

**¿Qué es Android?**

Android es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tablets o tabléfonos; y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, la compró. Android fue presentado en 2007 junto la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008. Los dispositivos de Android venden más que las ventas combinadas de Windows Phone e IOS.

El éxito del sistema operativo se ha convertido en objeto de litigios sobre patentes en el marco de las llamadas «Guerras por patentes de teléfonos inteligentes» (en inglés, Smartphone patent wars) entre las empresas de tecnología. Según documentos secretos filtrados en 2013 y 2014, el sistema operativo es uno de los objetivos de las agencias de inteligencia internacionales.

La versión básica de Android es conocida como Android Open Source Project (AOSP).

El 25 de junio de 2014 en la Conferencia de Desarrolladores Google I/O, Google mostró una evolución de la marca Android, con el fin de unificar tanto el hardware como el software y ampliar mercados.

Dejando un poco de lado los tecnicismos, Android es otra de las opciones de interfaces y características que podemos encontrar en teléfonos móviles; así como podemos identificar aspectos particulares de un Windows Phone 8, o un iPhone con IOS o incluso los Blackberry, también existen cosas muy específicas en los teléfonos que funcionan con Android.

Ahora volvámonos más técnicos. Android es un sistema operativo basado en el kernel de Linux, por esa razón tiene inmersas las características de ser libre, gratuito y multiplataforma. Cualquier desarrollador puede crear aplicaciones para Android sin la necesidad de pagar membresías anuales para obtener el kit de desarrollo (SDK).

Android utiliza una variación del lenguaje de programación Java que es diferente a la Java ME. Si has desarrollado en Java, seguramente conocerás lo que es trabajar con una máquina virtual que sirve para interpretar todo ese código que genera nuestro programa (bytecode) y pueda ejecutarse. Pues bien, Android, tiene una adaptación de esta máquina virtual y se llama **Dalvik**. Pese a las pesadillas que tengas de la virtual machine por default de Java, **Dalvik** es una excelente versión que optimiza muchas cosas en la plataforma y te sorprenderá saber que la rapidez de Android y de algunas aplicaciones se debe precisamente a esto.

### Reemplazo de Dalvik por ART

Hasta la versión 5.0, Android utiliza Dalvik como máquina virtual con la compilación just-in-time (JIT) para ejecutar Dalvik "dex-code" (Dalvik ejecutable), que es una traducción de Java bytecode. Siguiendo el principio JIT, además de la interpretación de la mayoría del código de la aplicación, Dalvik realiza la compilación y ejecución nativa de segmentos de código seleccionados que se ejecutan con frecuencia ("huellas") cada vez que se inicia una aplicación. Android 4.4 introdujo el ART (Android Runtime) como un nuevo entorno de ejecución, que compila el Java bytecode durante la instalación de una aplicación. Se convirtió en la única opción en tiempo de ejecución en la versión 5.0

Cabe mencionar que el sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla utilizando Java.

# Y en los orígenes alguien creó Android…

Android era un sistema operativo para móviles prácticamente desconocido propiedad de una empresa llamada **Android Inc**. hasta que en 2005 Google lo compró. Actualmente Andy Rubin, el creador de Android, trabaja como vicepresidente de ingeniería de Google y tiene bajo su mando este proyecto.

En noviembre de 2007 sólo circulaban rumores de que el gigante de Internet tenía en planes lanzar un proyecto para móviles, y precisamente por esas fechas se lanzó la Open Handset Alliance, que agrupaba a muchos fabricantes de teléfonos móviles, chipsets y Google y se proporcionó la primera versión de Android, junto con el SDK para que los programadores empezaran a crear sus aplicaciones para este sistema.

Aunque los inicios fueran un poco lentos, debido a que se lanzó el sistema operativo antes que el primer móvil, rápidamente se ha colocado como una plataforma que ya se ha ganado muchos adeptos y que ha demostrado una velocidad de madurez importante debido a los diferentes fabricantes que han adaptado interesantes piezas de hardware para acompañar las diferentes funciones de Android.

# Cosas curiosas de Android

* Las versiones de Android aparte del número tienen un nombre de un postre en idioma inglés. En cada versión el postre elegido empieza con una letra distinta siguiendo un orden alfabético:

|  |  |  |  |
| --- | --- | --- | --- |
| A | [Apple Pie](https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android#Android_1.0_Apple_Pie) | 1.0 | [Tarta de manzana](https://es.wikipedia.org/wiki/Tarta_de_manzana) |
| B | [Banana Bread](https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android#Android_1.1_Banana_Bread) | 1.1 | [Pan de plátano](https://es.wikipedia.org/wiki/Pan_de_banana) |
| C | [Cupcake](https://es.wikipedia.org/wiki/Android_Cupcake) | 1.5 | [Cupcake](https://es.wikipedia.org/wiki/Cupcake) |
| D | [Donut](https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android#Android_1.6_Donut) | 1.6 | [Rosquilla](https://es.wikipedia.org/wiki/Rosquilla) *o* [dónut](https://es.wikipedia.org/wiki/D%C3%B3nut) |
| E | [Éclair](https://es.wikipedia.org/wiki/Android_Eclair) | 2.0/2.1 | [Pepito](https://es.wikipedia.org/wiki/Pepito) *o* relámpago |
| F | [Froyo](https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android#Android_2.2.x_Froyo) | 2.2 | [Yogur helado](https://es.wikipedia.org/wiki/Yogur_helado) |
| G | [Gingerbread](https://es.wikipedia.org/wiki/Android_Gingerbread) | 2.3 | [Pan de jengibre](https://es.wikipedia.org/wiki/Pan_de_jengibre) |
| H | [Honeycomb](https://es.wikipedia.org/wiki/Android_Honeycomb) | 3.0 - 3.1 - 3.2 | [Panal](https://es.wikipedia.org/wiki/Panal) |
| I | [Ice Cream Sandwich](https://es.wikipedia.org/wiki/Android_Ice_Cream_Sandwich) | 4.0 | [Sándwich de helado](https://es.wikipedia.org/wiki/S%C3%A1ndwich_de_helado) |
| J | [Jelly Bean](https://es.wikipedia.org/wiki/Android_Jelly_Bean) | 4.1 - 4.2 - 4.3 | [Gominola](https://es.wikipedia.org/wiki/Pastilla_de_goma) *o* [pastilla de goma](https://es.wikipedia.org/wiki/Pastilla_de_goma) |
| K | [KitKat](https://es.wikipedia.org/wiki/Android_KitKat) | 4.4 | [Kit Kat](https://es.wikipedia.org/wiki/Kit_Kat) |
| L | [Lollipop](https://es.wikipedia.org/wiki/Android_Lollipop) | 5.0/5.1 | [Piruleta](https://es.wikipedia.org/wiki/Piruleta)[*64*](https://es.wikipedia.org/wiki/Android#cite_note-64) |
| M | [Marshmallow](https://es.wikipedia.org/wiki/Android_Marshmallow) | 6.0 | [Malvavisco](https://es.wikipedia.org/wiki/Malvavisco) *o* nube [*65*](https://es.wikipedia.org/wiki/Android#cite_note-65) |
| N | [Nougat](https://es.wikipedia.org/wiki/Android_Nougat) | 7.0 | [Turrón](https://es.wikipedia.org/wiki/Turr%C3%B3n) |

* El logotipo de Android fue diseñado con la fuente Droid, hecha por Ascender Corporation.
* El verde es el color del robot Android que representa el sistema operativo. El color print es PMS 376C y color GBN en hexadecimal es #A4C639, como se específica en la Android Brand Guidelines que puedes consultar para ocupar en la personalización de algún logo para tu aplicación o proyecto.

# Arquitectura de Android

Para empezar con el desarrollo de aplicaciones en Android es importante conocer cómo está estructurado este sistema operativo. A esto le llamamos arquitectura y en el caso de Android está formada por varias capas que facilitan al desarrollador la creación de aplicaciones. Además, esta distribución permite acceder a las capas más bajas mediante el uso de librerías para que así el desarrollador no tenga que programar a bajo nivel las funcionalidades necesarias para que una aplicación haga uso de los componentes de hardware de los teléfonos.

Cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones, es por ello que a este tipo de arquitectura se le conoce también como *pila*. Para entender mejor, a continuación, cito el diagrama de la arquitectura:



Los componentes principales del sistema operativo de Android (cada sección se describe en detalle):

**Aplicaciones**: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

**Marco de trabajo de aplicaciones**: los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

**Bibliotecas**: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

**Runtime de Android**: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecutaba hasta la versión 5.0 archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx". Desde la versión 5.0 utiliza el ART, que compila totalmente al momento de instalación de la aplicación.

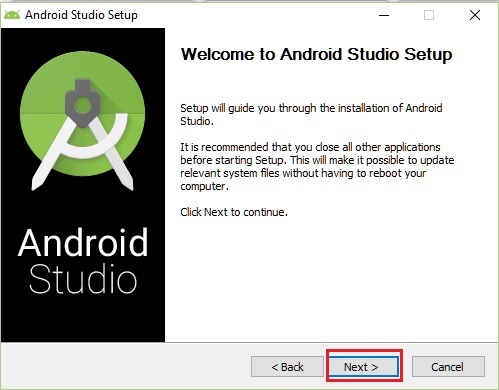
**Núcleo Linux**: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

**Descarga de herramientas necesarios para programar para Android con el Android Studio**

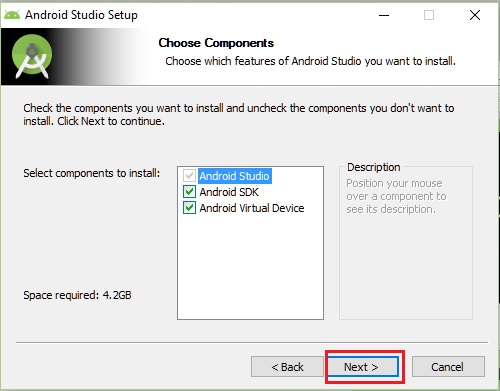
1 - Primero debemos instalar el compilador de Java y la máquina virtual. Estas herramientas las podemos descargar de:  
[Java SE Development Kit (JDK).](http://www.oracle.com/technetwork/es/java/javase/downloads/index.html)

2 - El segundo paso es la descarga del Android Studio (que contiene todo lo necesario para comenzar el desarrollo de aplicaciones en Android), lo hacemos del sitio :  
[Android Studio.](https://developer.android.com/studio/index.html?hl=es)

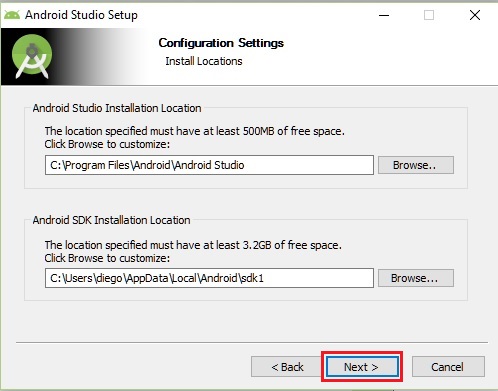
Ahora procedemos a su instalación en el equipo ejecutando el archivo que acabamos de descargar:



Dejamos por defecto para que instale el Android Studio, el Android SDK, Android Virtual Device etc.:

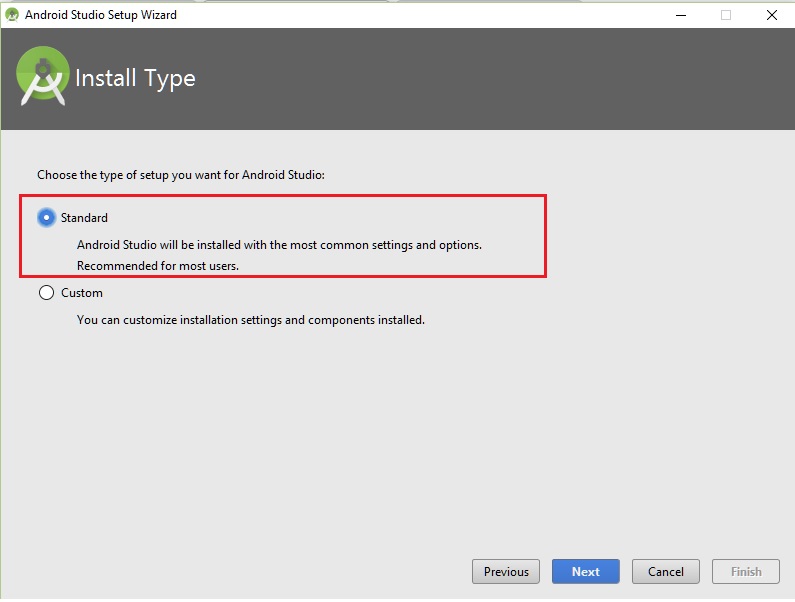


Dejamos con los valores por defecto en las siguientes pantallas de instalación.

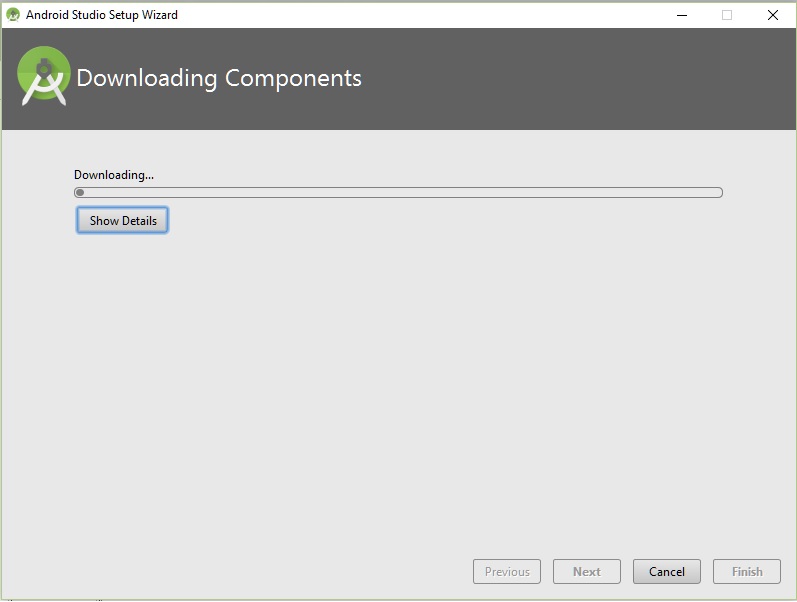


3 - El tercer paso es ejecutar el Android Studio para poder empezar a desarrollar para Android:  
Ejecución Android Studio

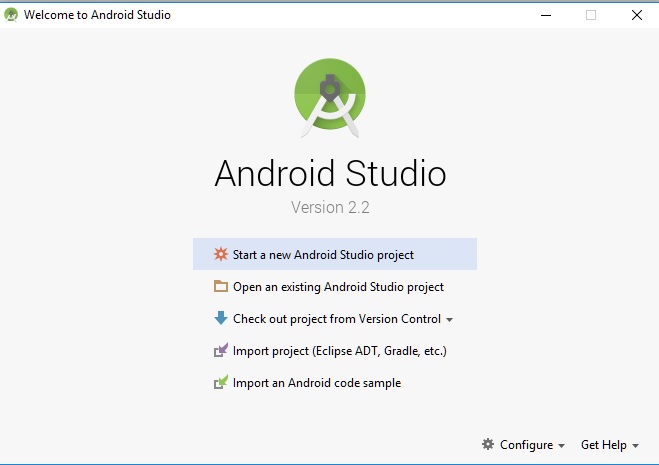
La primera vez que lo ejecutamos elegimos el tipo de configuración "Standard":



Seguidamente se instalan las últimas componentes necesarias:



Finalmente aparece el siguiente diálogo desde donde podemos crear nuestro primer proyecto:



Desde esta interfaz siempre procedemos para crear un nuevo proyecto.

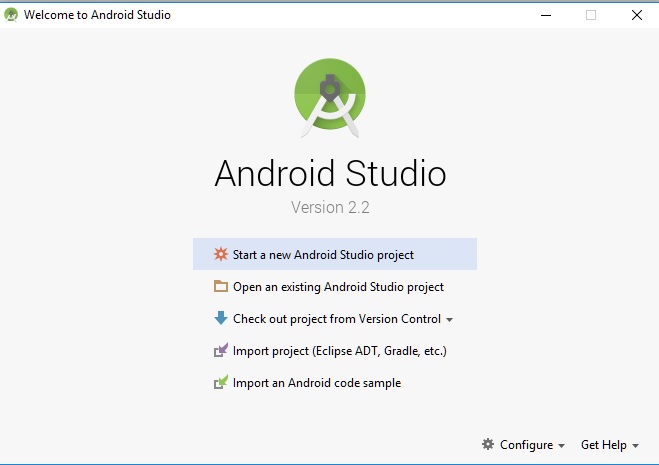
**Programa Básico de Java con Android**

**Obtenido de:**

**http://www.tutorialesprogramacionya.com/javaya/androidya/androidstudioya/detalleconcepto.php?codigo=2&inicio=0**

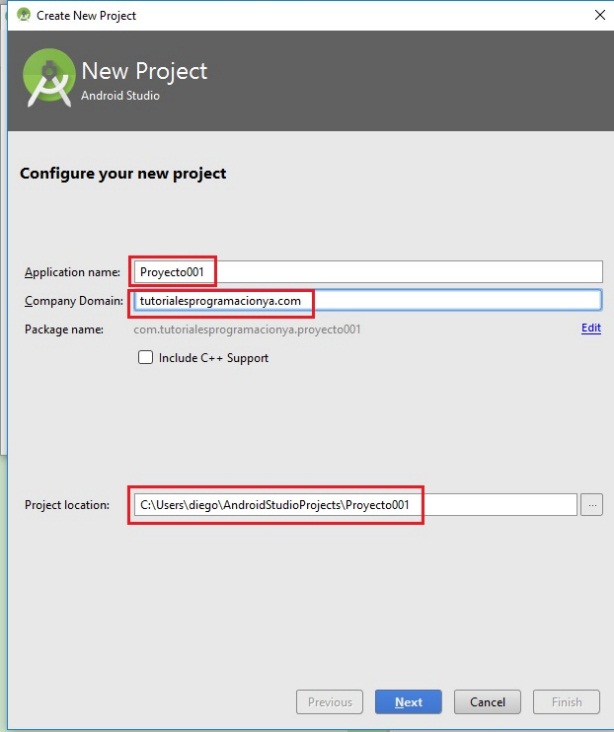
Utilizaremos el programa de Android Studio para crear un nuevo proyecto de Android. Un proyecto de Android contiene un programa de java y otros datos en una serie de archivos, que son utilizados por el programa o por el sistema. Un programa de java es una colección de clases, contenidas en uno o varios archivos fi1.java, file2.java,

Una vez que iniciamos el entorno del Android Studio aparece el diálogo principal:

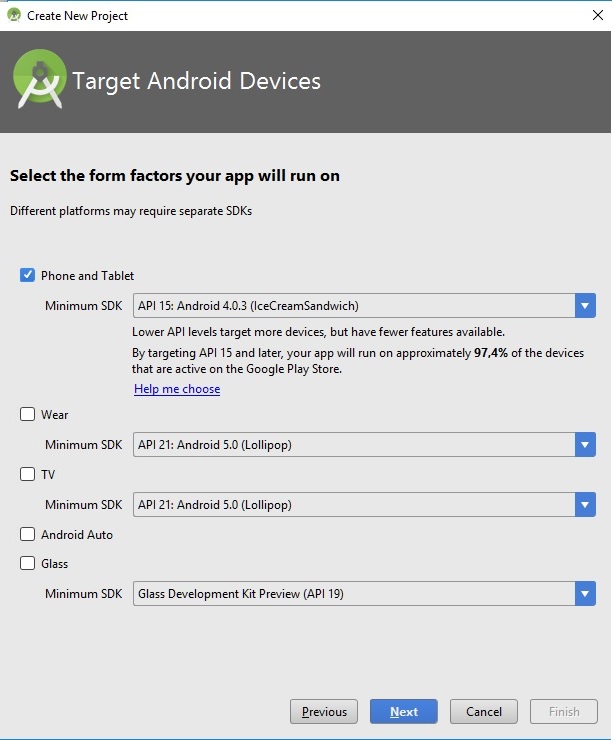


Elegimos la opción "Start a New Android Studio project"

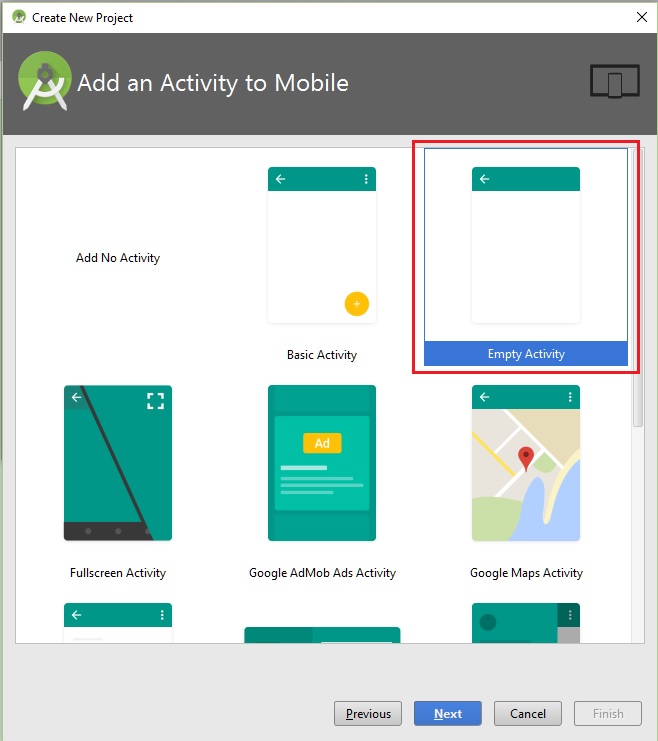
Ahora aparecerán una serie de ventanas para configurar el proyecto, el primer diálogo debemos especificar el Nombre de la aplicación, la url de nuestra empresa (que será el nombre del paquete que asigna java para los archivos fuentes) y la ubicación en el disco de nuestro proyecto:



