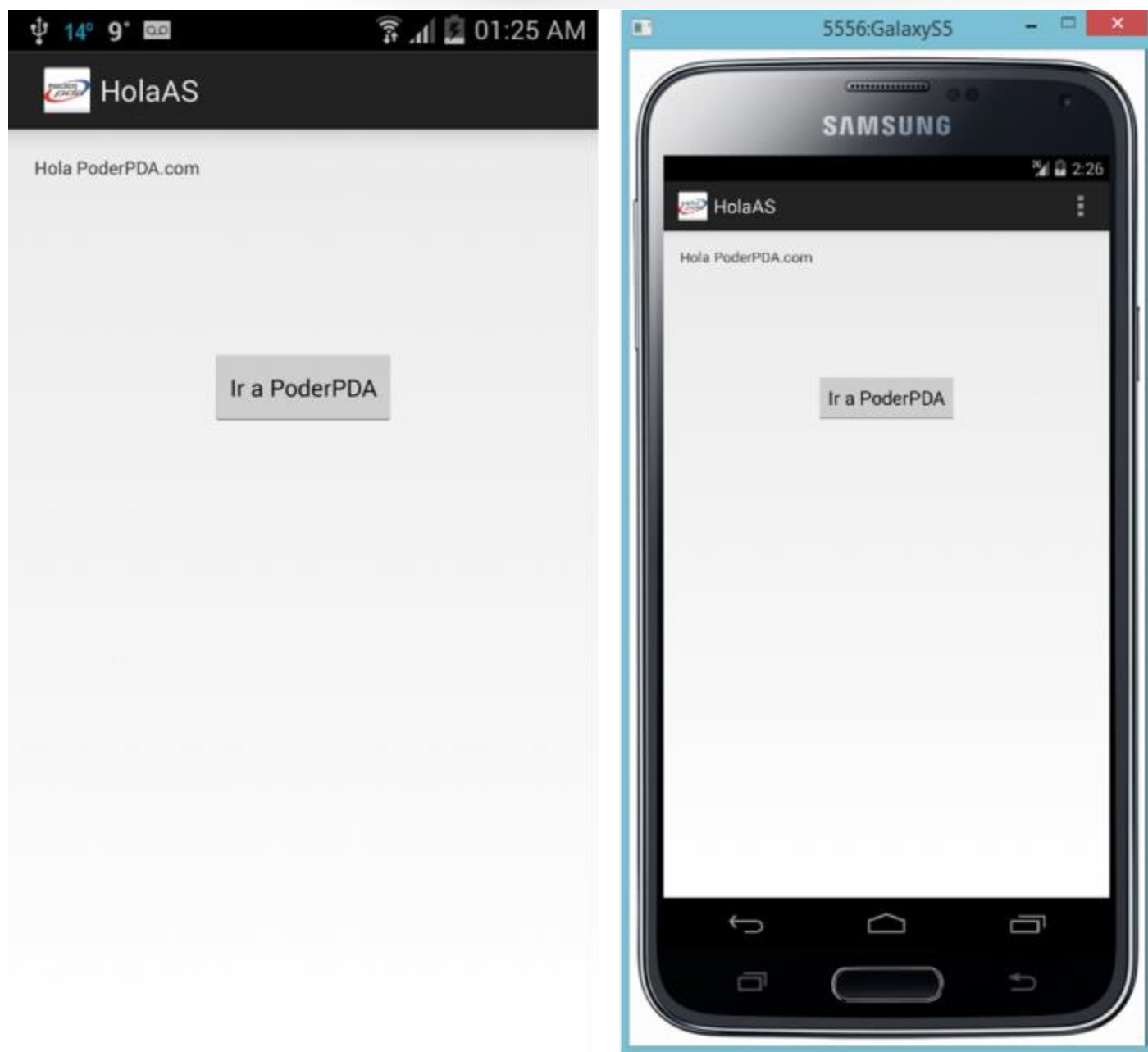
Ejecutar la Aplicación

- La aplicación se instala en los dispositivos seleccionados, tal y como se muestra en la ventana **RUN**