En el segundo diálogo procedemos a especificar la versión de Android mínima donde se ejecutará la aplicación que desarrollemos (dejaremos la versión 4.0.3):



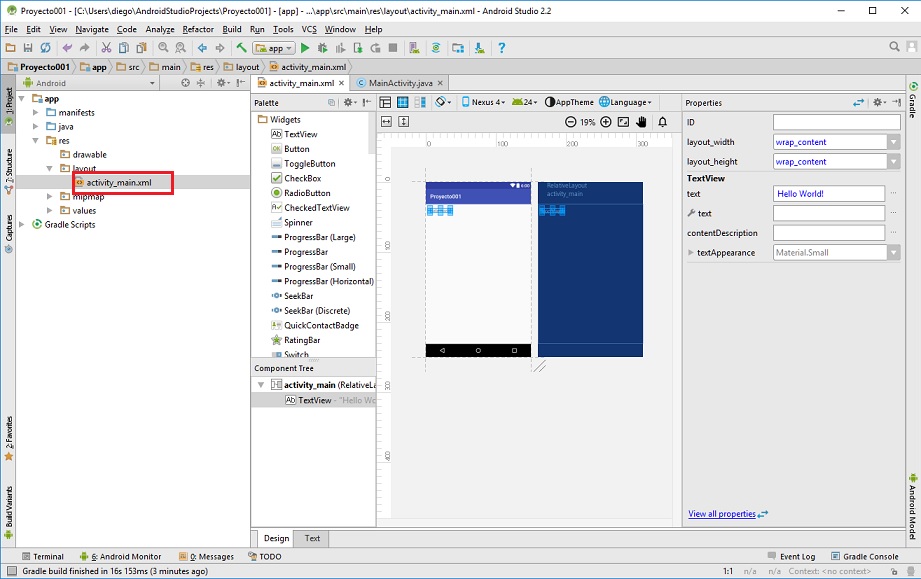
El tercer diálogo especificamos el esqueleto básico de nuestra aplicación, seleccionaremos "Empty Activity" si tenemos el Android Studio 2.2:



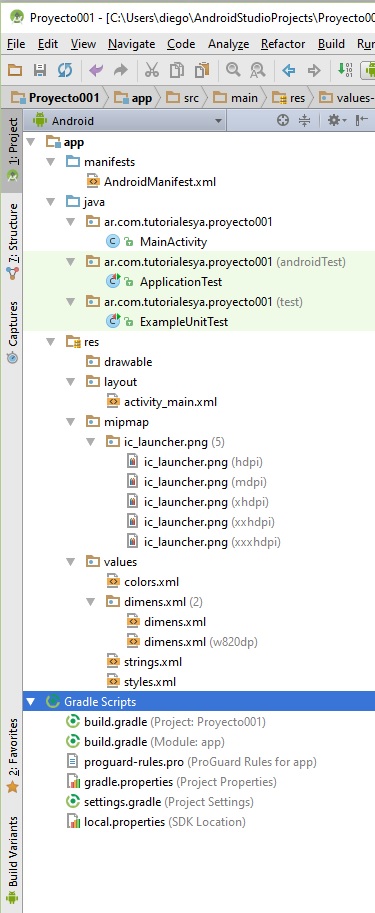
Finalmente, el último diálogo tenemos que indicar el nombre de la ventana principal de la aplicación (Activity Name) y otros datos más que veremos a lo largo del curso (dejaremos con los nombres por defecto que propone Android Studio):



Tenemos finalmente creado nuestro primer proyecto en Android Studio y podemos ahora ver el entorno del Android Studio para codificar la aplicación:



El Android Studio nos genera todos los directorios y archivos básicos para iniciar nuestro proyecto, los podemos ver en el lado izquierdo del entorno de desarrollo:

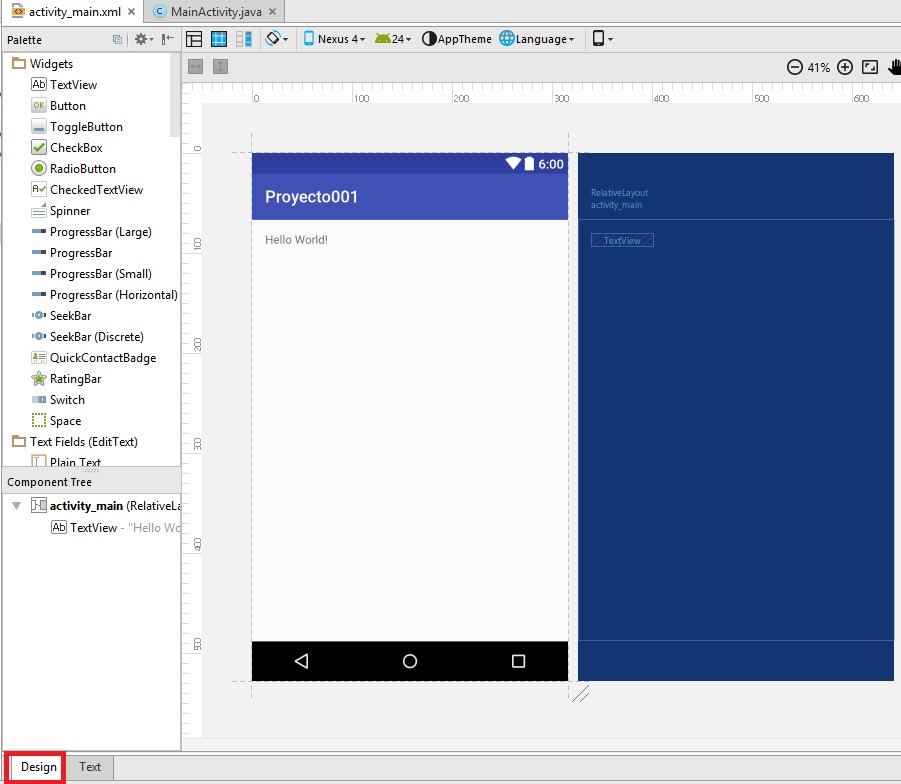


Archivos del proyecto en la vista de Android

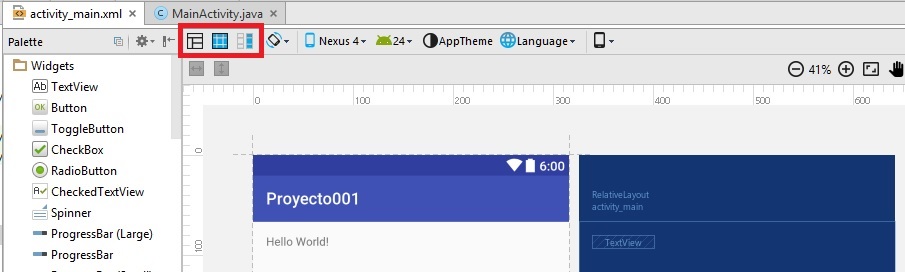
La interfaz visual de nuestro programa para Android se almacena en un archivo XML en la carpeta res, subcarpeta layout y el archivo se llama activity\_main.xml. En esta carpeta tenemos creada nuestra primera pantalla.

Al seleccionar este archivo el Android Studio nos permite visualizar el contenido en "Design" o "Text" (es decir en vista de diseño o en vista de código):

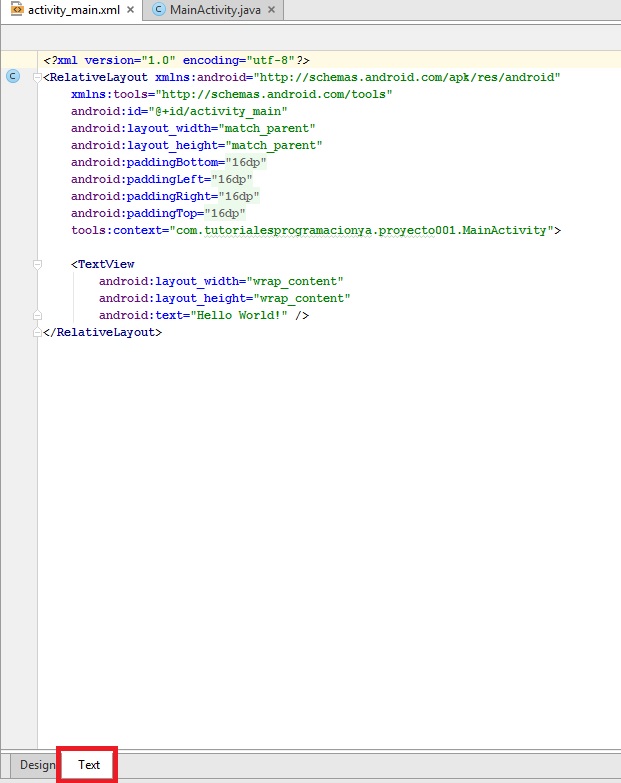
Vista de diseño:



A partir de la versión 2.2 del Android Studio tenemos la vista "blueprint" que nos nuestra una interfaz simplificada muy útil cuando tenemos pantallas complejas que veremos más adelante. Podemos ver solo la vista de diseño o "blueprint" seleccionando alguno de los botones que aparecen aquí:



Vista de código:

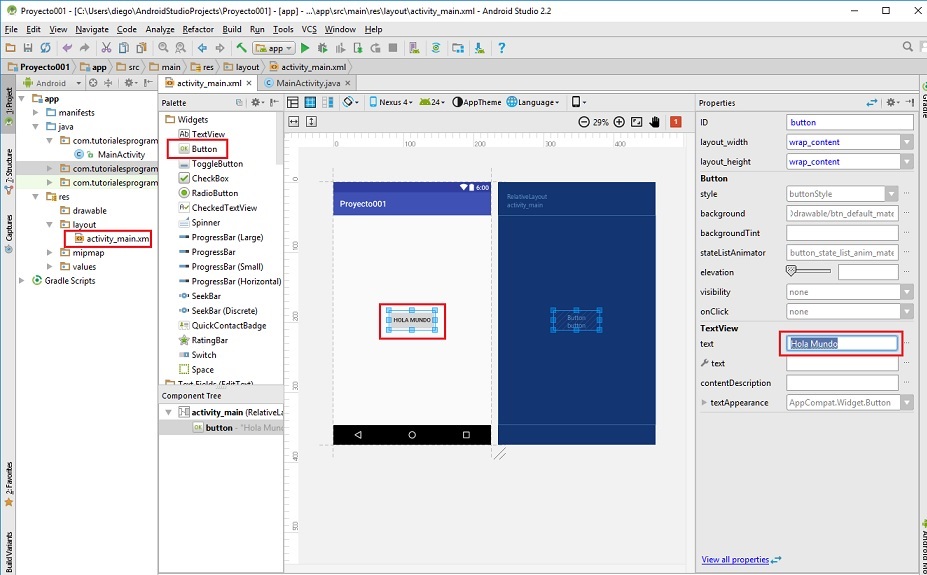


El Android Studio ya insertó un control de tipo RelativeLayout que permite ingresar controles visuales alineados a los bordes y a otros controles que haya en la ventana (más adelante analizaremos este layout)

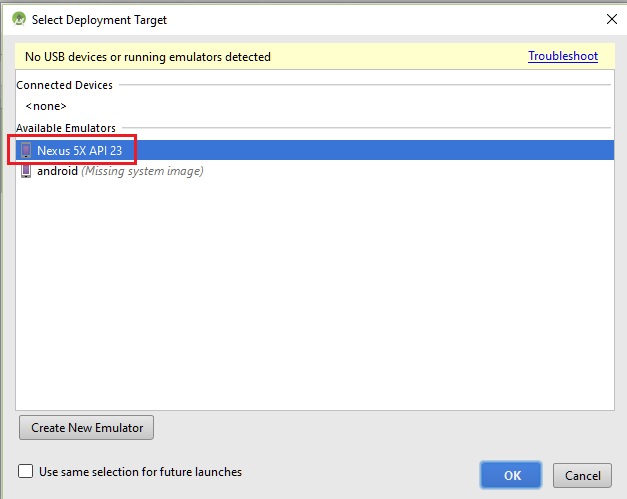
Ya veremos que podemos modificar todo este archivo para que se adapte a la aplicación que queremos desarrollar.

A lo largo de este curso iremos viendo los objetivos de cada una de las secciones que cuenta el Android Studio para implementar la interfaz, codificar en java las funcionalidades de la aplicación etc.

Antes de probar la aplicación en el emulador de un dispositivo Android procederemos a hacer un pequeño cambio a la interfaz que aparece en el celular: borraremos la label que dice "Hello World" (simplemente seleccionando con el mouse dicho elemento y presionando la tecla delete, podemos seleccionarla de cualquiera de las dos interfaces "Design" o "blueprint") y de la "Palette" arrastraremos un "Button" al centro del celular y en la ventana "Properties" estando seleccionado el "Button" cambiaremos la propiedad "text" por la cadena "Hola Mundo":



Para ejecutar la aplicación presionamos el triángulo verde o seleccionamos del menú de opciones "Run -> Run app" y en este diálogo procedemos a dejar seleccionado el emulador por defecto que aparece (Nexus 5X) y presionamos el botón "OK" (si no tiene ningún emulador puede crear uno):



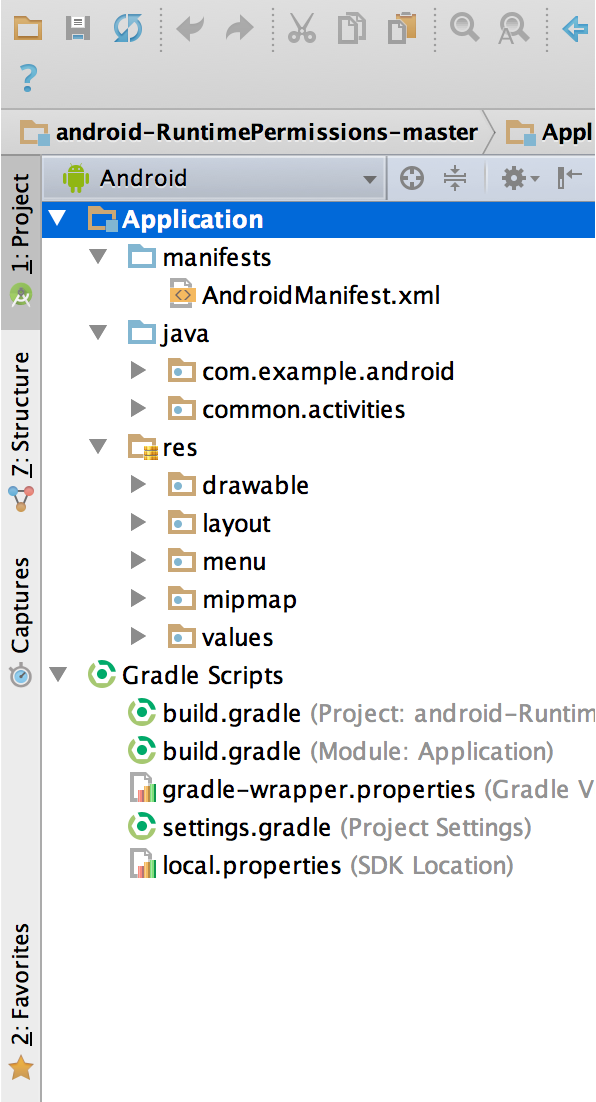
Luego de un rato aparecerá el emulador de Android en pantalla (el arranque del emulador puede llevar más de un minuto), es IMPORTANTE tener en cuenta que una vez que el emulador se ha arrancado no lo debemos cerrar cada vez que hacemos cambios en nuestra aplicación o codificamos otras aplicaciones, sino que volvemos a ejecutar la aplicación con los cambios y al estar el emulador corriendo el tiempo que tarda hasta que aparece nuestro programa en el emulador es muy reducido.

Cuando terminó de cargarse el emulador debe aparecer nuestra aplicación ejecutándose:



**Estructura del proyecto**

Obtenido de: https://developer.android.com/studio/intro/index.html?hl=es-419



Cada proyecto en Android Studio contiene uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

* Módulos de apps para Android
* Módulos de bibliotecas
* Módulos de Google App Engine

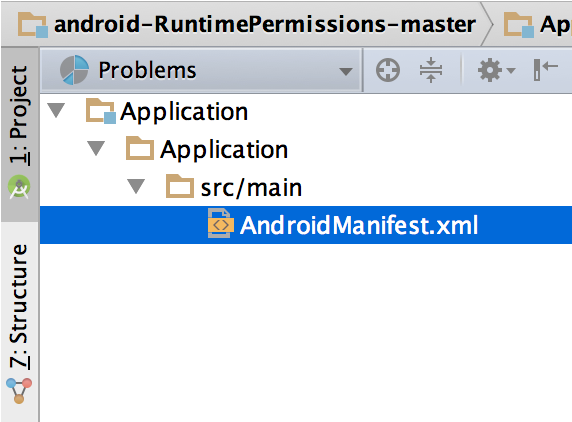
De forma predeterminada, en Android Studio se muestran los archivos de tu proyecto en la vista de proyectos de Android, como se muestra en la figura de arriba. Esta vista está organizada en módulos para que puedas acceder rápidamente a los archivos de origen claves de tu proyecto.

Todos los archivos de compilación son visibles en el nivel superior de **Secuencias de comando de Gradle** y cada módulo de la aplicación contiene las siguientes carpetas:

* **manifiestos**: contiene el archivo AndroidManifest.xml.
* **java**: contiene los archivos de código fuente de Java, incluido el código de prueba JUnit.
* **res**: Contiene todos los recursos, como diseños XML, cadenas de IU e imágenes de mapa de bits.

La estructura del proyecto para Android en el disco difiere de esta representación plana. Para ver la estructura de archivos real del proyecto, selecciona **Project** en la lista desplegable **Project** (en la figura 1 se muestra como **Android**).

También puedes personalizar la vista de los archivos del proyecto para concentrarte en aspectos específicos del desarrollo de tu app. Por ejemplo, al seleccionar la vista **Problems** de tu proyecto, aparecerán enlaces a los archivos de origen que contengan errores conocidos de codificación y sintaxis, como una etiqueta de cierre faltante para un elemento XML en un archivo de diseño.



Archivos del proyecto en la vista Problems, en la que se muestra un archivo de diseño con un problema.

**Interfaz de usuario**

La ventana principal de Android Studio consta de varias áreas lógicas que se identifican en la siguiente figura.



Ventana principal de Android Studio.

1. La **barra de herramientas** te permite realizar una gran variedad de acciones, como la ejecución de tu app y el inicio de herramientas de Android.
2. La **barra de navegación** te ayuda a explorar tu proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana **Project**.
3. La **ventana del editor** es el área en la que puedes crear y modificar código. Según el tipo de archivo actual, el editor puede cambiar. Al visualizar un archivo de diseño, por ejemplo, el editor muestra el Editor de diseño.
4. Las **ventanas de herramientas** te permiten acceder a tareas específicas, como la administración de proyectos, la búsqueda y los controles de versión, entre otras. Puedes expandirlas y contraerlas.
5. En la **barra de estado** se muestra el estado de tu proyecto y el IDE, además de advertencias o mensajes.

Puedes organizar la ventana principal para tener más espacio en pantalla ocultando o desplazando barras y ventanas de herramientas. También puedes usar combinaciones de teclas para acceder a la mayoría de las funciones del IDE.

En cualquier momento, puedes realizar búsquedas en tu código fuente, bases de datos, acciones, elementos de la interfaz de usuario, etc., presionando dos veces la tecla Shift o haciendo clic en la lupa que se encuentra en la esquina superior derecha de la ventana de Android Studio. Esto puede ser muy útil, por ejemplo, si intentas localizar una acción específica del IDE que olvidaste cómo activar.

**Ventanas de herramientas**

En lugar de usar perspectivas preestablecidas, Android Studio sigue tu contexto y te ofrece automáticamente ventanas de herramientas relevantes mientras trabajas. De forma predeterminada, las ventanas de herramientas usadas con mayor frecuencia se fijan en la barra de ventanas de herramientas en los bordes de la ventana de la aplicación.

* Para expandir o contraer una ventana de herramientas, haz clic en el nombre de la herramienta en la barra de la ventana de herramientas. También puedes arrastrar, anclar, desanclar, adjuntar y ocultar ventanas de herramientas.
* Para volver al diseño predeterminado actual de ventanas de herramientas, haz clic en **Window > Restore Default Layout** o personaliza tu diseño predeterminado haciendo clic en **Window > Store Current Layout as Default**.
* Para mostrar u ocultar la barra de ventanas de herramientas completa, haz clic en el ícono de ventana https://developer.android.com/studio/images/intro/window-icon_2-1_2x.png?hl=es-419en la esquina inferior izquierda de la ventana de Android Studio.
* Para localizar una ventana de herramientas específica, posiciona el puntero sobre el ícono de ventana y selecciona la ventana de herramientas en el menú.

También puedes usar combinaciones de teclas para abrir ventanas de herramientas. En la Tabla 1 se muestran las combinaciones de teclas para las ventanas más comunes.

*Tabla de Combinaciones de teclas para algunas ventanas de herramientas útiles.*

|  |  |  |
| --- | --- | --- |
| **Ventana de herramientas** | **Windows y Linux** | **Mac** |
| Proyecto | **Alt+1** | **Comando+1** |
| Control de versión | **Alt+9** | **Comando+9** |
| Ejecutar | **Shift+F10** | **Control+R** |
| Depurar | **Shift+F9** | **Control+D** |
| Android Monitor | **Alt+6** | **Comando+6** |
| Volver al editor | **Esc** | **Esc** |
| Ocultar todas las ventanas de herramientas | **Control+Shift+F12** | **Comando+Shift+F12** |

Si quieres ocultar todas las barras de herramientas, ventanas de herramientas y pestañas del editor, haz clic en **View > Enter Distraction Free Mode**. Esto habilita el modo *Distraction Free Mode*. Para salir del modo Distraction Free Mode, haz clic en **View > Exit Distraction Free Mode**.

Puedes usar la *Búsqueda rápida* para buscar y aplicar filtros en la mayoría de las ventanas de herramientas en Android Studio. Para usar la búsqueda rápida, selecciona la ventana de herramientas y luego escribe el texto de tu búsqueda.

**Completar código**

Android Studio ofrece tres opciones para completar código, a las que puedes acceder usando combinaciones de teclas.

***Tabla de combinaciones de teclas para completar código.***

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo** | **Descripción** | **Windows y Linux** | **Mac** |
| Compleción básica | Muestra sugerencias básicas para variables, tipos, métodos y expresiones, entre otras. Si llamas a la compleción básica dos veces seguidas, verás más resultados. Entre otros, miembros privados y miembros estáticos sin importar. | **Control+Espacio** | **Control+Espacio** |
| Compleción inteligente | Muestra opciones relevantes en función del contexto. La compleción inteligente reconoce el tipo y los flujos de datos previstos. Si llamas a la compleción inteligente dos veces seguidas, verás más resultados. Por ejemplo, cadenas. | **Control+Shift+Espacio** | **Control+Shift+Espacio** |
| Compleción de enunciados | Completa la instrucción actual agregando elementos faltantes, como paréntesis, corchetes, llaves y formato, entre otros. | **Control+Shift+Enter** | **Shift+Comando+Enter** |

También puedes realizar correcciones rápidas y mostrar acciones de intención presionando **Alt+Enter**.

**Navegación**

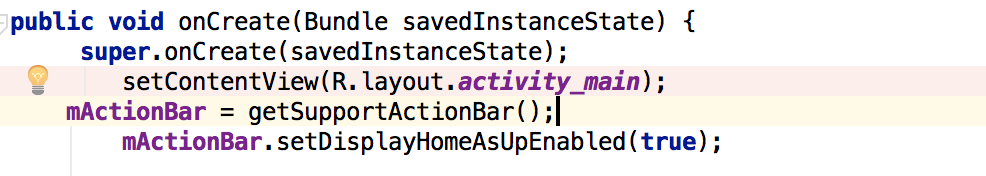
Aquí te ofrecemos algunas sugerencias para ayudarte a recorrer Android Studio.

* Recorre los archivos a los que accediste recientemente usando la acción *Recent Files* . Presiona **Control+E** (**Comando+E** en una Mac) para activar la acción Recent Files. De forma predeterminada, se selecciona el último archivo al que accediste. También puedes acceder a cualquier ventana de herramientas a través de la columna izquierda en esta acción.
* Visualiza la estructura del archivo actual con la acción *File Structure*. Activa la acción File Structure presionando **Control+F12** (**Comando+F12** en una Mac). Con esta acción, podrás navegar rápidamente hacia cualquier parte del archivo actual.
* Busca una clase específica en tu proyecto y navega hacia ella con la acción *Navigate to Class*. Activa la acción presionando **Control+N** (**Comando+O** en una Mac). Navigate to Class admite expresiones sofisticadas, como jorobas de camellos, rutas de acceso, la línea de navegar a y coincidencia de segundo nombre, entre otras. Si la llamas dos veces seguidas, te mostrará los resultados fuera de las clases de proyectos.
* Navega a un archivo o una carpeta con la acción *Navigate to File*. Activa la acción Navigate to File presionando **Control+Shift+N** (**Comando+Shift+O** en una Mac). Para buscar carpetas en lugar de archivos, agrega una “/” al final de la expresión.
* Navega hasta un método o campo por nombre con la acción *Navigate to Symbol*. Activa la acción Navigate to Symbol presionando **Control+Shift+Alt+N** (**Comando+Shift+Alt+O** en una Mac).
* Busca todas las partes de código que hagan referencia a la clase, el método, el campo, el parámetro o el enunciado en la posición actual del cursor presionando **Alt+F7**.

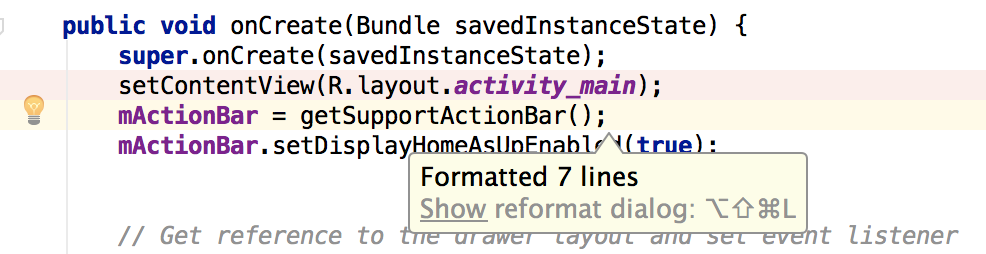
**Estilo y formato**

Mientras editas, Android Studio aplica automáticamente formatos y estilos según lo especificado en tu configuración de estilo de código. Puedes personalizar la configuración de estilo de código programando el idioma, que incluye la especificación de convenciones para pestañas y sangrías, espacios, ajuste y llaves, y líneas en blanco. Para personalizar la configuración de estilo de tu código, haz clic en **File > Settings > Editor > Code Style** (**Android Studio > Preferences > Editor > Code Style** en una Mac.)

Si bien el IDE aplica formato automáticamente mientras trabajas, también puedes llamar explícitamente a la acción *Reformat Code* presionando **Control+Alt+L** (**Opt+Comando+L** en una Mac), o aplicar sangrías automáticas a todas las líneas presionando **Control+Alt+I** (**Alt+Option+I** en una Mac).



*Código antes de la aplicación de formato.*



*Código después de la aplicación de formato.*

**Aspectos básicos del control de versión**

Android Studio admite diferentes sistemas de control de versión (VCS), incluidos Git, GitHub, CVS, Mercurial, Subversion y Google Cloud Source Repositories.

Después de importar tu app a Android Studio, usa las opciones del menú del VCS de Android Studio a fin de habilitar la compatibilidad con VCS para el sistema de control de versión deseado, crea un repositorio, importa los nuevos archivos al control de versión y realiza otras operaciones de control de versión:

1. En el menú **VCS** de Android Studio, haz clic en **Enable Version Control Integration**.
2. En el menú desplegable, selecciona un sistema de control de versión para asociarlo con la raíz del proyecto y luego haz clic en **OK**.

En el menú del VCS ahora se muestran diversas opciones de control de versión según el sistema que hayas seleccionado.

**Nota:** También puedes usar la opción del menú **File > Settings > Version Control** para configurar y modificar los ajustes de control de versión.

**Sistema de compilación de Gradle**

Android Studio usa Gradle como base del sistema de compilación, y proporciona más características específicas de Android a través del Complemento de Android para Gradle. Este sistema de compilación se ejecuta en una herramienta integrada desde el menú de Android Studio, y lo hace independientemente de la línea de comandos. Puedes usar las funciones del sistema de compilación para lo siguiente:

* personalizar, configurar y extender el proceso de compilación;
* crear varios APK para tu app con diferentes funciones usando el mismo proyecto y los mismos módulos;
* volver a usar códigos y recursos entre conjuntos de orígenes.

Recurriendo a la flexibilidad de Gradle, puedes lograr todo esto sin modificar los archivos de origen de tu app. Los archivos de compilación de Android Studio se denominan build.gradle. Son archivos de texto sin formato que usan sintaxis Groovy para configurar la compilación con elementos proporcionados por el complemento de Android para Gradle. Cada proyecto tiene un archivo de compilación de nivel superior para todo el proyecto y archivos de compilación de nivel de módulo independientes para cada módulo. Cuando importas un proyecto existente, Android Studio genera automáticamente los archivos de compilación necesarios.

**Variantes de compilación**

El sistema de compilación puede ayudarte a crear diferentes versiones de la misma aplicación a partir de un solo proyecto. Esto resulta útil cuando tienes una versión gratuita o una versión paga de tu app, o si quieres distribuir múltiples APK para diferentes configuraciones de dispositivos en Google Play.

Para obtener más información acerca de cómo configurar variantes de compilación, consulta Configuración de compilaciones de Gradle.

**Divisiones de APK**

La división de APK te permite crear de forma eficiente varios APK en función de la densidad de la pantalla o ABI. Por ejemplo, la división de APK te permite crear versiones hdpi y mdpi independientes para una app sin dejar de considerarlas como una variante individual y permitiéndoles compartir la configuración de una app de prueba, javac, dx y ProGuard.

Para obtener más información acerca de cómo usar divisiones de APK, lee Divisiones de APK.

**Reducción de recursos**

La reducción de recursos en Android Studio elimina automáticamente los recursos sin usar del paquete de tu app y de las dependencias de bibliotecas. Por ejemplo, si en tu aplicación se usan servicios de Google Play para acceder a la funcionalidad de Google Drive y actualmente no usas Google Sign-In, la reducción de recursos puede eliminar los diferentes recursos de elemento de diseño de los botones SignInButton.

**Nota:** La reducción de recursos funciona con herramientas de reducción de código, como ProGuard.

**Administración de dependencias**

Las dependencias para tu proyecto se especifican por nombre en el archivo build.gradle. Gradle se ocupa de buscar tus dependencias y hacer que estén disponibles en tu compilación. Puedes declarar dependencias de módulos, dependencias binarias remotas y dependencias binarias locales en tu archivo build.gradle. Android Studio configura los proyectos para que usen el repositorio central de Maven de forma predeterminada. (Esta configuración está incluida en el archivo de compilación de nivel superior del proyecto). Para obtener más información acerca de la configuración de dependencias, consulta Configurar variantes de compilación.

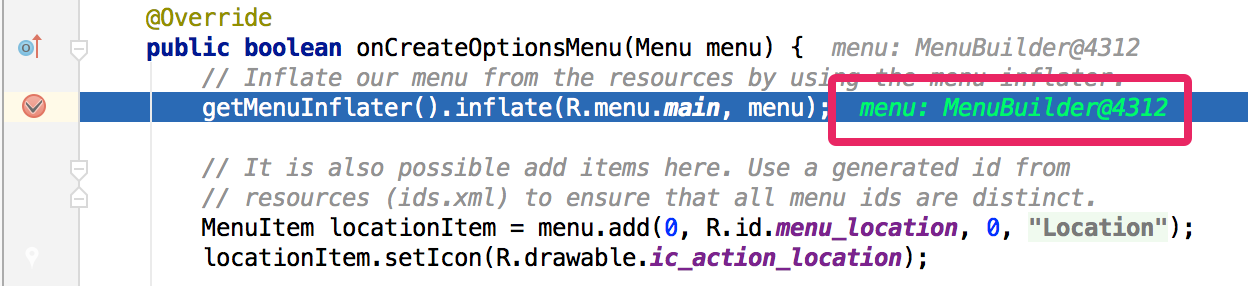
**Herramientas de depuración y perfil**

Android Studio te ayuda a depurar y mejorar el rendimiento de tu código. Esto incluye herramientas integradas de depuración y análisis de rendimiento.

**Depuración integrada**

Usa la depuración integrada para mejorar las revisiones de código en la vista del depurador con verificación integrada de referencias, expresiones y valores de variables. La información de depuración integrada incluye:

* valores de variables integradas;
* objetos que hacen referencia a un objeto seleccionado;
* valores de retorno de métodos;
* expresiones Lambda y de operador;
* valores de información sobre herramientas.



*Valor de una variable integrada.*

Para habilitar la depuración integrada, en la ventana **Debug** haz clic en **Settings** https://developer.android.com/images/tools/studio-debug-settings-icon.png?hl=es-419y selecciona la casilla de verificación **Show Values Inline**.

**Monitores de rendimiento**

Android Studio proporciona monitores de rendimiento para que puedas realizar de manera más sencilla un seguimiento del uso de CPU y memoria de tu app, buscar objetos sin asignar, localizar pérdidas de memoria, optimizar el rendimiento de los gráficos y analizar solicitudes de la red. Con tu app ejecutándose en un dispositivo o emulador, abre la ventana de herramientas **Android Monitor** y haz clic en la pestaña **Monitors**.

**Volcado de montón**

Cuando controlas el uso de la memoria en Android Studio, puedes iniciar simultáneamente la recolección de elementos no usados y volcar el montón de Java a una captura instantánea del montón en un archivo de formato binario HPROF específico de Android. El visor de HPROF muestra las clases, las instancias de cada clase y un árbol de referencia para ayudarte a realizar el seguimiento del uso de la memoria y encontrar fugas de memoria.

**Seguimiento de asignaciones**

Android Studio te permite realizar un seguimiento de la asignación de memoria mientras controla el uso de esta. El seguimiento de la asignación de memoria te permite controlar dónde se asignan los objetos cuando realizas ciertas acciones. Conocer estas asignaciones te permite optimizar el rendimiento de tu app y el uso de la memoria ajustando las llamadas del método relacionadas con las acciones en cuestión.

**Acceso a archivos de datos**

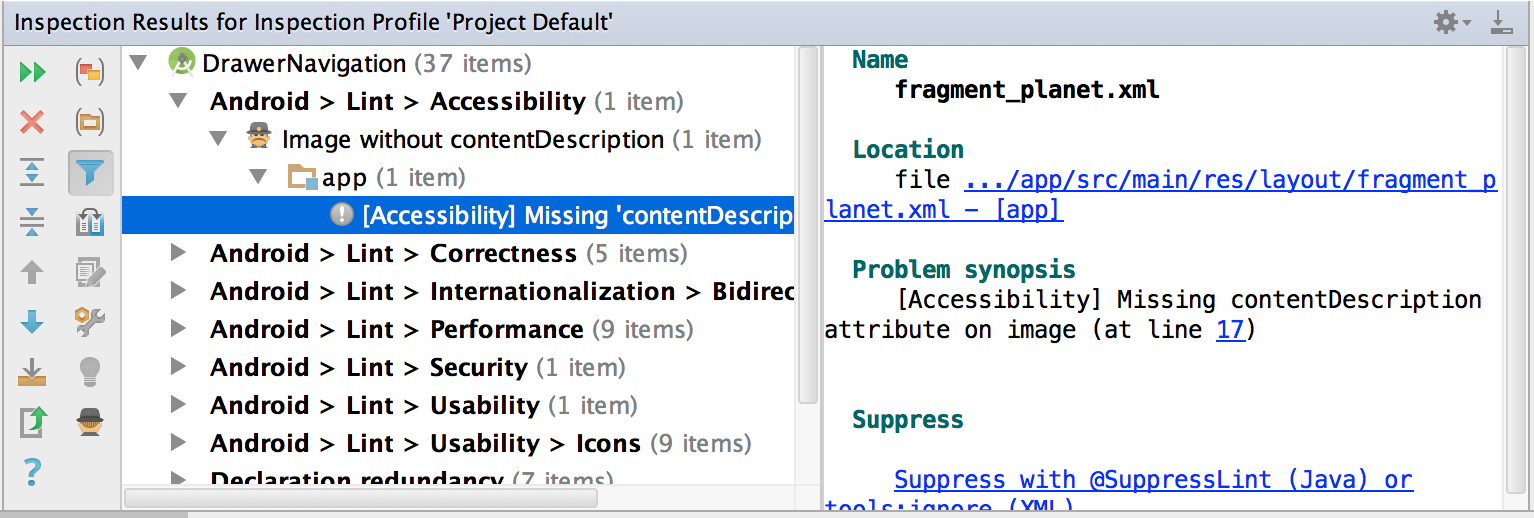
Las herramientas del Android SDK, como Systrace, logcat y Traceview, , generan datos de rendimiento y depuración para un análisis detallado de la app.

Para ver los archivos de datos generados disponibles, abre la ventana de herramientas Captures. En la lista de los archivos generados, haz doble clic en uno para ver los datos. Haz clic con el botón secundario en cualquiera de los archivos .hprof para convertirlos al formato de archivo .hprof estándar.

**Inspecciones de código**

Cuando compilas tu programa, Android Studio ejecuta automáticamente inspecciones de Lint y otras inspecciones de IDE configuradas para ayudarte a identificar y corregir fácilmente problemas en la calidad estructural de tu código.

La herramienta Lint verifica los archivos de origen de tu proyecto Android para detectar posibles errores y mejoras de optimización en relación con la corrección, la seguridad, el rendimiento, el uso, la accesibilidad y la internacionalización.



*Resultados de una inspección de Lint en Android Studio.*

Además de las verificaciones de Lint, Android Studio también realiza inspecciones de código de IntelliJ y valida anotaciones para simplificar tu flujo de trabajo de codificación.

**Anotaciones en Android Studio**

Android Studio admite anotaciones para variables, parámetros y valores de retorno para ayudarte a detectar errores, como excepciones de puntero nulo y conflictos de tipos de recurso. SDK Manager de Android empaqueta la biblioteca de compatibilidad-anotaciones en el Repositorio de compatibilidad de Android para usarla con Android Studio. Android Studio valida las anotaciones configuradas durante la inspección del código.

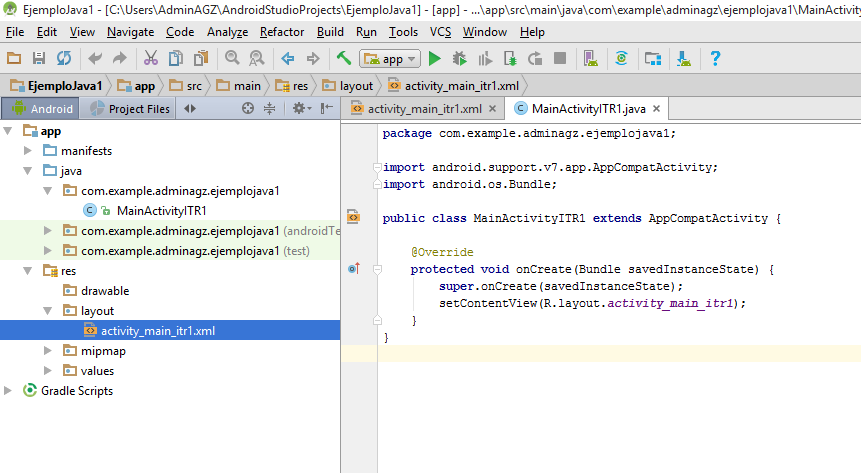
**Mensajes de registro**

Cuando compilas y ejecutas tu app con Android Studio, puedes ver mensajes adb de salida y mensajes de registro del dispositivo (logcat) haciendo clic en **Android Monitor** en la parte inferior de la ventana.

Si quieres depurar tu app con el Monitor de dispositivos Android, puedes iniciar el Monitor de dispositivos haciendo clic en **Tools > Android > Android Device Monitor**. En el Monitor de dispositivos encontrarás el conjunto completo de herramientas DDMS para perfilar tu app, controlar comportamientos del dispositivo y más. En este también se incluye la herramienta del Visor de jerarquía para ayudarte a optimizar tus diseños.

# **Programa Básico de Java con Android**

Utilizaremos el programa de Android Studio para crear un nuevo proyecto de Android llamado EjemploJava1, con una actividad vacía de nombre MainActivityITR1. Un proyecto de Android contiene un programa de java y otros datos en una serie de archivos, que son utilizados por el programa o por el sistema. Un programa de java es una colección de clases, contenidas en uno o varios archivos fi1.java, file2.java. En Android estos archivos java se almacenan en Android->app->java->com.example.adminusr.myapplication. Para nuestro proyecto se crea MainActivityITR1.java, y un layout con el nombre activity\_main\_itr1.xml ubicado en app->res->layout, a continuación se muestra la ubicación de estos archivos:



En la siguiente tabla se muestra el código del archivo: MainActivityITR1.java

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  } } |

Este es el modelo que utilizaremos como punto de partida en todos nuestros programas Java. Examinaremos su estructura:

* El programa contiene una serie de instrucciones o sentencias. En Java cada instrucción o sentencia debe terminar con un punto y coma. Se pueden escribir varias instrucciones en una línea o una instrucción en varias líneas. Los espacios en blanco son ignorados.
* La primera línea **package** com.example.adminagz.ejemplojava1; indica que esta clase pertenece al paquete *com.example*. Cada punto en el nombre de un paquete indica un subdirectorio, por lo que nuestro programa Java está contenido en el directorio:

src/com/example/ejemplojava1

* Las dos líneas siguientes precedidas por *import* indican que este programa utiliza dos clases predefinidas en dos paquetes del sistema Android.