Pantalla Run

- Cuando la instalación finaliza, la aplicación se ejecutara mostrando la interfaz que hemos diseñado



Ejecución en un Dispositivo Físico y en un Emulador

- Al presionar el botón **Ir a PoderPDA**, lanzara el navegador con la dirección previamente configurada



Resultado de la ejecución del botón



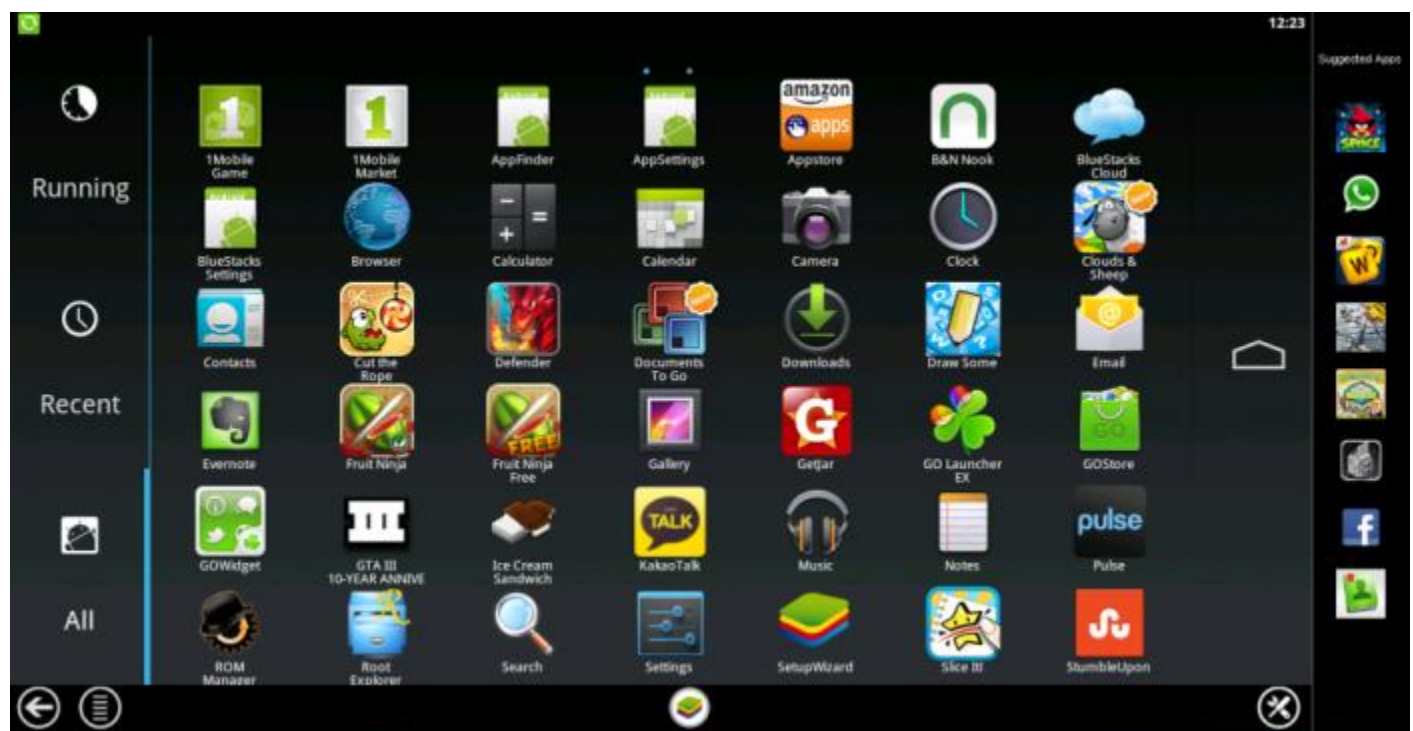
INTRODUCCIÓN AL DESARROLLO DE APLICACIONES CON ANDROID STUDIO (IV)

30/10/2014

por [pepe abel](#)

[Comentarios 0](#)

Hemos llegado a la **cuarta y última entrega** del tutorial introductorio a **Android Studio (AS)**. A estas alturas ya sabemos crear un nuevo proyecto con características específicas, conocemos la estructura del mismo, podemos editar la interfaz de usuario y agregar código java para eventos de los widgets, crear emuladores personalizados con skins específicos, configurar equipos físicos y ejecutar aplicaciones en los dispositivos configurados. Ahora trataremos algunos aspectos relacionados con la **mejora del desempeño de los emuladores** y su instalación directa mediante archivos en los dispositivos.