**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;

En este caso, el programa importa la clase *Activity* del paquete *android*.*app* y la clase *Bundle* del paquete del paquete *android*.*os*. Android contiene numerosos paquetes con miles de clases. En el editor de Android Studio, cuando escribimos el nombre de una clase, ésta se subraya en rojo si se requiere añadir un *import*, por lo que no necesitamos recordar el nombre del paquete. Basta pulsar Alt-Enter, para que se agregue la importación faltante.

* El programa contiene una clase llamada EjemploJava1 cuya definición abarca el bloque de código comprendido entre la llave inicial y final:

**public class** MainActivityITR1 **extends** AppCompatActivity {  
  
 }

Nótese que la definición de la clase no termina con un punto y coma. Esto es debido a que no se trata de una sentencia o instrucción ejecutable, sino un tipo de definición. El atributo public indica que la clase es pública y puede ser utilizada externamente. La declaración finaliza con extends AppCompatActivity. Esto significa que nuestra clase es una subclase AppCompatActivity, definida en el paquete android.app y hereda todas sus propiedades, además de las que nosotros le queramos añadir. Por tanto, JavaEjemplo1 es una extensión o una generalización de la clase AppCompatActivity. Se dice que AppCompatActivity es una superclase de JavaEjemplo1.

* La siguiente línea:

// Inflate the menu; this adds items to the action bar if it is present.

es un comentario. Los comentarios pueden abarcar varias líneas y se delimitan por las parejas de caracteres.

Las dobles barras // comienzan un comentario hasta el final de la línea.

Otra manera de poner comentarios:

/\* comentario \*/

* Dentro de la definición de la clase JavaEjemplo1 encontramos la declaración de un método de la clase llamado *onCreate*.

@Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main\_itr1***);  
 }

Este método es una función con un parámetro llamado *savedInstanceState* de tipo *Bundle*. La declaración comienza con la clave *@override*, lo que indica que se está redefiniendo o sobreescribiendo el método *onCreate* de la super-clase. El tipo de acceso del método es *public*, al que se puede acceder externamente desde otro programa. El método es de tipo *void* porque esta función no devuelve ningún resultado.

* Finalmente, entre dos llaves, tenemos la definición del método, que consiste en dos instrucciones:

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

En la primera, el prefijo *super* indica ejecutar un método de la super-clase *Activity*. Por tanto, la instrucción:

**super**.onCreate(savedInstanceState);

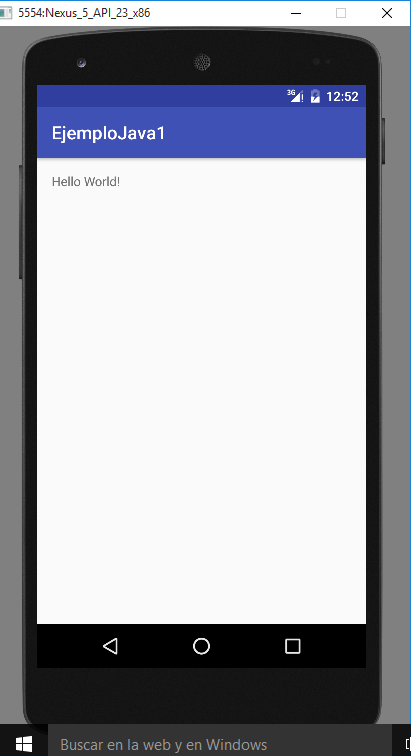
ejecuta el método *onCreate*() de *Activity*, aplicado sobre el argumento *saveInstanceState*. La ultima instrucción ejecuta el método *setContentView*() tomando como argumento la variable *R.layout*.*activity*\_*main*, que es una referencia al archivo *main*.*xml* que contiene el *layout* de la interfaz de usuario que vemos en la pantalla.

A continuación, se muestra la estructura básica de una actividad o programa de Android, que puede considerarse un **molde, o *template* para todos los programas que escribiremos a continuación**. Esta actividad se llama: activity\_main\_itr1.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_main\_itr1"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  tools:context="com.example.adminagz.ejemplojava1.MainActivityITR1"**>   <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Hello World!"  android:id="@+id/textView"** /> </**RelativeLayout**> |

Como puede observar se agregó el id al TextView.

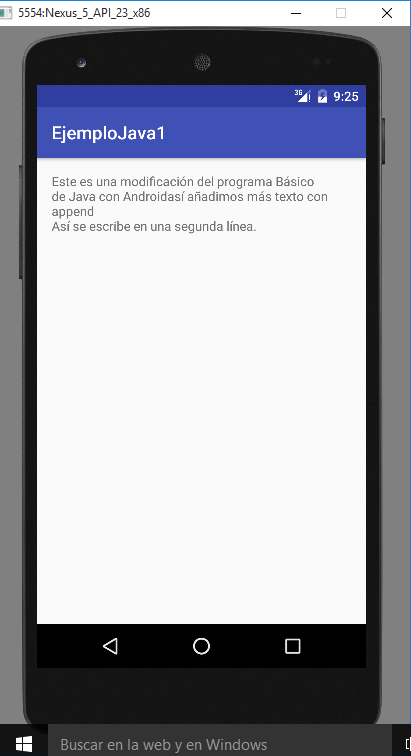
La ejecución del programa en el dispositivo virtual (a partir de este punto lo llamaremos emulador) se muestra en la siguiente ventana:



A continuación, modificaremos el programa básico para enviar texto al TextView del dispositivo como se muestra a continuación en el archivo MainActivityITR1.java:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  */\* Declaramos y asignamos la referencia del TextView a la  variable texto  \*/* TextView texto= (TextView) findViewById(R.id.***textView***);  *// Enviamos información al TextView, con el método setText* texto.setText(**"Este es una modificación del programa Básico de Java con Android"**);  texto.append(**"así añadimos más texto con append"**);  texto.append(**"\nAsí se escribe en una segunda línea."**);  } } |

El archivo **activity\_main\_itr1.xml**, no tiene cambios, nada más asegúrese que tenga la propiedad id en el TextView. Al ejecutar el proyecto se presenta la siguiente pantalla del dispositivo virtual (a partir de este punto lo llamaremos emulador):



# **Uso de Variables**

Las variables almacenan datos. Los tipos de datos primitivos son: int, float, double, char, boolean, long, short y byte.

int. Un número entero +- 2,147,483,647

float. Un número con decimales entre 3.4e-38 y 3.4e+38 doublé. Un número en doble precisión entre 1,7e-308 y 1.7e+308 char. Un carácter UNICODE. boolean. Una variable lógica true o false.

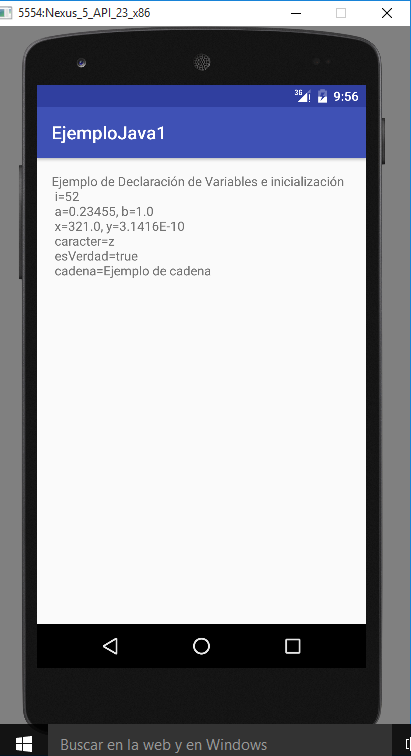
Otros tipos importantes:

string. Es una clase para almacenar cadenas de caracteres. void. Se usa para describir un método que no devuelve ningún valor.

Las variables deben declararse e inicializarse en cualquier punto del programa. Por ejemplo, modifiquemos el programa anterior para declarar e inicializar algunas variables, que escribiremos en pantalla.

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**"Ejemplo de Declaración de Variables e inicialización"**);   **int** i;  i=52;  **float** a,b=1;  a=0.23455f;  **double** x=321, y=3.1416e-10;  **char** car=**'z'**;  **boolean** esVerdad=**true**;  String cad=**"Ejemplo de cadena"**;  *// Manda las variables al TextView* texto.append(**"\n i="** + i);  texto.append(**"\n a="** + a + **", b="** + b);  texto.append(**"\n x="** + x + **", y="** + y);  texto.append(**"\n caracter="** + car);  texto.append(**"\n esVerdad="**+ esVerdad);  texto.append(**"\n cadena="** + cad);  } } |

La ejecución del programa en el emulador se muestra en la siguiente ventana:



**Conversiones de Tipos de Dato**

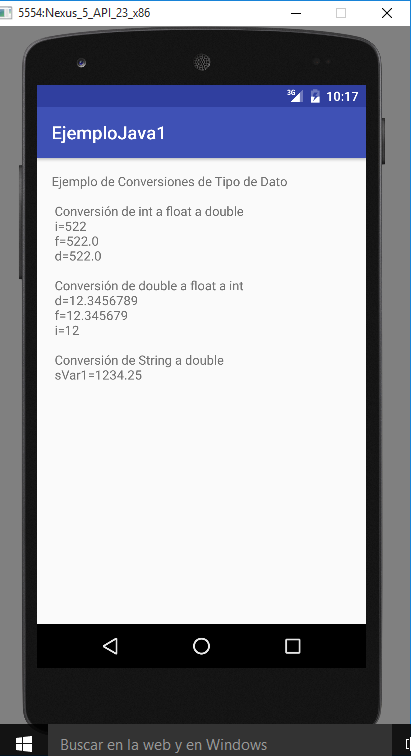
Los tipos de dato de las variables se pueden convertir a otro. Si la conversión se hace a un tipo más amplio, por ejemplo de int a float, la conversión se realiza implícitamente. Si, al contrario, la conversión se realiza a un tipo más restringido, necesitamos indicarlo explícitamente con una conversión o cast, indicando el nuevo tipo entre paréntesis pues se puede perder información. Por ejemplo:

float x = 1.5f;

int a = (int) x;

En el siguiente ejemplo convertimos un valor de int a float a double y viceversa.

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**"Ejemplo de Conversiones de Tipo de Dato"**);  String sVar1 = **"1234.25"**;  **double** dVar2;  **int** i=522;  **float** f=i;  **double** d=f;  dVar2 = Double.*valueOf*(sVar1); *// Convertimos un String a double   // Manda las variables al TextView* texto.append(**"\n\n Conversión de int a float a double"**);  texto.append(**"\n i="** + i);  texto.append(**"\n f="** + f);  texto.append(**"\n d="** + d);   d=0.0123456789e3;  f=(**float**) d;  i=(**int**) f;  *// Manda las variables al TextView* texto.append(**"\n\n Conversión de double a float a int"**);  texto.append(**"\n d="** + d);  texto.append(**"\n f="** + f);  texto.append(**"\n i="** + i);   texto.append(**"\n\n Conversión de String a double"**);  texto.append(**"\n sVar1="** + dVar2);  } } |

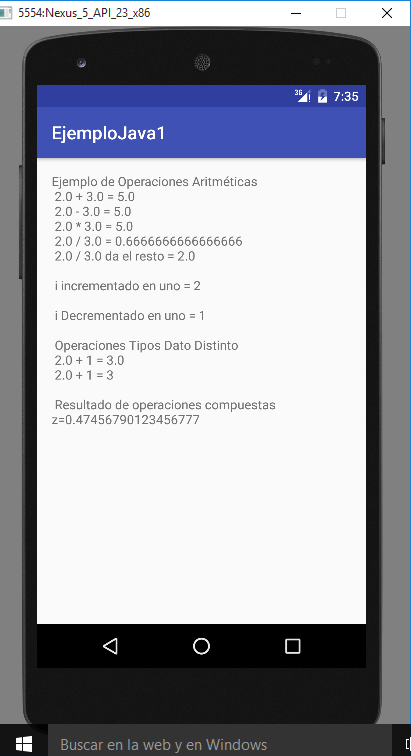


**Operaciones**

Con las variables numéricas se pueden realizar las operaciones aritméticas de suma, resta, multiplicación y división (+, -, \*, /). También está definido el modulo (%) y el incremento y decremento en una unidad (++, --). A continuación, presentamos un ejemplo de utilización de estos operadores. Se pueden sumar variables de distinto tipo y el resultado es una variable del tipo más amplio.

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**"Ejemplo de Operaciones Aritméticas"**);   **int** i = 1,j;  **double** x=2,y=3,z;  *// Suma* z = x + y;  texto.append(**"\n "** + x + **" + "** + y + **" = "** + z);  *// Resta z = x - y;* texto.append(**"\n "** + x + **" - "** + y + **" = "** + z);  *// Multiplicación z = x \* y;* texto.append(**"\n "** + x + **" \* "** + y + **" = "** + z);  *// División* z = x / y;  texto.append(**"\n "** + x + **" / "** + y + **" = "** + z);  *// Modulo x=74;* z = x % y;  texto.append(**"\n "** + x + **" / "** + y + **" da el resto = "** + z);  *// Incremento* i++;  texto.append(**"\n\n i incrementado en uno = "** + i);  *// Decremento* i--;  texto.append(**"\n\n i Decrementado en uno = "** + i);   *// Suma de tipos distintos* texto.append(**"\n\n Operaciones Tipos Dato Distinto"**);  z = x + i;  j = (**int**) x + i;  texto.append(**"\n "** + x + **" + "** + i + **" = "** + z);  texto.append(**"\n "** + x + **" + "** + i + **" = "** + j);   *//Opreaciones más complejas* z = (x \*(y + j) / (x \* x + 1) - 1 / y) \* (1 - x) / y; z = z \* z;  texto.append(**"\n\n Resultado de operaciones compuestas z="** + z);   } } |

La ejecución del programa en el emulador se muestra en la siguiente ventana:



**Funciones Matemáticas**

La clase Math contiene métodos para realizar operaciones numéricas básicas, como la exponencial, logaritmo, raíz cuadrada, y las funciones trigonométricas.

Esta clase, está ya definida en java, esto quiere decir que no hace falta importar su correspondiente librería. Podemos hacer su uso directamente, como si fueran métodos ya definidos anteriormente.

Aquí os dejo algunos ejemplos de uso de la clase Math:

Math.pow(base,exp) -> esto representa base elevado a exp. Potencias

Math.abs(valor) –>devuelve el valor absoluto de un valor

Math.cos(valor)—> devuelve el coseno del valor

Math.max(valor1,valor2) –> devuelve el máximo de 2 valores

Math.min(valor1,valor2)–>devuelve el mínmo de 2 valores

Math.sqrt(valor) –> devuelve la raiz cuadrada del valor

Para ver más funciones consulta la siguiente tabla:

|  |  |
| --- | --- |
| **Funciones Matemáticas** | |
| static double | [**abs**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#abs(double))double a)  Devuelve el valor absoluto de un valor doble. |
| static float | [**abs**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#abs(float))float a)  Devuelve el valor absoluto de un valor float. |
| static int | [**abs**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#abs(int))int a)  Devuelve el valor absoluto de un valor int. |
| static long | [**abs**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#abs(long))long a)  Devuelve el valor absoluto de un valor long. |
| static double | [**acos**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#acos(double))double a)  Devuelve el arco coseno de un ángulo. |
| static double | [**asin**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#asin(double))double a)  Devuelve el arco seno de un ángulo. |
| static double | [**atan**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#atan(double))double a)  Devuelve el arco tangente de un ángulo. |

|  |  |
| --- | --- |
| static double | [**atan2**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#atan2(double, double))double y, double x)  Convierte coordenadas rectangulares (x, y) a polares (r, theta). |
| static double | [**ceil**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#ceil(double))double a)  Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer. |
| static double | [**cos**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#cos(double))double a)  Devuelve el coseno trigonométrico de un ángulo. |
| static double | [**exp**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#exp(double))double a)  Devuelve el número de Euler e elevado a la potencia de un valor doble. |
| static double | [**floor**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#floor(double))double a)  Devuelve el valor doble más grande (el más cercano a infinito positivo) que no es mayor que el argumento y es igual a un número entero matemática. |
| static double | [**IEEEremainder**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#IEEEremainder(double, double))double f1, double f2)  Calcula el funcionamiento restante en dos argumentos, según lo establecido por la norma IEEE 754. |
| static double | [**log**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#log(double))double a)  Devuelve el logaritmo natural (base e) de un valor doble. |
| static double | [**max**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#max(double, double))double a, double b)  Devuelve el mayor de dos valores dobles. |
| static float | [**max**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#max(float, float))float a, float b)  Devuelve el mayor de dos valores float. |
| static int | [**max**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#max(int, int))int a, int b)  Devuelve el mayor de dos valores int. |
| static long | [**max**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#max(long, long))long a, long b)  Devuelve el mayor de dos valores largos. |
| static double | [**min**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#min(double, double))double a, double b)  Returns the smaller of two double values. |
| static float | [**min**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#min(float, float))float a, float b)  Devuelve el menor de dos valores dobles. |
| static int | [**min**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#min(int, int))int a, int b)  Devuelve el menor de dos valores enteros. |
| static long | [**min**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#min(long, long))long a, long b)  Devuelve el menor de dos valores enteros largos. |
| static double | [**pow**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#pow(double, double))double a, double b)  Devuelve el valor del primer argumento elevado a la potencia del segundo argumento. |
| static double | [**random**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#random()))  Devuelve un valor de tipo double con signo positivo, mayor o igual a 0.0 e inferior a 1.0. |
| static double | [**rint**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#rint(double))double a)  Devuelve el doble valor que más se aproxime al valor del argumento y es igual a un entero matemático. |
| static long | [**round**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#round(double))double a)  Devuelve el número entero largo más cercano al argumento (redondea). |
| static int | [**round**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#round(float))float a)  Devuelve el int más cercano al argumento. |
| static double | [**sin**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#sin(double))double a)  Devuelve el seno trigonométrico de un ángulo. |
| static double | [**sqrt**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#sqrt(double))double a)  Devuelve la raíz cuadrada positiva de un valor doble. |
| static double | [**tan**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#tan(double))double a)  Devuelve la tangente trigonométrica de un ángulo. |
| static double | [**toDegrees**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#toDegrees(double))double angrad)  Convierte un ángulo medido en radianes hasta un ángulo aproximadamente equivalente medido en grados. |
| static double | [**toRadians**(](http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Math.html#toRadians(double))double angdeg)  Convierte un ángulo medido en grados a un ángulo aproximadamente equivalente medido en radianes. |

En el siguiente ejemplo utilizamos algunas de las funciones más usuales:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**"Ejemplo de Funciones Matematicas"**);   **double** x,y,z; x=Math.***PI***;  texto.append(**"\n x = PI = "** + x);  y=-1;  texto.append(**"\n y = "** + y);  texto.append(**"\n Valor absoluto de y ="** + Math.*abs*(y)); z = Math.*sqrt*(x);  texto.append(**"\n Raiz Cuadrada de x="** + z); z=Math.*log*(x);  texto.append(**"\n Logaritmo de x ="** + z); z=Math.*exp*(x);  texto.append(**"\n Exponencial(x) ="** + z); z=Math.*pow*(x, 3);  texto.append(**"\n x al cubo ="** + z); z= Math.*cos*(x);  texto.append(**"\n coseno de x="** + z); z = Math.*sin*(x);  texto.append(**"\n seno de x="** + z); z = Math.*tan*(x);  texto.append(**"\n Tangente(x)="** + z); z = Math.*acos*(y);  texto.append(**"\n Arcocoseno (y)="** + z); z = Math.*asin*(y);  texto.append(**"\n Arcoseno (y)="** + z); z = Math.*atan*(y);  texto.append(**"\n Arcotangente (y)="** + z); z = Math.*max*(x,y);  texto.append(**"\n Maximo de x e y="** + z);  z = Math.*IEEEremainder*(x, y);  texto.append(**"\n El resto de x entre y es="** + z); z = Math.*toDegrees*(x);  texto.append(**"\n Expresado en grados x="** + z); z = Math.*toRadians*(y);  texto.append(**"\n Expresando en Radianes y="** + z);  } } |

La ejecución del programa en el emulador se muestra en la siguiente ventana:



# **Números Aleatorios y métodos de Redondeo**

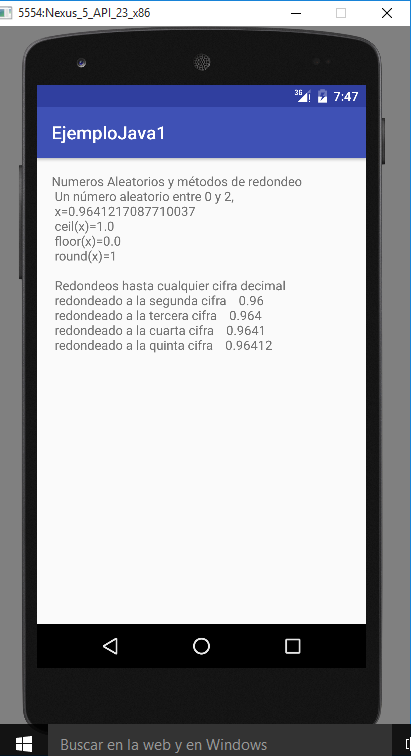
Otras funciones de interés son:

* Random() devuelve un número aleatorio entre 0 y 1.
* Los métodos de redondeo hacia arriba ceil(), hacia abajo floor(), que proporcionan números doublé con cifras decimales nulas.
* Round() redondea un float y lo transforma en el int más próximo, o redondea un doublé y lo transforma en long.

El siguiente ejemplo hace uso de estas funciones:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**"Numeros Aleatorios y métodos de redondeo"**);   **double** x,y; **int** i;  x=2\*Math.*random*();  texto.append(**"\n Un número aleatorio entre 0 y 2, \n x="** + x);   y = Math.*ceil*(x);  texto.append(**"\n ceil(x)="** + y);   y = Math.*floor*(x); texto.append(**"\n floor(x)="** + y);   i=(**int**) Math.*round*(x);  texto.append(**"\n round(x)="** + i);   texto.append(**"\n\n Redondeos hasta cualquier cifra decimal"**); y= Math.*round*(x\*1.e2)/1.e2;  texto.append(**"\n redondeado a la segunda cifra "** + y);   y=Math.*round*(x\*1.e3)/1.e3;  texto.append(**"\n redondeado a la tercera cifra "** + y);   y=Math.*round*(x\*1.e4)/1.e4;  texto.append(**"\n redondeado a la cuarta cifra "** + y);   y=Math.*round*(x\*1.e5)/1.e5;  texto.append(**"\n redondeado a la quinta cifra "** + y);  } } |

La ejecución del programa en el emulador se muestra en la siguiente ventana:



# **Bloque if-else**

Un bloque es una serie de instrucciones entre los caracteres abrir y cerrar llave

{. . .}

En Java un bloque se procesa como si se tratara de una sola instrucción o sentencia. Las variables declaradas dentro de un bloque no están definidas fuera de él.

Los bloques if-else permiten ejecutar instrucciones dependiendo de la relación entre los valores de ciertas variables. Esta relación se establece mediante el uso de los operadores de comparación, que son: igual, distinto, mayor, menor, mayor o igual, menor igual: ==, !=, >, <, >=,<=

También se pueden utilizar variables booleanas, que darán true o false si cierta relación se verifica, por ejemplo:

float x=1, y=2;

boolean condición = x < y;

Los operadores lógicos entre variables booleanas son:

AND, denotado en java: &, &&

y OR: |, ||

Por ejemplo:

boolean condicion1, condicion2, condicion3, condicion4; condicion3 = condicion1 & condicion2;

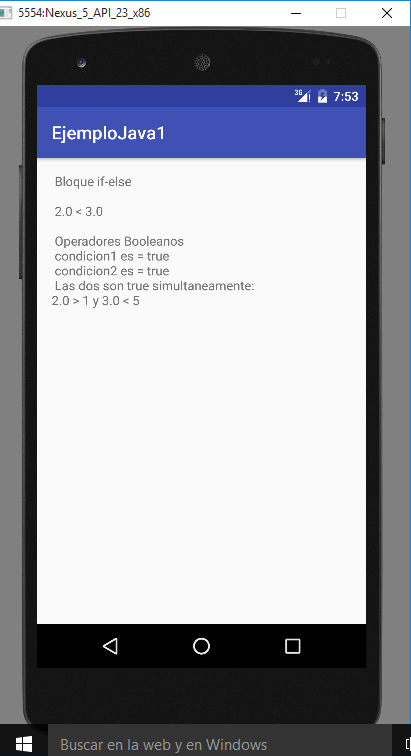
condicion4 = condicion1 | condicion2;

La diferencia entre los operadores simples ( & ) o dobles ( && ), es que el simple evalúa ambas condiciones, mientras que el doble evalúa la primera y, si es falsa no evalúa la segunda.

En el siguiente ejemplo demostramos el uso del bloque if-else y del uso de variables boolean:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**" Bloque if-else \n"**);   **double** x=2, y=3;  **if** (x==y){  texto.append(**"\n "**+ x +**" = "**+y);  } **else if**(x>y){  texto.append(**"\n "**+x+**" > "**+y);  }**else** {  texto.append(**"\n "**+x+**" < "**+y);  }  texto.append(**"\n\n Operadores Booleanos"**);   **boolean** condicion1 = x>1; **boolean** condicion2 = y<5;  texto.append(**"\n condicion1 es = "**+ condicion1); texto.append(**"\n condicion2 es = "**+ condicion2);  **if** (condicion1 & condicion2){  texto.append(**"\n Las dos son true simultaneamente: "** + **"\n"** + x + **" > 1 y "** + y + **" < 5"**);  } **else** {  texto.append(**"\n Una de las dos es falsa"**);  }  } } |

La ejecución del programa en el emulador se muestra en la siguiente ventana:



# **Ciclo for**

Un ciclo *for* permite repetir una instrucción normalmente utilizando un índice que se incrementa en cada paso. La instrucción a repetir puede ser una única sentencia o un bloque delimitado por llaves. Su estructura es la siguiente:

***for (inicialización ; condición ; incremento) {***

***// Bloque o sentencias***

***Sentencia1;***

***Sentencia2; . . .***

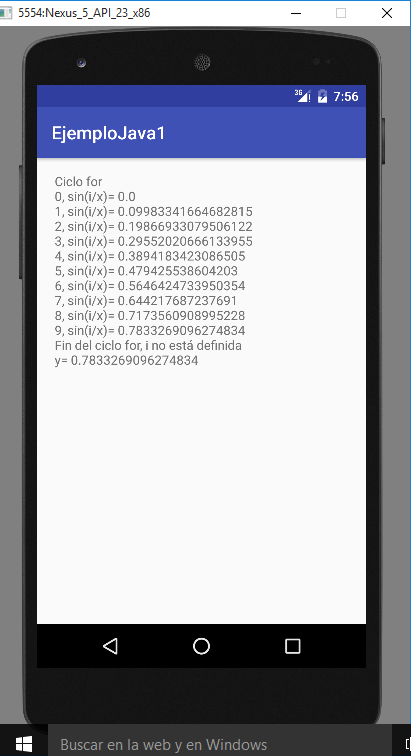
***}***

El argumento de *for*, entre paréntesis, consta de tres sentencias separadas por un punto y coma. En la primera se inicializa una o varias variables. La segunda es una condición o expresión booleana. Si la condición es verdadera, se ejecutará el bloque que viene a continuación. Finalmente se incrementa la variable. Se repite el proceso hasta que la condición sea falsa. Las variables declaradas dentro del ciclo no están definidas fuera de él.

En el siguiente ejemplo usamos un ciclo *for* para generar una tabla de valores de la función seno entre 0 y 1. Si intentamos escribir el valor de la variable i después del ciclo, el compilador de java genera un error, variable no definida.

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**" Ciclo for"**);   **double** x=10, y=0; **for** (**int** i=0; i<10; i++)  {  y=Math.*sin*(i/x);  texto.append(**"\n "** + i + **", sin(i/x)= "**+ y);  }  texto.append(**"\n Fin del ciclo for, i no está definida"**);  texto.append(**"\n y= "** + y);   } } |

La ejecución del programa en el emulador se muestra en la siguiente ventana:



# **Ciclo while**

Un ciclo *while* es similar al ciclo *for*, pero sólo requiere como argumento una condición o variable booleana. El ciclo se ejecuta repetidamente hasta que la condición sea falsa.

while ( boolean )

{

//Bloque de sentencias

Sentencia1;

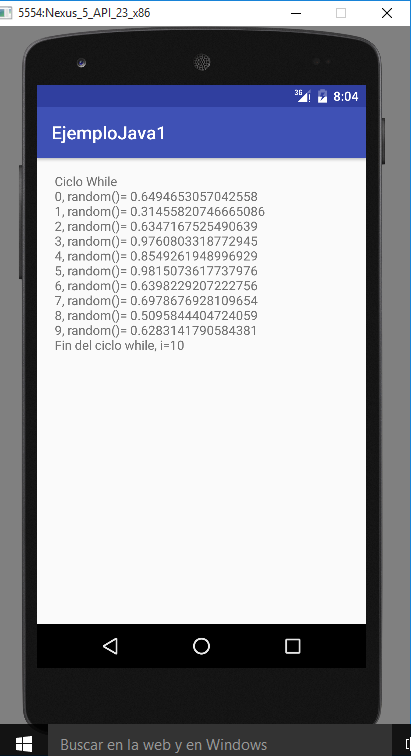
Sentencia2;

…

}

En el siguiente ejemplo generamos una tabla de 10 números aleatorios usando un ciclo while con una variable que se va incrementando hasta que toma el valor 10, y el ciclo finaliza:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**" Ciclo While"**);   **int** i=0;  **while** (i<10)  {  **double** y=Math.*random*();  texto.append(**"\n "** + i + **", random()= "**+ y);  i++;  }  texto.append(**"\n Fin del ciclo while, i="**+i);    } } |



Para mayor control del desarrollo de un ciclo, pueden utilizarse además las sentencias:

***break***. Finaliza un ciclo ***continue***. Vuelve al inicio del ciclo.

Por ejemplo en el siguiente ejercicio utilizamos *break* y *continue* para controlar la terminación de un ciclo, saltando el quinto paso:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**" Ciclo While"**);   **int** i=0;  **while** (**true**)  {  i++;  **if** (i==5) **continue**; **if** (i==10) **break**;  **double** y=Math.*sqrt*(i);  texto.append(**"\n "** + i + **", sqrt(i)= "**+ y);  }  texto.append(**"\n Fin del ciclo while, i="**+i);  } } |

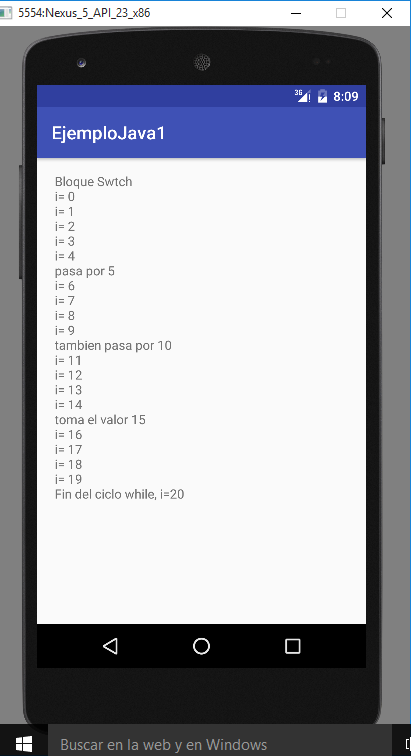


# **Bloque switch**

El bloque switch es una alternativa al bloque if-else si queremos realizar una acción dependiendo del valor de una variable, comparándola con una serie de casos. Después de cada caso hay que utilizar break para finalizar el bloque, porque todo lo que viene a continuación se ejecuta siempre sin realizar las comparaciones. He aquí un ejemplo de utilización de switch dentro de un ciclo while:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**" Bloque Swtch"**);  **int** i=0; **while** (i<20)  {  **switch**(i)  {  **case** 5:{  texto.append(**"\n pasa por 5"**);  **break**; }  **case** 10:{  texto.append(**"\n tambien pasa por 10"**);  **break**;  }  **case** 15:{  texto.append(**"\n toma el valor 15"**);  **break**;  } **default**:  texto.append(**"\n i= "**+i);  } *// fin del bloque switch* i++;  } *//Fin del ciclo while* texto.append(**"\n Fin del ciclo while, i="**+i);  } } |

La ejecución en el emulador se muestra a continuación:



# **Métodos**

Un método es un bloque de sentencias que se puede ejecutar repetidas veces invocando su nombre. Los métodos pueden tener cero, uno o varios parámetros y devolver o no un resultado. Los métodos son el análogo de las funciones o subrutinas de otros lenguajes. La declaración de un método tiene la siguiente estructura:

Acceso resultado nombre (lista de parámetros){

// Bloque del método

. . .

return valor; // si no es void

}

* ***Acceso*** indica si el método se puede ejecutar desde otra clase distinta, y puede ser ***public, private, protected***, o del tipo por defecto, si no se indica nada.

* ***Resultado*** debe ser de un tipo de dato primitivo*, int,float, doublé*, etc., o el nombre de una clase o *void*, si el método no devuelve ningún resultado.

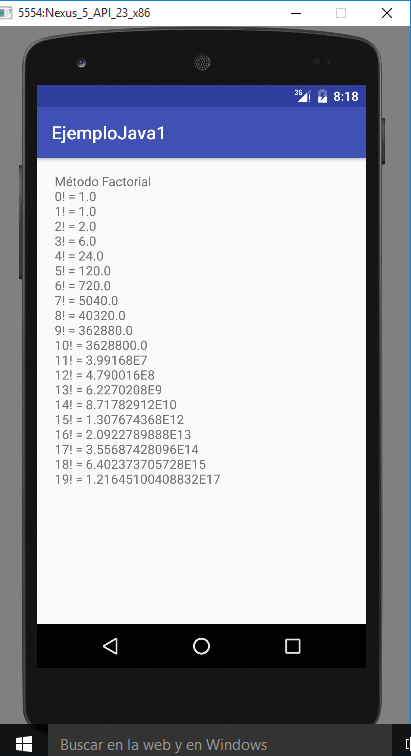
* ***Nombre*** es el nombre del método.

* La lista de parámetros, indicando tipo y nombre, separados por comas.

En el siguiente ejemplo definimos un método para calcular la factorial de un número n. Este método lo incluimos como método de la clase EjemploJava1 después del método *onCreate*, en el que hemos estado programando hasta ahora. El método factorial es llamado repetidas veces en un ciclo desde el método *onCreate*. Distintos métodos de una clase pueden llamarse entre sí. El resultado es una tabla con el factorial de los primeros 20 números.

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  texto.setText(**" Método Factorial"**);  **for** (**int** i=0; i<20; i++)  {  texto.append(**"\n "** + i + **"! = "** + factorial(i));  }  }   **double** factorial(**int** n)  {  *// Método para calcular el factorial de un número factorial (int n)* **double** fac=1;  **if** (n==0) **return** fac;  **for** (**int** i=1; i<=n;i++)  fac = fac \* i;  **return** fac;  }  } |

La ejecución en el emulador se muestra a continuación:



# **Clase String**

Un String en Java representa una cadena de caracteres no modificable. Todos los literales de la forma "*cualquier texto*", es decir, literales entre comillas dobles, que aparecen en un programa java se implementan como objetos de la clase String.

Crear un String

Se puede crear un String de varias formas, entre ellas:

* Utilizando una cadena de caracteres entre comillas:

String s1 = "abcdef";

* Utilizando operador de concatenación *+*con dos o más objetos String:

String s2 = s1 + "ghij"; //s2 contiene "abcdefghij"

String s3 = s1 + s2 + "klm"; //s3 contiene " abcdefabcdefghijklm"

Además, la clase String proporciona varios constructores, entre ellos:

**CONSTRUCTOR**

**DESCRIPCIÓN**

String()

Constructor por defecto. El nuevo

String

toma el valor

""

String s = new String(); //crea el string s vacío.

Equivale a: String s = "";

String(String s )

Crea un nuevo

String

, copiando el que recibe como parámetro.

String s = "hola";

String s1 = new String(s);

//crea el String s1 y le copia el contenido de s

String( char[] v )

Crea un String y le asigna como valor los caracteres contenidos en el

array recibido como parámetro.

char [] a = {'a', 'b', 'c', 'd', 'e'};

String s = new String(a);

//crea String s con valor "abcde"

String(char[] v, int pos, int n)

Crea un String y le asigna como valor los n caracteres contenidos en

el array recibido como parámetro, a partir de la posición pos.

char [] a = {'a', 'b', 'c', '

d', 'e'};

String s = new String(a, 1, 3);

//crea String s con valor

"bcd";

**Métodos de la Clase String**

La clase String proporciona métodos para el tratamiento de las cadenas de caracteres: acceso a caracteres individuales, buscar y extraer una subcadena, copiar cadenas, convertir cadenas a mayúsculas o minúsculas, etc.

**MÉTODO** **DESCRIPCIÓN**

length() Devuelve la longitud de la cadena

indexOf(‘caracter’) Devuelve la posición de la primera aparición de carácter lastIndexOf(‘caracter’) Devuelve la posición de la última aparición de carácter

charAt(n) Devuelve el carácter que está en la posición n

substring(n1,n2) Devuelve la subcadena comprendida entre las posiciones n1 y n2 ambas incluidas

toUpperCase() Devuelve la cadena convertida a mayúsculas toLowerCase() Devuelve la cadena convertida a minúsculas equals("cad") Compara dos cadenas y devuelve true si son iguales

equalsIgnoreCase("cad") Igual que equals pero sin considerar mayúsculas y minúsculas

compareTo(OtroString) Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.

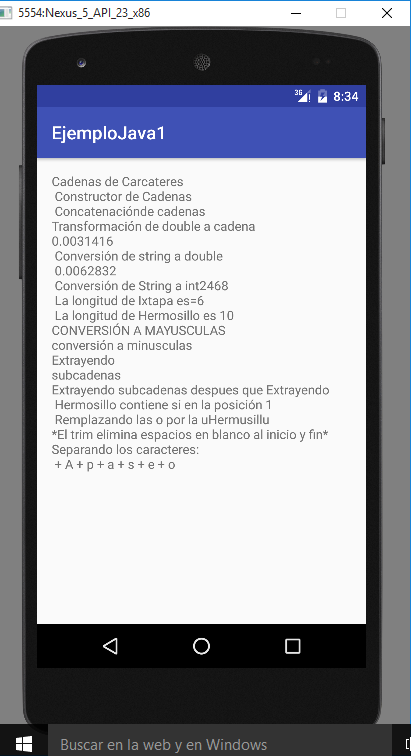
compareToIgnoreCase(OtroString) Igual que compareTo pero sin considerar mayúsculas y minúsculas.

valueOf(N) Método estático. Convierte el valor N a String. N puede ser de cualquier tipo.

En el siguiente ejemplo mostraremos demostraremos el manejo de cadenas:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto= (TextView) findViewById(R.id.***textView***);  String cadena1=**"Cadenas de Carcateres"**;  texto.setText(cadena1);   *// Inicializa la variable en el constructor* String cadena2= **new** String(**"\n Constructor de Cadenas"**);  texto.append(cadena2);   cadena1=**"\n Concatenación"**;  cadena2=**"de cadenas \n"**;  String cadena3 = cadena1 + cadena2;  texto.append(cadena3); *// Ejemplo de transformación a cadena con valueOf* **double** x=3.1416e-3;  cadena2= String.*valueOf*(x);  texto.append(**"Transformación de double a cadena \n"**);  texto.append(cadena2);   cadena1=**"3.1416e-3"**;  x= Double.*parseDouble*(cadena1);  x = x \* 2;  texto.append(**"\n Conversión de string a double \n "**+ x);   cadena1=**"1234"**;  **int** i = Integer.*parseInt*(cadena1);  i = i \*2;  texto.append(**"\n Conversión de String a int"** + i);   cadena1=**"Ixtapa"**;  **int** longitud = cadena1.length();  texto.append(**"\n La longitud de Ixtapa es="** + longitud);  longitud=**"Hermosillo"**.length();  texto.append(**"\n La longitud de Hermosillo es "**+ longitud);   cadena1=**"Conversión a mayusculas"**;  cadena1 = cadena1.toUpperCase();  texto.append(**"\n"**+ cadena1);  cadena1=**"CONVERSIÓN A MINUSCULAS"**;  cadena1 = cadena1.toLowerCase();  texto.append(**"\n"**+ cadena1);   cadena1=**"Extrayendo subcadenas"**;  cadena2 = cadena1.substring(0,10);  cadena3 = cadena1.substring(11);  texto.append(**"\n"** + cadena2);  texto.append(**"\n"** + cadena3);   *// Comparando cadenas cadena1 = "Garcia"; cadena2 = "Aura"; i = cadena1.compareTo(cadena2);* **if** (i < 1)  texto.append(**"\n"**+ cadena1 + **" antes que "**+ cadena2);  **else** texto.append(**"\n"**+ cadena1 + **" despues que "** + cadena2);   *// Busqueda de cadenas cadena1 = "Hermosillo, Son"; i = cadena1.indexOf("si");* texto.append(**"\n Hermosillo contiene si en la posición "** +1);  cadena1 = **"Hermosillo"**; cadena2 = cadena1.replace(**'o'**, **'u'**);  texto.append(**"\n Remplazando las o por la u"** + cadena2);   cadena1 = **" El trim elimina espacios en blanco al inicio y fin"**;  cadena1 = cadena1.trim();  texto.append(**"\n\*"** + cadena1 + **"\*"**);   cadena1 = **"Apaseo"**;  **char**[] caracteres = cadena1.toCharArray();  texto.append(**"\nSeparando los caracteres:\n"**);  **for** (**int** k = 0;k < caracteres.**length**; k++)  texto.append(**" + "** + caracteres[k]); }  } |

La ejecución en el emulador se muestra a continuación:



# **Dar Formato a Números**

Para dar formato a números decimales utilizaremos la clase de java *DecimalFormat*. La clase *DecimalFormat* de java nos permite mostrar los números en pantalla con el formato que queramos, es decir, con cuántos decimales, si queremos punto o coma para los decimales, etc.

import java.text.DecimalFormat;

...

DecimalFormat formateador = new DecimalFormat("####.####");

// Esto sale en pantalla con cuatro decimales, es decir, 3.4324

String cadena = formateador.format(3.43242383));

Donde "####.####" es un patrón que indica el número cifras deseado. Las características principales de los patrones se en listan a continuación.

1. El número que se desea formatear se redondea hacia arriba.
2. Si usamos ceros en vez de #, los huecos se rellenarán con ceros.

import java.text.DecimalFormat;

...