Emulación Android

Mejorando el desempeño

Cuando creaste tu emulador, seguramente observaste que el desempeño del mismo no era precisamente el mejor, sino todo lo contrario: una ejecución lenta y quizás desesperante, eso es normal, ya que como su nombre lo indica, es una emulación, lo cual implica traducir instrucciones nativas de una arquitectura **ARM** (*Advanced RISC Machine*) hacia una x86, emulando un teléfono completo, incluida la *CPU*, *GPU*, memoria, etc. Sin embargo, para mejorar este aspecto, es posible utilizar una de las dos opciones siguientes:

- **Snapshots.** Guardando imágenes del estado actual del emulador, con el objetivo de que en la siguiente ejecución la carga sea más rápida
- **Usando la GPU del Host.** Utilizando el hardware del equipo a fin de agilizar los procesos del emulador.

Si bien, estas opciones agilizan la velocidad del emulador, no resuelven el problema en su totalidad, ya que el comportamiento aún se mostrara lento. Ambas configuraciones se encuentran explicadas a detalle en el post **Android Virtual Device: Creando y Configurando Emuladores**.

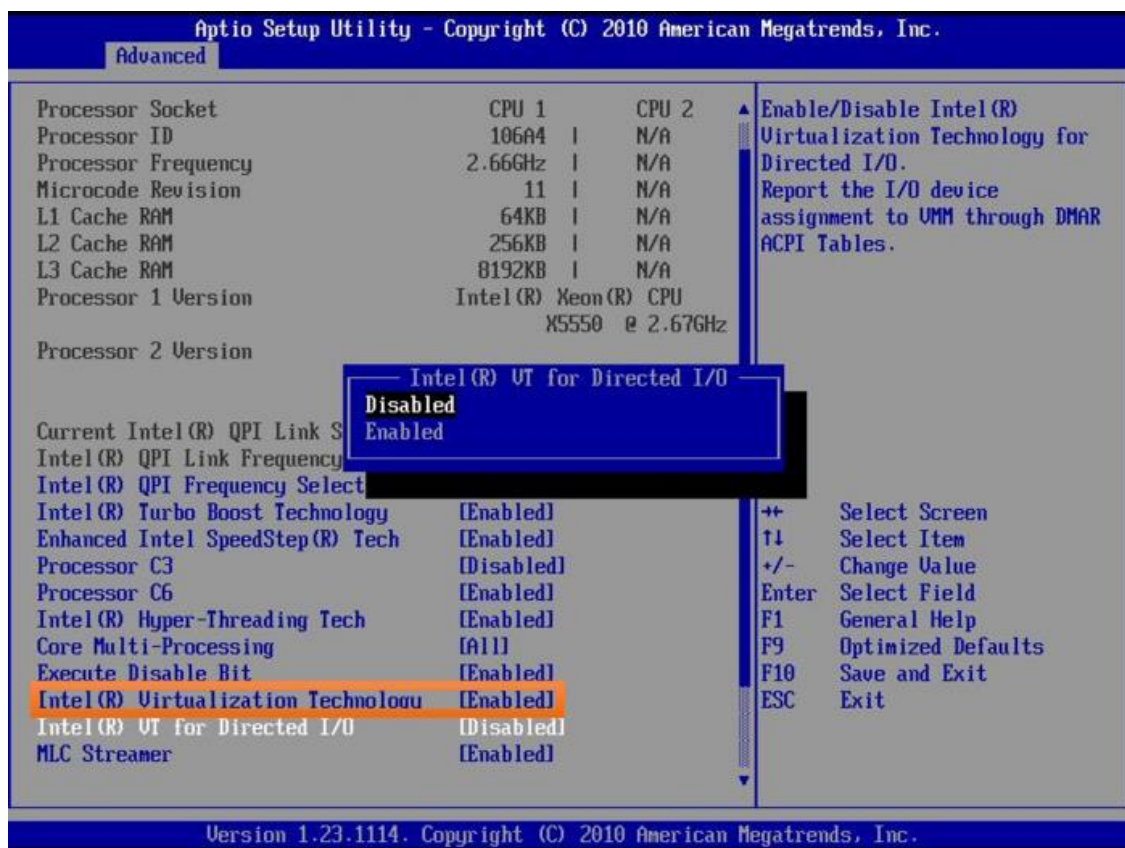
SD Card:

Size: MiB
 File: Browse...

Emulation Options: ☐ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

Opciones de emulación Android
 Intel x86 Emulator Accelerator



Activación de la virtualización desde el BIOS

Una opción con mejores resultados es el acelerador de Intel HAXM (Hardware Accelerated Execution Manager), el cual concede acceso directo al hardware, con lo cual el comportamiento mejora de manera significativa, haciendo del desarrollo una tarea más llevadera. El único problema es que tu equipo debe contar con un procesador Intel moderno (i3 en adelante) y además permitir la virtualización por hardware (configurada desde el BIOS), algo que por ejemplo un Core 2 Duo no puede llevar a cabo. De igual forma, los detalles de la configuración se encuentran en el post **Android Virtual Device: Creando y Configurando Emuladores**.

The image shows the installation of Intel HAXM. On the left, a file explorer window displays the contents of the 'extras\intel\Hardware_Accelerated_Execution_Manager' folder, including 'IntelHaxm.exe', 'Release Notes.txt', and 'source.properties'. An orange arrow points from this folder to the 'Intel® Software Tools for Android*' setup wizard. The wizard is in the 'Completed' stage, showing the memory limit for Intel HAXM (2.0 GB) and the 'Next' button. The text on the right side of the wizard reads: 'Completed the Intel® Hardware Accelerated Execution Manager Setup Wizard. Click the Finish button to exit the Setup Wizard. Intel Hardware Accelerated Execution Manager is now installed. Note: The memory reservation setting can be changed by running this installer again. Please refer to Intel® HAXM documentation for more information. [Launch Intel HAXM Documentation]'. The bottom of the wizard shows the 'Finish' button.