DecimalFormat formateador = new DecimalFormat("0000.0000");

// Esto sale en pantalla con cuatro cifras enteras

// y cuatro decimales, es decir, 0003.4300

String cadena = formateador.format (3.43);

Una característica curiosa, es que si usamos en la máscara el signo de porcentaje %, el número se multiplicará automáticamente por 100 al presentarlo en pantalla.

DecimalFormat formateador = new DecimalFormat("###.##%");

// Esto saca en pantalla 34.44%

String formateador.format(0.3444);

La clase DecimalFormat usa por defecto el formato para el lenguaje que tengamos instalado en el ordenador. Es decir, si nuestro sistema operativo está en español, se usará la coma para los decimales y el punto para los separadores de miles. Si estamos en inglés, se usará el punto decimal.

Una opción para cambiar esto, es crear una clase DecimalFormatSymbols, que vendrá rellena con lo del idioma por defecto, y cambiar en ella el símbolo que nos interese. Por ejemplo, si estamos en español y queremos usar el punto decimal en vez de la coma, podemos hacer esto:

import java.text.DecimalFormat; import java.text.DecimalFormatSymbols;

...

DecimalFormatSymbols simbolos = new DecimalFormatSymbols();

simbolos.setDecimalSeparator('.'); DecimalFormat formateador = new

DecimalFormat("####.####",simbolos);

// Esto sale en pantalla con punto decimal, es decir, 3.4324,

String cadena=formateador.format (3.43242383);

import java.text.DecimalFormat;

...

DecimalFormat formateador = new DecimalFormat("0000.0000");

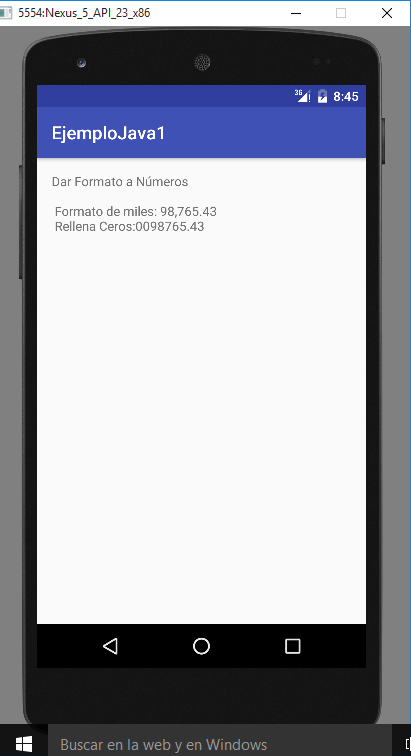
// Esto sale en pantalla con cuatro cifras enteras

// y cuatro decimales, es decir, 0003.4300

String cadena = formateador.format (3.43);

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.icu.text.DecimalFormat; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto = (TextView) findViewById(R.id.***textView***);  String cadena1=**"Dar Formato a Números"**;  texto.setText(cadena1);  **double** a;  DecimalFormat df = **new** DecimalFormat(**",###.##"**);  a=98765.4321;  texto.append(**"\n\n Formato de miles: "** + df.format(a));  df=**new** DecimalFormat(**"0000000.00"**);  texto.append(**"\n Rellena Ceros:"** + df.format(a));  }  } |

La ejecución en el emulador se muestra a continuación:



# **Manejo de Errores usando Excepciones**

Existe una regla de oro en el mundo de la programación: **en los programas ocurren errores**. Esto es sabido, pero ¿Qué sucede realmente después de que ha ocurrido el error?

¿Cómo se maneja el error? ¿Quien lo maneja?, ¿Puede recuperarlo el programa?.

El lenguaje java utiliza excepciones para proporcionar capacidades en el manejo de errores. El término excepción es una forma corta de la expresión “suceso excepcional” y puede definirse como:

“Una excepción es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias”

La captura de errores se realiza incluyendo el código que puede potencialmente contener un error (o provocar una excepción) entre las llaves de la sentencia try, como en el siguiente código:

**try**

{

// Se ejecuta algo que puede producir una excepción

}

**catch** (Exception e)

{

// Manejo de una excepción

}

**finally**

{

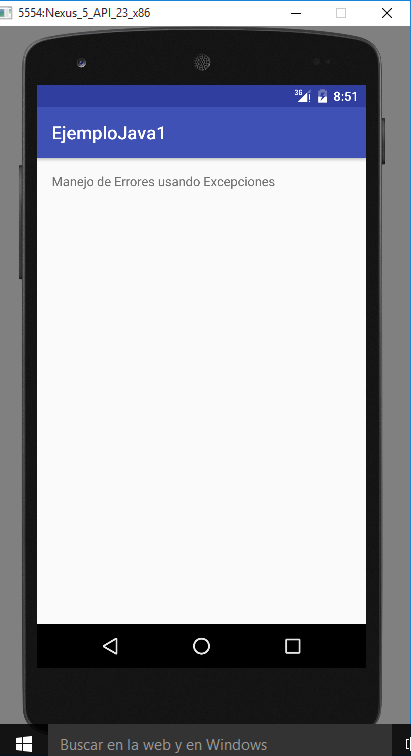
// Código a ejecutar haya o no excepción

}

En el siguiente ejemplo demostramos el uso de try para atrapar un error de división por cero:

|  |
| --- |
| **package** com.example.adminagz.ejemplojava1;  **import** android.icu.text.DecimalFormat; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.widget.TextView;  **public class** MainActivityITR1 **extends** AppCompatActivity {   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_itr1***);  TextView texto = (TextView) findViewById(R.id.***textView***);  String cadena1=**"Manejo de Errores usando Excepciones"**;  texto.setText(cadena1);  **int** r,i;  **try** {  *// Se ejecuta algo que puede producir una excepción* i=57;  r = i/0;  texto.append(**"\n El resultado es:"**+ r);  }  **catch** (Exception e)  {  *// Manejo de una excepción texto.append("\n Ocurrio error:"+ e);* }  **finally** {  *// Código a ejecutar haya o no excepción texto.append("\n\n Este código se ejecuta haya o no excepción");* }  } } |

La ejecución en el emulador se muestra a continuación:



# Diseño de Interfaces de Usuario

El diseño de la interfaz de usuario es de vital importancia en el desarrollo de una aplicación. Lo atractivo de una interfaz de usuario puede ser uno de los factores que conduzca al éxito o al fracaso de todo el proyecto.

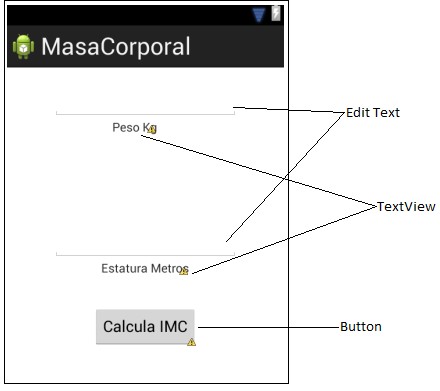
Si has realizado alguna aplicación utilizando otras plataformas, advertirás que el diseño de la interfaz de usuario en Android sigue una filosofía muy diferente. En Android la interfaz de usuario no se diseña en código, sino utilizando un lenguaje de marcado del tipo xml.

# **Masa Corporal**

Hacer un programa en android que calcule la masa corporal de una persona, proporcionando los datos de altura y peso. El programa deberá dar el resultado en base a la siguiente tabla:

|  |  |
| --- | --- |
| **IMC** | **Descripción** |
| < 25 | Tu IMC es el Adecuado |
| >=25 y < 30 | Tu IMC es de sobre peso |
| >=30 y <= 40 | Tu IMC es de obesidad |

La interfaz de la aplicación debe quedar como a continuación se muestra:

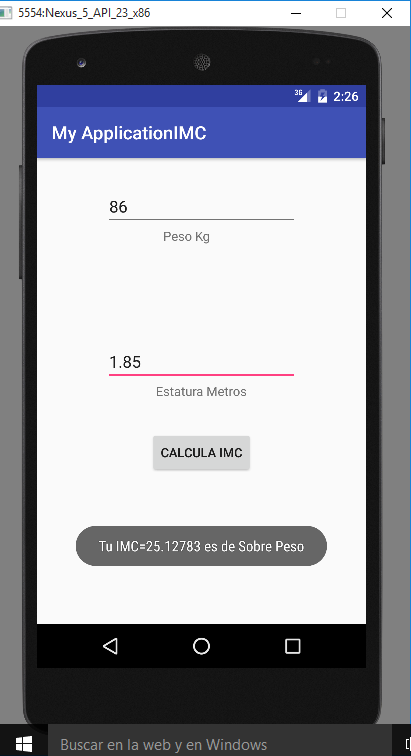


El código xml de esta interfaz se muestra a continuación:

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_main\_imc"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  tools:context="com.example.adminagz.myapplicationimc.MainActivityIMC"**>   <**TextView  android:id="@+id/textView2"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentLeft="true"  android:layout\_alignParentTop="true"  android:layout\_marginLeft="122dp"  android:layout\_marginTop="60dp"  android:text="Peso Kg"** />  <**EditText  android:id="@+id/peso"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignBottom="@+id/textView2"  android:layout\_centerHorizontal="true"  android:layout\_marginBottom="19dp"  android:ems="10"  android:inputType="numberSigned|numberDecimal"** />  <**TextView  android:id="@+id/textView3"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_centerHorizontal="true"  android:layout\_centerVertical="true"  android:text="Estatura Metros"** />  <**EditText  android:id="@+id/estatura"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignBottom="@+id/textView3"  android:layout\_centerHorizontal="true"  android:layout\_marginBottom="19dp"  android:ems="10"  android:inputType="numberSigned|numberDecimal"**/>  <**Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@+id/textView3"  android:layout\_centerHorizontal="true"  android:layout\_marginTop="34dp"  android:text="Calcula IMC"  android:onClick="CalcularIMC"** /> </**RelativeLayout**> |

El código de java se muestra a continuación:

|  |
| --- |
| **package** com.example.adminagz.myapplicationimc;  **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.view.View; **import** android.widget.EditText; **import** android.widget.Toast;  **public class** MainActivityIMC **extends** AppCompatActivity {  **private** EditText **peso**,**altura**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_imc***);  **peso**=(EditText) findViewById(R.id.***peso***);  **altura**=(EditText) findViewById(R.id.***estatura***);  }   **public void** CalcularIMC(View v)  {  **if**((**peso**.getText().length()==0)|(**altura**.getText().length()==0)){  Toast.*makeText*(**this**, **"Faltan valores"**,Toast.***LENGTH\_LONG***).show();  **return**;  }  **float** pes=Float.*parseFloat*(**peso**.getText().toString());  **float** alt=Float.*parseFloat*(**altura**.getText().toString());  **float** imc;  imc = pes/(alt \* alt);  **if**(imc<25)  Toast.*makeText*(**this**, **"Tu IMC="** + imc + **" es el Adecuado"** ,Toast.***LENGTH\_LONG***).show();  **if**(imc>=25&imc<=29.9)  Toast.*makeText*(**this**, **"Tu IMC="** + imc + **" es de Sobre Peso"** ,Toast.***LENGTH\_LONG***).show();  **if**(imc>=30&imc<=40)  Toast.*makeText*(**this**, **"Tu IMC="** + imc + **" Tienes Obesidad"** ,Toast.***LENGTH\_LONG***).show();  } }  - |



**Layouts**

Si queremos combinar varios elementos de tipo vista tendremos que utiliza un objeto de tipo Layout. Un Layout es un contenedor de una o más vistas y controla su comportamiento y posición. Hay que destacar que un Layout puede contener a otro Layout y que es un descendiente de la clase Víew.

La siguiente lista describe los Layout más utilizados en Android:

**LinearLayout**: Dispone los elementos en una fila o en una columna.

**TableLayout**: Distribuye los elementos de forma tabular.

**RelativeLayout**: Dispone los elementos en relación a otro o al padre.

**AbsoluteLayout**: Posiciona los elementos de forma absoluta.

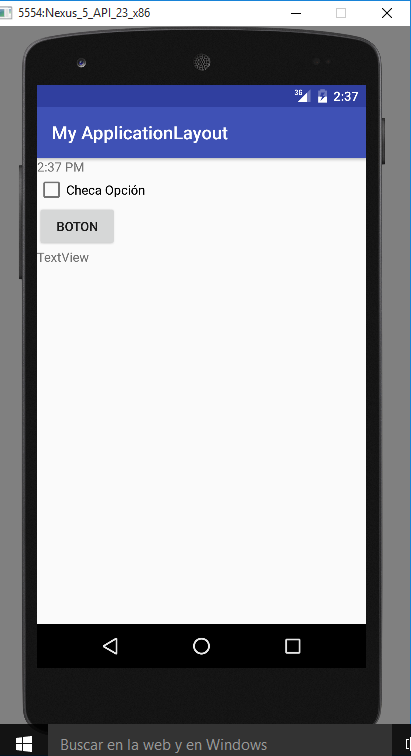
**FrameLayout**: Permite el cambio dinámico de los elementos que contiene.

A continuación demostraremos cada uno de los Layouts:

**LinearLayout** es el Layout más utilizado en la práctica. Distribuye los elementos uno detrás de otro, de forma horizontal o vertical. Coloque este código en el activity\_main.xml de un proyecto Android:

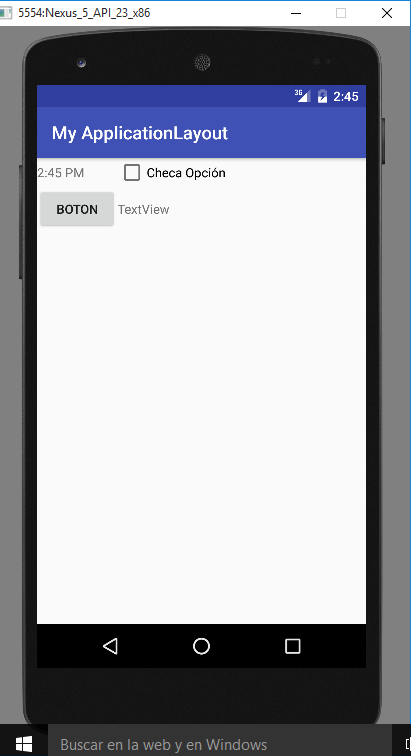
|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView"  android:layout\_alignTop="@+id/textView"  android:orientation="vertical"  tools:context=".MainActivityLayout"** >   <**DigitalClock  android:id="@+id/digitalClock1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="DigitalClock"** />   <**CheckBox  android:id="@+id/checkBox1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Checa Opción"** />   <**Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Boton"** />   <**TextView  android:id="@+id/TextView1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="TextView"** />  </**LinearLayout**> |

La ejecución en el emulador se muestra a continuación:



**TableLayout** distribuye los elementos de forma tabular. Se utiliza la etiqueta TableRow cada vez que queremos insertar una nueva línea.

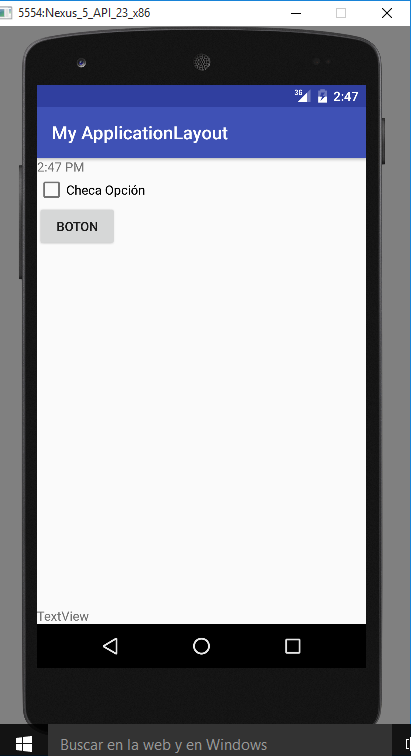
|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**TableLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView"  android:layout\_alignTop="@+id/textView"  android:orientation="vertical"  tools:context=".MainActivityLayout"**>   <**TableRow**>   <**DigitalClock  android:id="@+id/digitalClock1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="DigitalClock"** />   <**CheckBox  android:id="@+id/checkBox1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Checa Opción"** />  </**TableRow**>   <**TableRow**>   <**Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Boton"** />   <**TextView  android:id="@+id/TextView1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="TextView"** />  </**TableRow**>  </**TableLayout**> |



**RelativeLayout** permite comenzar a situar los elementos en cualquiera de los cuatro lados del contenedor e ir añadiendo nuevos elementos pegados a estos.

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView"  android:layout\_alignTop="@+id/textView"  android:orientation="vertical"  tools:context=".MainActivityLayout"**>   <**DigitalClock  android:id="@+id/digitalClock1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentTop="true"** />   <**CheckBox  android:id="@+id/checkBox1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@+id/digitalClock1"  android:text="Checa Opción"** />   <**Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@+id/checkBox1"  android:text="Boton"** />   <**TextView  android:id="@+id/TextView1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentBottom="true"  android:text="TextView"** />  </**RelativeLayout**> |

La ejecución en el emulador se muestra a continuación:



**AbsoluteLayout** permite indicar las coordenadas (x,y) donde queremos que se visualice cada elemento.

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView"  android:layout\_alignTop="@+id/textView"  android:orientation="vertical"  tools:context=".MainActivityLayout"**>   <**AnalogClock  android:id="@+id/digitalClock1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_x="50px"  android:layout\_y="50px"** />   <**CheckBox  android:id="@+id/checkBox1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_x="150px"  android:layout\_y="50px"  android:text="Checa Opción"** />   <**Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_x="50px"  android:layout\_y="250px"  android:text="Boton"** />   <**TextView  android:id="@+id/TextView1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_x="150px"  android:layout\_y="200px"  android:text="TextView"** />  </**AbsoluteLayout**> |

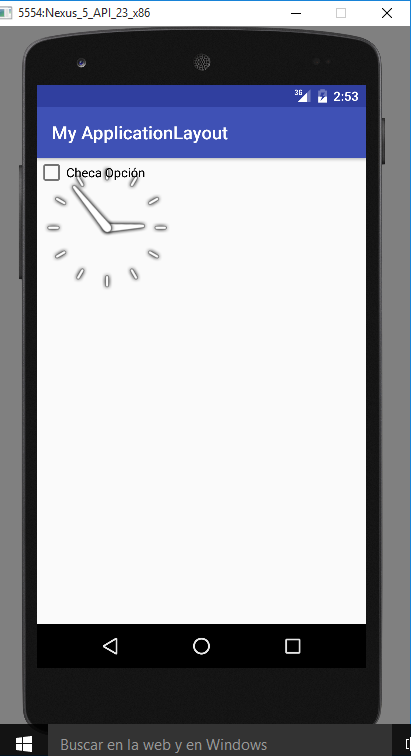
La ejecución en el emulador se muestra a continuación, como se puede observar cambiamos a un reloj analógico:



**FrameLayout** posiciona todos los elementos usando todo el contenedor, sin distribuirlos espacialmente. Este layout suele utilizarse cuando queremos que varios elementos ocupen un mismo lugar, pero solo uno será visible. Para modificar la visibilidad utilizaremos la propiedad visibility.

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView"  android:layout\_alignTop="@+id/textView"  android:orientation="vertical"  tools:context=".MainActivityLayout"**>   <**AnalogClock  android:id="@+id/digitalClock1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"** />   <**CheckBox  android:id="@+id/checkBox1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Checa Opción"** />   <**Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Boton"  android:visibility="invisible"** />   <**TextView  android:id="@+id/TextView1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="TextView"  android:visibility="invisible"** />  </**FrameLayout**> |

La ejecución en el emulador se muestra a continuación:



En algunas ocasiones, como en el AbsoluteLayout, tendremos que indicar las coordenadas donde ha de situarse un elemento. Dado que nuestra aplicación podrá ejecutarse en gran variedad de dispositivos con resoluciones muy diversas, Android nos permite indicar estas coordenadas de varias formas. En la siguiente tabla se muestran las diferentes posibilidades:

* **px** (píxeles): Estas dimensiones representan los píxeles en la pantalla.
* **mm** (milímetros): Distancia real medida sobre la pantalla.
* **in** (pulgadas): Distancia real medida sobre la pantalla.
* **pt** (puntos): Equivale a 1172 pulgadas.
* **dp** o **dip** (pixeles independientes de la densidad): Presupone un dispositivo de 160 píxeles por pulgada. Si luego el dispositivo tiene otra densidad, se realizará la correspondiente regla de tres.
* **sp** (píxeles escalados): Similar a dp pero también se escala en función del tamaño de fuente que el usuario ha escogido en las preferencias.

**Bases de Datos SQLite**

Las bases de datos son una herramienta de gran potencia en la creación de aplicaciones informáticas. Android incorpora la librería SQLite que nos permite utilizar bases de datos mediante el lenguaje SQL, de una forma sencilla y utilizando muy pocos recursos del sistema.

Para manipular una base de datos en Android usaremos la clase SQLiteOpenHelper que nos facilita la creación de bases de datos. Para crear un descendiente de esta clase hay que implementar los métodos onCreate() y onUpgrade() y opcionalmente onOpen(). La gran ventaja de utilizar esta clase es que ella se encargará de abrir la base de datos si existe o crearla si no existe. Incluso de actualizar la versión si decidimos crear una nueva estructura de la base de datos. Además esta clase tiene dos métodos getRedableDatabase() y getWritableDatabase() que abren la base de datos en modo solo lectura o lectura y escritura.