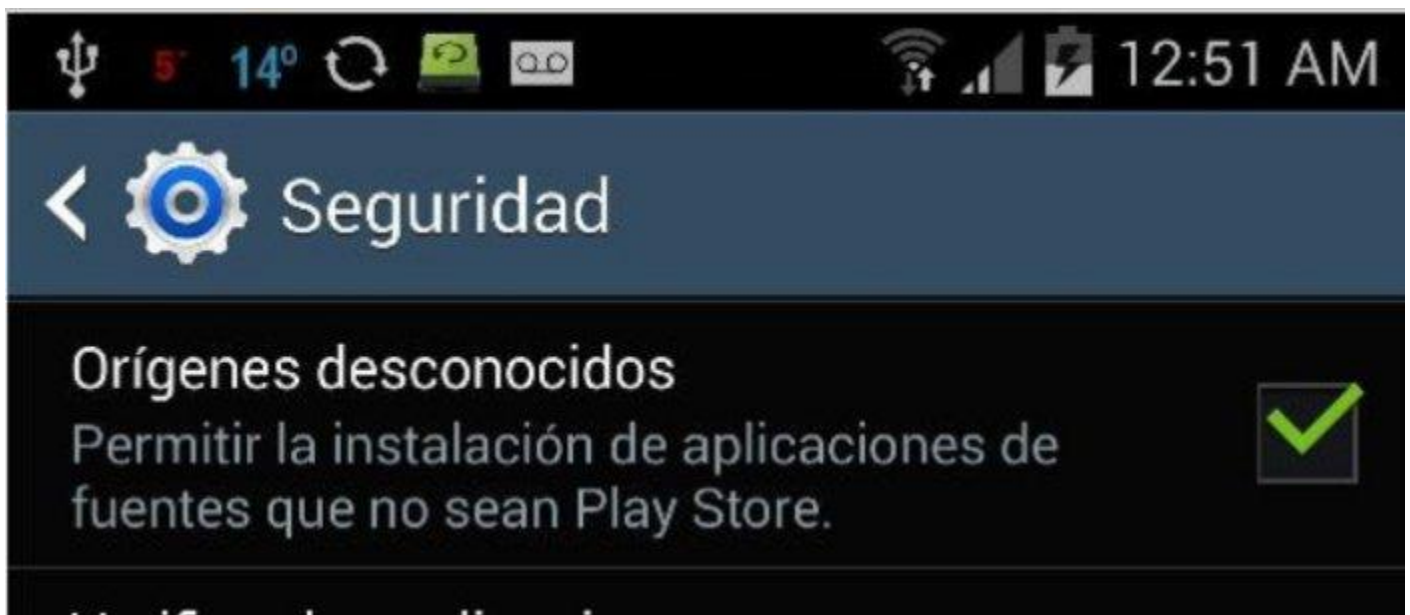
Instalación del acelerador Intel

Existen varias opciones más, entre ellas **BlueStacks**, **GenyMotion** y **Android x86**.

Instalación directa de la Aplicación

Ya hemos instalado nuestra aplicación en un emulador y en un dispositivo físico al momento de realizar la ejecución, sin embargo también es posible realizar la distribución mediante la tienda de aplicaciones Google Play o ejecutando el archivo directamente en el equipo, como actividad final, realizaremos la segunda opción. Aunque lo recomendable es liberar la aplicación en modo release, dado que es solo una prueba, utilizaremos la debug generada por default.

- Primero, es necesario que tengamos habilitada la instalación de aplicaciones de orígenes desconocidos, desde configuración > Seguridad

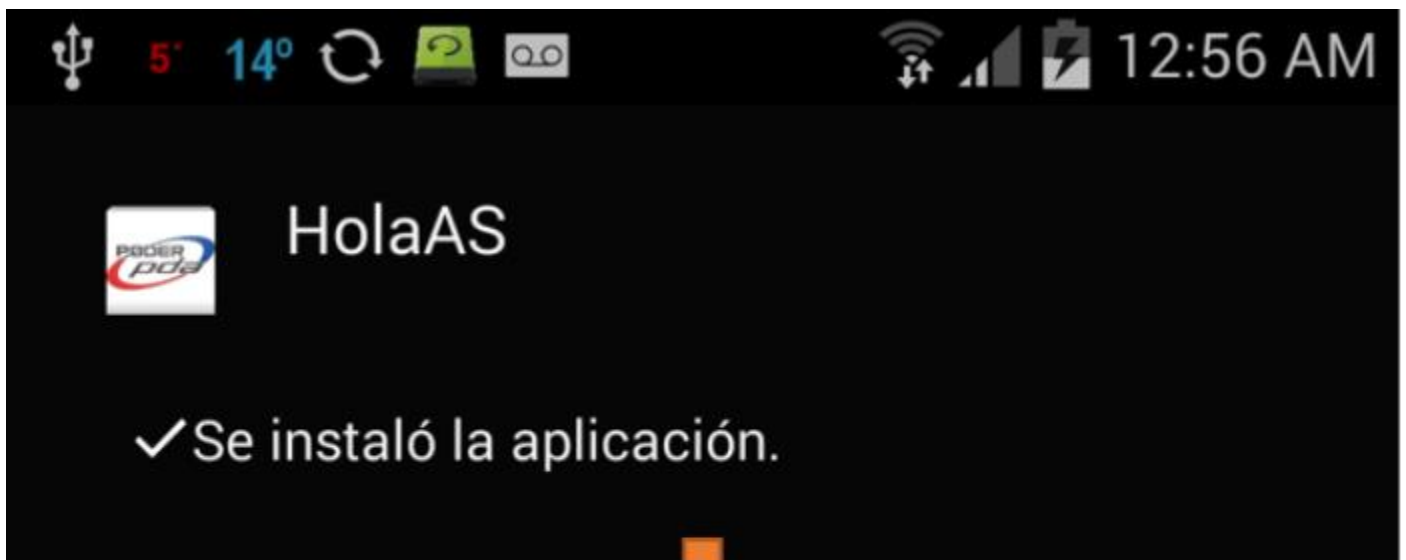
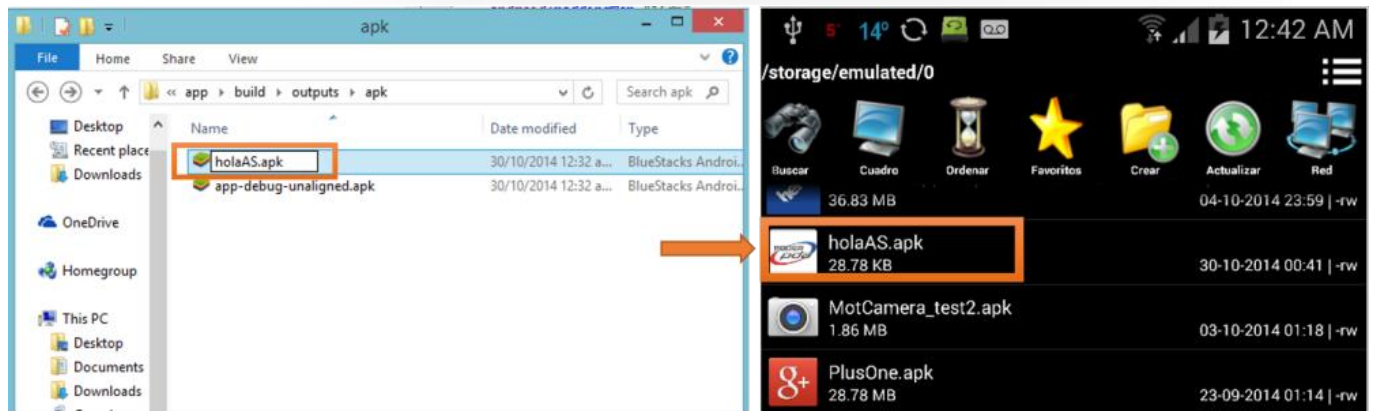


Opciones de seguridad

- Dentro de las carpetas del proyecto, acudir a la ruta **|app|build|outputs|apk**
- Dentro de esta carpeta existe un archivo llamado **app-debug.apk**, renombrarlo como **holaAS.apk** y copiarlo a la memoria del dispositivo donde se instalara
- Ingresar al explorador de archivos del dispositivo, localizar el archivo copiado y ejecutarlo

Instalación del archivo .apk

- Seguir las instrucciones para la instalación/actualización de la aplicación. Al final nos aparecerá un mensaje indicando que la instalación ha finalizado y que ya podemos abrirla



Instalación finalizada

Bibliografía

Autor:	Bill Phillips (Author), Chris Stewart (Author), Brian Hardy (Author), Kristin Marsicano (Author)
Título:	Android Programming: The Big Nerd Ranch Guide
País:	USA
Editorial:	Big Nerd Ranch Guides; 2 edition
Año de publicación:	2015

Autor:	Dawn Griffiths (Author), David Griffiths (Author)
Título:	Head First Android Development
País:	USA
Editorial:	O'Reilly Media; 1 edition
Año de publicación:	2015

Cibergrafía

Autor:	José Dimas Luján
Título:	Introducción a Android
Vínculo:	http://www.desarrolloweb.com/manuales/
Editor:	José Dimas Luján
Año de publicación:	17 de enero de 2014

Autor:	Carlos Picca
Título:	Java desde Cero
Vínculo:	http://codehero.co/series/java-desde-cero.html
Editor:	Carlos Picca
Año de publicación:	04/12/2013

Autor:	Enrique López Mañas
Título:	Android de 0 a 100
Vínculo:	https://www.video2brain.com/mx/cursos/android-de-0-a-100
Editor:	Enrique López Mañas
Año de publicación:	21/05/2014