A diferencia de otros gestores de bases de datos como Oracle, **SQL Server** y **MySQL**, SQLite tiene las siguientes ventajas:

* **No requiere el soporte de un servidor**: SQLite no ejecuta un proceso para administrar la información, si no que implementa un conjunto de librerías encargadas de la gestión.
* **No necesita configuración**: Libera al programador de todo tipo de configuraciones de puertos, tamaños, ubicaciones, etc.
* **Usa un archivo para el esquema**: Crea un archivo para el esquema completo de una base de datos, lo que permite ahorrarse preocupaciones de seguridad, ya que los datos de las aplicaciones Android no pueden ser accedidos por contextos externos.
* **Es de Código Abierto**: Está disponible al dominio público de los desarrolladores al igual que sus archivos de compilación e instrucciones de escalabilidad.

Es por eso que **SQLite** es una tecnología cómoda para los dispositivos móviles. Su simplicidad, rapidez y usabilidad permiten un desarrollo muy amigable.

A continuación, crearemos el ejemplo del Abonero, el cual contempla un control de los adeudos de los clientes, con una consulta del Estado de Cuenta.

**Abonero App** es un pequeño ejemplo que demuestra cómo crear una aplicación Android con bases de datos relacionales.

Su función es servir como plataforma para el control de Abonos a la deuda de los Clientes.

El archivo xml de la interfaz **MainActivitySQLite** de nuestra aplicación se muestra a continuación:

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_main\_abonero"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  tools:context="com.example.adminagz.myapplicationabonero.MainActivityAbonero"**>   <**EditText  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:inputType="number"  android:ems="10"  android:layout\_alignParentTop="true"  android:layout\_alignParentEnd="true"  android:layout\_marginEnd="56dp"  android:layout\_marginTop="33dp"  android:id="@+id/etNoCte"  android:hint="Numero del Cliente"** />   <**EditText  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:inputType="textPersonName"  android:ems="10"  android:id="@+id/etNomCte"  android:layout\_below="@+id/etNoCte"  android:layout\_alignStart="@+id/etNoCte"  android:hint="Nombre del Cliente"** />   <**EditText  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:inputType="textPersonName"  android:ems="10"  android:layout\_below="@+id/etNomCte"  android:layout\_alignStart="@+id/etNomCte"  android:layout\_marginTop="34dp"  android:id="@+id/etDomCte"  android:hint="Domicilio del Cliente"** />   <**EditText  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:inputType="numberDecimal"  android:ems="10"  android:layout\_below="@+id/etDomCte"  android:layout\_alignEnd="@+id/etDomCte"  android:layout\_marginTop="33dp"  android:id="@+id/etSaldo"  android:hint="Saldo"  android:text="0"** />   <**EditText  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:inputType="textPersonName"  android:ems="10"  android:layout\_below="@+id/etSaldo"  android:layout\_alignEnd="@+id/etSaldo"  android:layout\_marginTop="14dp"  android:id="@+id/etCoordenadas"  android:hint="Coordenadas"** />   <**Button  android:text="Agregar"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@+id/etCoordenadas"  android:layout\_alignParentStart="true"  android:layout\_marginTop="32dp"  android:id="@+id/btnAgregar"  android:onClick="Agrega"** />   <**Button  android:text="Borrar"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignTop="@+id/btnAgregar"  android:layout\_centerHorizontal="true"  android:id="@+id/btnBorrar"  android:onClick="Borrar"** />   <**Button  android:text="Altualiza"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignBottom="@+id/btnBorrar"  android:layout\_alignParentEnd="true"  android:id="@+id/btnActualiza"  android:onClick="Actualizar"** />   <**Button  android:text="..."  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignTop="@+id/etNoCte"  android:layout\_toEndOf="@+id/etNomCte"  android:id="@+id/btnBuscar"  android:onClick="Buscar"** />   <**Button  android:text="Edo Cta"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentBottom="true"  android:layout\_alignStart="@+id/btnBorrar"  android:id="@+id/btnEdo"  android:onClick="EdoCta"** />  </**RelativeLayout**> |

La interfaz de la actividad ***MainActivityCte*** queda como a continuación se muestra:



A continuación, se agregamos la clase de java: AdminBD, esta clase es la que se va a encargar de gestionar la base de datos de SQLite, en esta vamos a crear las tablas de Cliente y Abono:

|  |
| --- |
| **package** com.example.adminagz.myapplicationabonero;  **import** android.content.Context; **import** android.database.Cursor; **import** android.database.sqlite.SQLiteDatabase; **import** android.database.sqlite.SQLiteOpenHelper;  */\*\*  \* Created by AdminAGZ on 07/02/2017.  \*/* **public class** AdminBD **extends** SQLiteOpenHelper {  **public static final** String ***DATABASE*** = **"Abonero"**;  *// Constructor de la Clase* **public** AdminBD(Context context){  **super**(context, ***DATABASE***, **null**,2);  }   @Override  **public void** onCreate(SQLiteDatabase db) {  db.execSQL(**"Create Table Cliente(NoCliente text primary key, "** +  **" NomCliente text,"** +  **" DomCliente text,"** +  **" Saldo float,"** +  **" Coordenadas text)"**);  db.execSQL(**"create table Abono(\_id integer primary key,"** +  **" fecha text, "** +  **" importe Float, "** +  **" NoCliente text)"**);  }   @Override  **public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {   }  *// Rutina generica para mandar ejecutar un Insert, Update O Delete a la base de datos* **public int** Ejecuta(String sentencia){  **try** {  SQLiteDatabase db = **this**.getWritableDatabase();  db.execSQL(sentencia);  db.close();  **return** 1;  }  **catch** (Exception ex){  **return** 0;  }  }  *// Rutina generica para mandar ejecutar una consulta a la base de datos* **public** Cursor Consulta(String select){  **try**{  SQLiteDatabase db = **this**.getReadableDatabase();  Cursor cur = db.rawQuery(select,**null**);  **return** cur;  }  **catch** (Exception ex){  Cursor cur = **null**;  **return** cur;  }  }  } |

A continuación, se muestra el código de la clase MainActivityCte.java, que es la implementación de los eventos de los botones de nuestra interfaz:

|  |
| --- |
| **package** com.example.adminagz.myapplicationabonero;  **import** android.content.Intent; **import** android.database.Cursor; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.util.Log; **import** android.view.View; **import** android.widget.EditText; **import** android.widget.TextView; **import** android.widget.Toast;  **public class** MainActivityAbonero **extends** AppCompatActivity {  **private** EditText **NoCte**,**NomCte**,**DomCte**,**SalCte**,**CoorCte**;  String **nocliente**,**nomcliente**,**domcliente**,**saldo**,**coordenadas**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_abonero***);  **NoCte** = (EditText) findViewById(R.id.***etNoCte***);  **NomCte**= (EditText) findViewById(R.id.***etNomCte***);  **DomCte** = (EditText) findViewById(R.id.***etDomCte***);  **SalCte** = (EditText) findViewById(R.id.***etSaldo***);  **CoorCte** = (EditText) findViewById(R.id.***etCoordenadas***);  }   **private void** LeerCajasTexto(){  **nocliente** = **NoCte**.getText().toString();  **nomcliente** = **NomCte**.getText().toString();  **domcliente** = **DomCte**.getText().toString();  **saldo** = **SalCte**.getText().toString();  **coordenadas** = **CoorCte**.getText().toString();  }   **private void** LimpiarCajasdeTexto(){  **NoCte**.setText(**""**);  **NomCte**.setText(**""**);  **DomCte**.setText(**""**);  **SalCte**.setText(**"0"**);  **CoorCte**.setText(**""**);  **NoCte**.requestFocus();  }   **public void** Agrega(View v){  AdminBD Admin = **new** AdminBD(**this**);  LeerCajasTexto();  String sentencia = **"Insert into Cliente(NoCliente,NomCliente,DomCliente,Saldo,Coordenadas)"** +  **" values('"** + **nocliente** + **"','"** + **nomcliente** + **"','"** + **domcliente** + **"',"** + **saldo** + **",'"** + **coordenadas** +**"')"**;  **if** (Admin.Ejecuta(sentencia) == 1){  LimpiarCajasdeTexto();  Toast.*makeText*(**this**,**"Cliente Agregado "**,Toast.***LENGTH\_LONG***).show();  String Sentencia1 = **"Insert into Abono(fecha,importe,NoCliente) "** +  **"values('10/02/2017',2500.00,'"** + **nocliente** + **"')"**;  Admin.Ejecuta(Sentencia1);  String Sentencia2 = **"Insert into Abono(fecha,importe,NoCliente)"** +  **" values('12/02/2017',1000.00,'"** + **nocliente** + **"')"**;  Admin.Ejecuta(Sentencia2);  }  **else**{  Toast.*makeText*(**this**,**"Error el Cliente no se Agrego "**,Toast.***LENGTH\_LONG***).show();  **NoCte**.requestFocus();  }  }   **public void** Actualizar(View v){  AdminBD Admin = **new** AdminBD(**this**);  LeerCajasTexto();  String sentencia = **"Update Cliente Set NomCliente='"** + **nomcliente** +**"',DomCliente='"**+ **domcliente**+**"',Saldo="**+ **saldo** +**",Coordenadas='"**+ **coordenadas** +**"' where NoCliente="** + **nocliente**;  **if** (Admin.Ejecuta(sentencia) == 1){  LimpiarCajasdeTexto();  Toast.*makeText*(**this**,**"Cliente Actualizado "** + sentencia,Toast.***LENGTH\_LONG***).show();  }  **else**{  Toast.*makeText*(**this**,**"Error no se guadaron los cambios del Cliente"**,Toast.***LENGTH\_LONG***).show();  **NoCte**.requestFocus();  }  }   **public void** Borrar(View v){  AdminBD Admin = **new** AdminBD(**this**);  LeerCajasTexto();  String sentencia = **"Delete From Cliente where NoCliente="** + **nocliente**;  **if** (Admin.Ejecuta(sentencia) == 1){  LimpiarCajasdeTexto();  Toast.*makeText*(**this**,**"Cliente Eliminado "** + sentencia,Toast.***LENGTH\_LONG***).show();  }  **else**{  Toast.*makeText*(**this**,**"Error el Cliente no se puedo borrar "** + sentencia,Toast.***LENGTH\_LONG***).show();  **NoCte**.requestFocus();  }  }   **public void** Buscar(View v){  String ncliente=**NoCte**.getText().toString();  AdminBD admin=**new** AdminBD(**this**);  Cursor fila = admin.Consulta(**"Select NomCliente, DomCliente,Saldo,Coordenadas from Cliente where NoCliente="** + ncliente);  Log.*d*(**"Error dentro de Buscar"**, **"Antes del if"**);  **if** (fila.moveToFirst()){  Log.*d*(**"Dentro de Buscar2"**, **"Pasomos el if"**);  **NomCte**.setText(fila.getString(0));  **DomCte**.setText(fila.getString(1));  **SalCte**.setText(fila.getString(2));  **CoorCte**.setText(fila.getString(3));  }  **else** Toast.*makeText*(**this**, **"No existe el Cliente"**, Toast.***LENGTH\_LONG***).show();  }   **public void** EdoCta(View v){  Intent in = **new** Intent(MainActivityAbonero.**this**,MainActivityLV.**class**);  LeerCajasTexto();  in.putExtra(**"nomb"**,**nomcliente**);  in.putExtra(**"idc"**,**nocliente**);  startActivity(in);  } } |

Como se puede observar agregamos el método de nombre EdoCta a la clase. Este método se encarga de comunicar los ***Activity***.

A continuación, se explica la funcionalidad del método:

Intent in = **new** Intent(MainActivityAbonero.**this**,MainActivityLV.**class**);

Los intent sirven para comunicar actividades, enviar mensajes es decir “intentar” realizar acciones.

Se crea un objeto llamado in de tipo Intent, entre los paréntesis el primer elemento es el nombre de la clase en la que nos encontramos, el segundo parámetro es la clase a la que se desea llamar.

Los variables de tipo String contienen los datos (número del cliente y su nombre) de los EditText, en la actividad principal.

in.putExtra(**"nomb"**,**nomcliente**);  
in.putExtra(**"idc"**,**nocliente**);

Con .putExtra agregamos valores al objeto in, los cuales serán enviados a la otra actividad. El primer parámetro es el nombre de la variable que se enviará (nosotros asignamos ese nombre), el segundo parámetro es el valor a enviar, el cual se recuperara en la otra actividad mediante el nombre de la variable que se asignó.

startActivity(in);

Y finalmente con la última línea indicamos que actividad se iniciara mediante el nombre del objeto el cual ya tiene la información a enviar, y la clase a llamar.

Agregaremos una actividad vacia de nombre **MainActivityLV**, esta deberá agregarse desde app y el layaout de esta actividad se muestra a continuación:

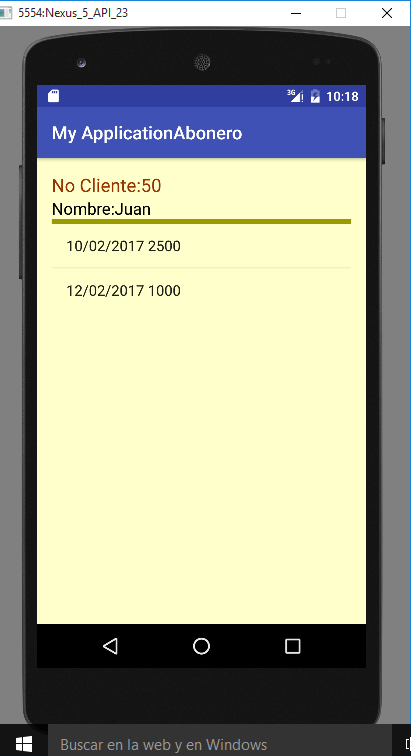
|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_main\_lv"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  android:background="#ffffcc"  android:orientation="vertical"  tools:context="com.example.adminagz.myapplicationabonero.MainActivityLV"**>   <**TextView  android:id="@+id/tvNocte"  android:textColor="#993300"  android:textSize="20sp"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:text="Nocte"** />   <**TextView  android:id="@+id/tvEstado"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:text="Estado"  android:textColor="#000000"  android:textSize="18sp"** />  <**View  android:layout\_width="fill\_parent"  android:layout\_height="5sp"  android:background="#999900"**/>   <**ListView  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/listView1"  android:layout\_alignParentTop="true"  android:layout\_centerHorizontal="true"** /> </**LinearLayout**> |

El código de java de esta actividad de nombre MainActivityLV.java se muestra a continuación:

|  |
| --- |
| **package** com.example.adminagz.myapplicationabonero;  **import** android.content.Intent; **import** android.database.Cursor; **import** android.database.sqlite.SQLiteDatabase; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.util.Log; **import** android.widget.ArrayAdapter; **import** android.widget.ListView; **import** android.widget.TextView; **import** android.widget.Toast;  **import** org.w3c.dom.Text;  **import** java.util.ArrayList; **import** java.util.List;  **public class** MainActivityLV **extends** AppCompatActivity {  TextView **tvnocte**,**tvestado**;  ListView **lista**;  AdminBD **db**;  List<String> **item** = **null**;    @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_lv***);  **lista** = (ListView) findViewById(R.id.***listView1***);  **tvnocte** = (TextView) findViewById(R.id.***tvNocte***);  **tvestado** = (TextView) findViewById(R.id.***tvEstado***);  showAbonos();  }   **private void** showAbonos()  {  Intent i = getIntent();*//Nos permite recuperar los valores* String idct, nombre;  *// Recuperamos los valores que fueron enviados y  // los asignamos variables de tipo String* idct = i.getStringExtra(**"idc"**);  nombre=i.getStringExtra(**"nomb"**);  **tvnocte**.setText(**"No Cliente:"** + idct);  **tvestado**.setText(**"Nombre:"** + nombre);  **db** = **new** AdminBD(**this**);  String[] args = **new** String[] {}; *// Declara un arreglo vacio args* Cursor c = **db**.Consulta(**"Select \* from Abono where NoCliente="** + idct);  Log.*d*(**"El Query es"** , **"Select \* from Abono where NoCliente="** + idct);   **item** = **new** ArrayList<String>();  String fecha =**""**, abono = **""**;  **if** (c.moveToFirst())  {  **do**{  fecha = c.getString(1); *// Campo 1 es la fecha* abono = c.getString(2); *// El campo 2 es Importe del abono* **item**.add(fecha + **" "** + abono);  }**while** (c.moveToNext());  }  ArrayAdapter<String> Adaptador = **new** ArrayAdapter<String>(**this**,  android.R.layout.***simple\_list\_item\_1***,**item**);  **lista**.setAdapter(Adaptador);  } } |

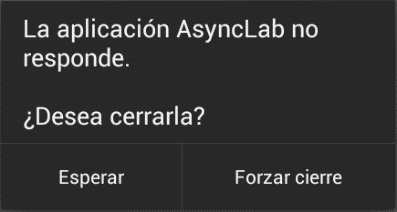
En esta se obtienen los parámetros idc y nomb que se enviaron desde el botón EdoCta de la actividad Anterior, para mostrar los abonos que tiene un determinado cliente.

La interfaz de esta actividad queda como a continuación se muestra:



**El Manejo de Hilos en Android**

Todos los componentes de una aplicación Android, tanto las actividades, los servicios (sí, también los servicios), o los broadcast receivers se ejecutan en el mismo hilo de ejecución, el llamado hilo principal, mainthread o GUI thread, que como éste último nombre indica también es el hilo donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación. Es por ello, que cualquier operación larga o costosa que realicemos en este hilo va a bloquear la ejecución del resto de componentes de la aplicación y por supuesto también la interfaz, produciendo al usuario un efecto evidente de lentitud, bloqueo, o mal funcionamiento en general, algo que deberíamos evitar a toda costa. Incluso puede ser peor, dado que Android monitoriza las operaciones realizadas en el hilo principal y detecta aquellas que superen los 5 segundos, en cuyo caso se muestra el famoso mensaje de “ApplicationNotResponding” (ANR) y el usuario debe decidir entre forzar el cierre de la aplicación o esperar a que termine.

[](http://www.hermosaprogramacion.com/wp-content/uploads/2014/12/android-dialogo-anr.png)

¿Qué pasaría si intentas cargar una imagen jpg de 3MB desde un servidor externo vía HTTP en tu aplicación?, si la conexión es rápida, tal vez nada. Pero para conexiones lentas esto tomara algunos segundos. ¿Crees que se vería muy bien, que tu aplicación se dedique a cargar primero la imagen y luego actualice la interfaz de usuario?

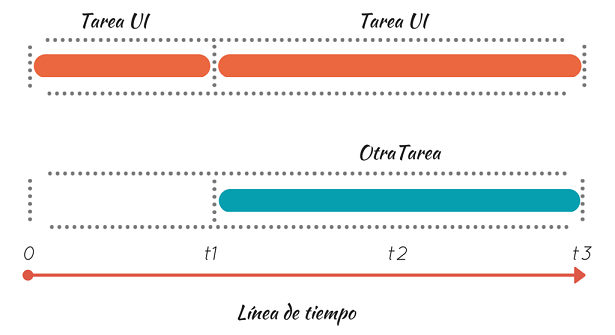
*¡En lo absoluto!*, esto arruina la fluidez visual y estropea la estadía de nuestros usuarios, lo que en la mayoría de casos termina en la eliminación de tu aplicación. A nivel computacional este caso podría apreciarse de la siguiente forma:

[](http://www.hermosaprogramacion.com/wp-content/uploads/2014/12/android-tareas-timeline.png)

La imagen ilustra la transición de las tareas que ocurren en el hilo principal de la aplicación. Si tu tarea toma algunos segundos se arruinaría la capacidad de respuesta, ya las tareas están en serie, es decir, hasta que una no acabe la otra no puede iniciar.

El camino correcto es renderizar la interfaz de la aplicación y al mismo tiempo ejecutar en segundo plano la otra actividad para continuar con la armonía de la aplicación y evitar paradas inesperadas. Es aquí donde entran los hilos, porque son los únicos que tienen la habilidad especial de permitir al programador generar concurrencia en sus aplicaciones y la sensación de multitareas ante el usuario.

Esta técnica es mostrada en el siguiente gráfico:

[](http://www.hermosaprogramacion.com/wp-content/uploads/2014/12/android-thread-concurrencia-timeline.png)

Se ha creado un nuevo hilo donde se ejecuta la tarea en el mismo intervalo de tiempo [t1, t2], pero esta vez el tiempo de ejecución de la tercera tarea UI se extendió debido a que se realizarán pequeños incrementos entre al segundo y tercera tarea. Aunque el tiempo empleado es el mismo, la respuesta ante el usuario simula una aplicación limpia.

Recuerda que existen dos tipos de procesamientos de tareas, **Concurrencia** y **Paralelismo**. La concurrencia se refiere a la existencia de múltiples tareas que se realizan simultáneamente compartiendo recursos de procesamiento.

El paralelismo es la ejecución de varias tareas al tiempo en distintas unidades de procesamiento, por lo que es mucho más rápido que la concurrencia.

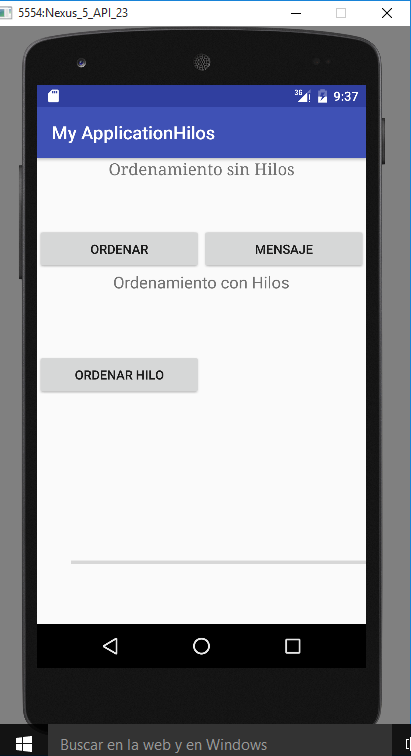
## Usar AsyncTasks en Android

Existen casos en los que se ejecutan varias instrucciones que deben presentar cambios en el hilo principal. Si se aplica el enfoque visto en la sección anterior el código para el envío de las ejecuciones tiende a ser muy largo, confuso y poco maleable a la hora de mantenimiento.

Por esta razón ha sido creada la interfaz AsyncTask, cuyo objetivo es liberar al programador del uso de hilos, la sincronización entre ellos y la presentación de resultados en el hilo primario. Esta clase unifica los aspectos relacionados que se realizarán en segundo plano y además gestiona de forma asíncrona la ejecución de las tareas.

Para implementarla debes extender una nueva clase con las características de AsyncTask e implementar los métodos correspondientes para la ejecución en segundo plano y la publicación de resultados en el UI Thread.

Para ir paso a paso, vamos a empezar por crear una aplicación de ejemplo en cuya actividad principal colocaremos un control TextView y cuatro botones, a continuación, se muestra como debe de quedar la interfaz:



El botón de ordenar, manda el proceso de ordenamiento por burbuja en el hilo principal de manera que se bloquea el dispositivo, y si quiere presionar el botón de MSG, este no se ejecuta por que el dispositivo esta ocupado.

En el caso de presionar el botón de Ordenar Hilos, el proceso de ordenamiento se ejecuta en un hilo y nos va mostrando el porcentaje de avance del ordenamiento, permitiendo presionar el botón de MSG, que lo único que hace es mandar un mensaje “Esta es una prueba de tarea alterna”, demostrando con esto que la ejecución de un proceso en un hilo no nos bloquea el dispositivo. De igual manera puede presionar el botón de Cancelar Hilo y el proceso que esta corriendo en el hilo se detendrá.

El archivo xml de la interfaz de nuestra aplicación se muestra a continuación:

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  tools:context="com.example.adminagz.myapplicationhilos.MainActivityHilos"**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="76dp"  android:orientation="vertical"**>   <**TextView  android:id="@+id/tv1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="center\_horizontal"  android:text="Ordenamiento sin Hilos"  android:textSize="18dp"  android:typeface="serif"** />  </**LinearLayout**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal"**>   <**Button  android:id="@+id/btnOrdena"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="Ordenar"** />   <**Button  android:id="@+id/btnMSG"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="Mensaje"** />   </**LinearLayout**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="90dp"  android:baselineAligned="false"  android:orientation="vertical"**>   <**TextView  android:id="@+id/tv2"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="center\_horizontal"  android:text="Ordenamiento con Hilos"  android:textSize="18dp"  android:typeface="sans"** />  </**LinearLayout**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal"**>   <**Button  android:id="@+id/btnOrdenaHilo"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="Ordenar Hilo"** />   <**Button  android:id="@+id/btnCancelarHilo"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="Cancelar Hilo"** />   </**LinearLayout**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="98dp"  android:orientation="vertical"**>  </**LinearLayout**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="76dp"  android:orientation="vertical"**>   <**TextView  android:id="@+id/tvEsperando"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="center\_horizontal"  android:background="@color/colorAccent"  android:text="Esperando..."** />  </**LinearLayout**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal"**>   <**ProgressBar  android:id="@+id/ProgressBar1"  style="?android:attr/progressBarStyleHorizontal"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:layout\_alignParentLeft="true"  android:layout\_alignParentRight="true"  android:layout\_alignParentTop="true"  android:layout\_marginTop="20dp"  android:layout\_gravity="center\_horizontal"**/>  </**LinearLayout**> </**LinearLayout**> |

El código de la actividad java se muestra a continuación:

|  |
| --- |
| **package** com.example.adminagz.myapplicationhilos;  **import** android.os.AsyncTask; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.util.Log; **import** android.view.View; **import** android.widget.Button; **import** android.widget.ProgressBar; **import** android.widget.TextView; **import** android.widget.Toast;  **import** java.util.Random;  **public class** MainActivityHilos **extends** AppCompatActivity **implements** View.OnClickListener {  Random **generador**;  **int**[] **numerosAordenar** = **new int**[20010];  TextView **tvespera**;  Button **OrdenarHilo**, **CancelarHilo**;  TareaSimple **SimpleTask**;  ProgressBar **pbrDialog**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_hilos***);  **tvespera** = (TextView) findViewById(R.id.***tvEsperando***);  **OrdenarHilo** = (Button) findViewById(R.id.***btnOrdenaHilo***);  **CancelarHilo** = (Button) findViewById(R.id.***btnCancelarHilo***);  **pbrDialog** = (ProgressBar) findViewById(R.id.***ProgressBar1***);  findViewById(R.id.***btnOrdena***).setOnClickListener(**this**); *// Activa el escucha del bóton de ordenar* findViewById(R.id.***btnMSG***).setOnClickListener(**this**); *// Activa el escucha del bóton de MSG* **OrdenarHilo**.setOnClickListener(**this**); *// Activa el escucha del boton* **CancelarHilo**.setOnClickListener(**this**); *// Activa el escucha del boton* **CancelarHilo**.setVisibility(View.***INVISIBLE***);  **tvespera**.setVisibility(View.***INVISIBLE***);  **generador** = **new** Random();  **for** (**int** i = 0; i <= 20000; i++) {  **numerosAordenar**[i] = Math.*abs*(**generador**.nextInt() % 10000);  }  }   @Override  **public void** onClick(View v) {  **switch** (v.getId()) {  **case** R.id.***btnOrdena***:  **tvespera**.setText(**"Esperando..."**);  **tvespera**.setVisibility(View.***VISIBLE***);  Ordenar();  **tvespera**.setText(**"Se ordenaron 20000 numeros "** + **numerosAordenar**[1000] + **", "** + **numerosAordenar**[1001]);  **break**;  **case** R.id.***btnMSG***:  Toast.*makeText*(**this**, **"Prueba de segunda tarea"**, Toast.***LENGTH\_LONG***).show();  **break**;  **case** R.id.***btnCancelarHilo***:  **SimpleTask**.cancel(**true**);  **pbrDialog**.setProgress(0);  **break**;  **case** R.id.***btnOrdenaHilo***:  **SimpleTask** = **new** TareaSimple();  **SimpleTask**.execute();  **break**;  }  }   **public void** Ordenar() {  **int** aux;  **for** (**int** i = 0; i <= 20000; i++) {  **for** (**int** j = 0; j <= 20000 - 1; j++) {  **if** (**numerosAordenar**[j] > **numerosAordenar**[j + 1]) {  aux = **numerosAordenar**[j];  **numerosAordenar**[j] = **numerosAordenar**[j + 1];  **numerosAordenar**[j + 1] = aux;  }  }  }  }   **public class** TareaSimple **extends** AsyncTask<Void, Integer, Void> {   @Override  **protected void** onPreExecute() {  **super**.onPreExecute();  **CancelarHilo**.setVisibility(View.***VISIBLE***);  **OrdenarHilo**.setEnabled(**false**);  **tvespera**.setVisibility(View.***VISIBLE***);  **pbrDialog**.setMax(100);  **pbrDialog**.setProgress(0);  }   @Override  **protected** Void doInBackground(Void... params) {  **int** aux;  **for** (**int** i = 0; i <= 20000; i++) {  **for** (**int** j = 0; j <= 20000 - 1; j++) {  **if** (**numerosAordenar**[j] > **numerosAordenar**[j + 1]) {  aux = **numerosAordenar**[j];  **numerosAordenar**[j] = **numerosAordenar**[j + 1];  **numerosAordenar**[j + 1] = aux;  }  }  **if** (!isCancelled()) publishProgress((**int**) (((i + 1) / (**float**) (**numerosAordenar**.**length** - 1)) \* 100));  **else  break**;  }  **return null**;  }   @Override  **protected void** onPostExecute(Void aVoid) {  **super**.onPostExecute(aVoid);  **tvespera**.setText(**"Ordenamiento completado"**);  **OrdenarHilo**.setEnabled(**true**);  **CancelarHilo**.setVisibility(View.***INVISIBLE***);  }   @Override  **protected void** onProgressUpdate(Integer... values) {  **super**.onProgressUpdate(values);  **tvespera**.setText(values[0] + **"%"**);  **pbrDialog**.setProgress(values[0]);  }   @Override  **protected void** onCancelled() {  **super**.onCancelled();  **tvespera**.setText(**"Hilo Cancelado..."**);  **CancelarHilo**.setVisibility(View.***INVISIBLE***);  **OrdenarHilo**.setEnabled(**true**);  }  } } |

TareaSimple se extiende de AsyncTask que además de ser abstracta es genérica. Las tres variables de entrada que posee se refieren a los Parámetros, Unidades de Progreso y Resultados respectivamente.

*// hay que sobreescribir estos métodos onPostExecute, onProgressUpdate, onCancelled, doInBackground, onPreExecute***private class** TareaSimple **extends** AsyncTask<Void, Integer, Void> {

La clase AsyncTask posee métodos que te permitirán coordinar la ejecución de las tareas que deseas ubicar en segundo plano. Estos métodos tienen los siguientes propósitos:

* onPreExecute(): En este método van todas aquellas instrucciones que se ejecutarán antes de iniciar la tarea en segundo plano. Normalmente es la inicialización de variables, objetos y la preparación de componentes de la interfaz.
* doInBackground(Parámetros…): Recibe los parámetros de entrada para ejecutar las instrucciones especificas que irán en segundo plano, luego de que ha terminado onPreExecute(). Dentro de él podemos invocar un método auxiliar llamado **publishProgress**(), el cual transmitirá unidades de progreso al hilo principal. Estas unidades miden cuanto tiempo falta para terminar la tarea, de acuerdo a la velocidad y prioridad que se está ejecutando.
* onProgressUpdate(Progreso…): Este método se ejecuta en el hilo de UI luego de que publishProgress() ha sido llamado. Su ejecución se prolongará lo necesario hasta que la tarea en segundo plano haya sido terminada. Recibe las unidades de progreso, así que podemos usar algún View para mostrarlas al usuario para que este sea consciente de la cantidad de tiempo que debe esperar.
* onPostExecute(Resultados…): Aquí puedes publicar todos los resultados retornados por doInBackground() hacia el hilo principal.
* onCancelled(): Ejecuta las instrucciones que desees que se realicen al cancelar la tarea asíncrona.

En el caso de doInBackground() se han puesto parámetros de tipo Void, ya que no se recibe valores de entrada y solo se ejecutarán las instrucciones del ordenamiento burbuja. Si eres buen observador, el método publishProgress() aparece al finalizar el primer bucle for. Esto con el fin de obtener una medida relativa en tiempo real del progreso actual. ¡Matemáticas básicas!

El tipo de dato para las unidades de progreso es Integer, así se obtienen números enteros que muestren un porcentaje entre el intervalo [0, 100]. Dicha medida se muestra en un TextView llamado progressLabel.

*// Desplegado del progreso del ordenamiento en tiempo real* @Override  
 **protected void** onProgressUpdate(Integer... values) {  
 **super**.onProgressUpdate(values);  
 **tvEspera**.setText(values[0] + **"%"**);  
 }

En onPostExecute() se recibe un tipo Void como resultado, debido a que no se recibe retorno de doInBackground(). Aquí aprovecharás para restablecer OrdenarHilo y CancelarHilo a su estado inicial. También puedes avisar a través de tvEspera que se ha completado el trabajo.

*//Se notifica que se completó el ordenamiento y se habilita  
 //de nuevo el botón "Ordenar"* @Override  
 **protected void** onPostExecute(Void aVoid) {  
 **super**.onPostExecute(aVoid);  
 **tvEspera**.setText(**"Completado"**);  
 **OrdenarHilo**.setEnabled(**true**);  
 }

### **Cancelar una tarea Asíncrona**

Puedes detener la ejecución de una tarea asíncrona usando el método cancel(). Este invoca el método onCancelled(), en vez de doInBackground(), por lo que se descartarán los resultados que estén por entregarse al hilo principal. Si necesitas saber el momento exacto en que terminó la tarea, puedes comprobar el valor arrojado por el método isCancelled()(que retorna en true si ya se ha cancelado):

En AsyncLab la tarea asíncrona se cancela en el método CancelarHilo.

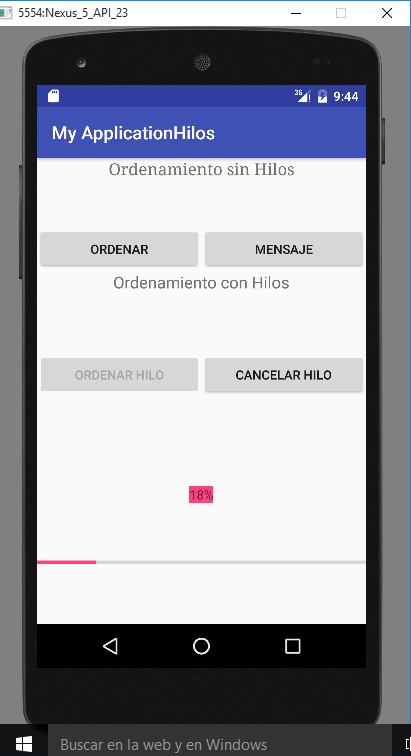
**public void** CancelaHilo(View v)  
{  
 **SimpleTask**.cancel(**true**);  
}

Como viste en la definición de SimpleTask, onCancelled() se sobrescribe para que ya no muestre unidades de progreso en progressLabel y para restablecer los estados de los botones:

*// Se informa en progressLabel que se canceló la tarea y  
// se hace invisile el botón "Cancelar"* @Override  
 **protected void** onCancelled() {  
 **super**.onCancelled();  
 **tvEspera**.setText(**"En la Espera"**);  
 **CancelarHilo**.setVisibility(View.***INVISIBLE***);  
 **OrdenarHilo**.setEnabled(**true**);  
 }

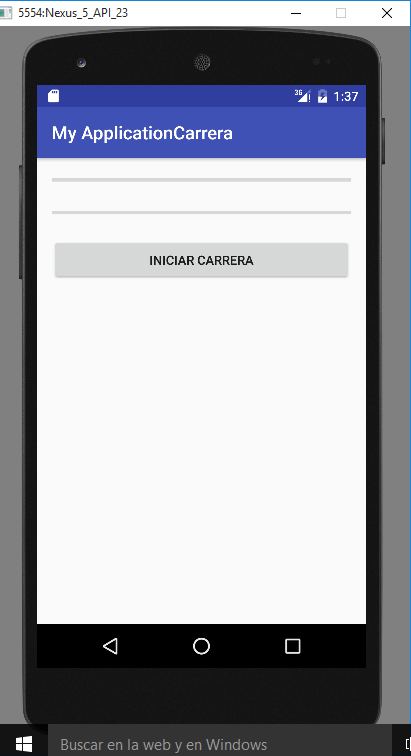
Complementariamente se coordina la detención del ordenamiento burbuja con un break si en algún momento la actividad se ha cancelado:

*// Notifica a onProgressUpdate() del progreso actual***if**(!isCancelled())  
 publishProgress((**int**)(((i+1)/(**float**)(**numerosAordenar**.**length**-1))\*100));  
**else break**;



**Carrera de caballos tareas asincronas**

Para ejemplificar el uso y aplicación de la programación con hilos, vamos hacer una carrera de caballos con dos controles PogresBar, la aplicación debe quedar como a continuación se muestra:



El código del archivo XML, se muestra a continuación:

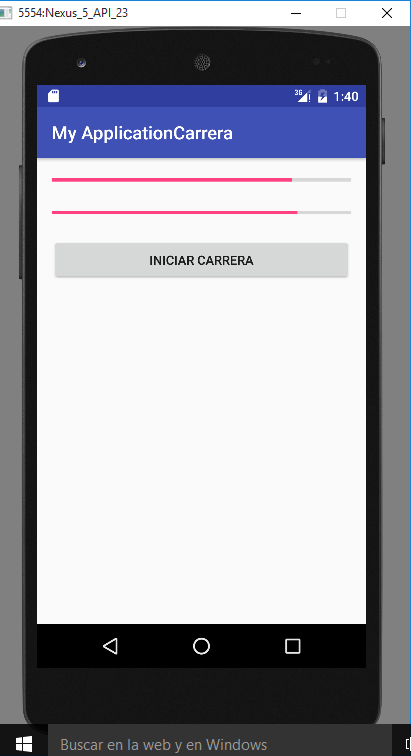
|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_main\_carrera"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  android:orientation="vertical"  tools:context="com.example.adminagz.myapplicationcarrera.MainActivityCarrera"**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal"**>   <**ProgressBar  android:id="@+id/Caballo1"  style="?android:attr/progressBarStyleHorizontal"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:layout\_alignParentLeft="true"  android:layout\_alignParentRight="true"  android:layout\_alignParentTop="true"  android:layout\_marginTop="20dp"  android:layout\_gravity="center\_horizontal"**/>  </**LinearLayout**>  <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal"**>   <**ProgressBar  android:id="@+id/Caballo2"  style="?android:attr/progressBarStyleHorizontal"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:layout\_alignParentLeft="true"  android:layout\_alignParentRight="true"  android:layout\_alignParentTop="true"  android:layout\_marginTop="20dp"  android:layout\_gravity="center\_horizontal"**/>  </**LinearLayout**>   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:orientation="horizontal"**>   <**Button  android:id="@+id/btnCarrera"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="Iniciar Carrera"** />  </**LinearLayout**>  </**LinearLayout**> |

El código de la actividad en JAVA, se muestra a continuación:

|  |
| --- |
| **package** com.example.adminagz.myapplicationcarrera;  **import** android.annotation.TargetApi; **import** android.os.AsyncTask; **import** android.os.Build; **import** android.support.v7.app.AppCompatActivity; **import** android.os.Bundle; **import** android.util.Log; **import** android.view.View; **import** android.widget.Button; **import** android.widget.ProgressBar; **import** android.widget.TextView; **import** android.widget.Toast;  **import** java.lang.annotation.Target; **import** java.util.Random;  **public class** MainActivityCarrera **extends** AppCompatActivity **implements** View.OnClickListener {  **static int** *fin* = 0;  Random **generador**;  **int**[] **numerosAordenar1** = **new int**[20010];  **int**[] **numerosAordenar2** = **new int**[20010];   Button **IniciarCarrera**;  TareaCaballo1 **CaballoTask1**; *// Variable para manejar la tarea asincrona* TareaCaballo2 **CaballoTask2**; *// Variable para manejar la tarea asincrona* ProgressBar **Caballo1**,**Caballo2**;    @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_carrera***);  **IniciarCarrera** = (Button) findViewById(R.id.***btnCarrera***);  **Caballo1** = (ProgressBar) findViewById(R.id.***Caballo1***);  **Caballo2** = (ProgressBar) findViewById(R.id.***Caballo2***);   **IniciarCarrera**.setOnClickListener(**this**);   **generador** = **new** Random();  **for** (**int** i = 0; i <= 20000; i++) {  **numerosAordenar1**[i] = Math.*abs*(**generador**.nextInt() % 10000);  **numerosAordenar2**[i] = Math.*abs*(**generador**.nextInt() % 10000);  }  }   *// Código para ejecutar los hilos en forma concurrente* @TargetApi(Build.VERSION\_CODES.***HONEYCOMB***)  **public static** <T> **void** executeTask(AsyncTask<T, ?, ?> asyncTask, T... params){  **if** (Build.VERSION.***SDK\_INT*** >= Build.VERSION\_CODES.***HONEYCOMB***)  asyncTask.executeOnExecutor(AsyncTask.***THREAD\_POOL\_EXECUTOR***, params);  **else** asyncTask.execute(params);  }   @Override  **public void** onClick(View v) {  **switch** (v.getId()) {  **case** R.id.***btnCarrera***:  *fin* = 0;  **CaballoTask1** = **new** TareaCaballo1();  **CaballoTask2** = **new** TareaCaballo2();  *executeTask*(**CaballoTask1**);  *executeTask*(**CaballoTask2**);  **break**;  }  }   **public class** TareaCaballo1 **extends** AsyncTask<Void, Integer, Void> {   @Override  **protected void** onPreExecute() {  **super**.onPreExecute();  **Caballo1**.setMax(100);  **Caballo1**.setProgress(0);  }   @Override  **protected** Void doInBackground(Void... params) {  **int** aux;  **for** (**int** i = 0; i <= 20000; i++) {  **for** (**int** j = 0; j <= 20000; j++) {  **if** (**numerosAordenar1**[j] > **numerosAordenar1**[j + 1]) {  aux = **numerosAordenar1**[j];  **numerosAordenar1**[j] = **numerosAordenar1**[j + 1];  **numerosAordenar1**[j + 1] = aux;  }  }  **if** (!isCancelled())  publishProgress((**int**) (((i + 1) / (**float**) (**numerosAordenar1**.**length** - 1)) \* 100));  **else  break**;  }  **return null**;  }   @Override  **protected void** onPostExecute(Void aVoid) {  **super**.onPostExecute(aVoid);  **if** (*fin* == 0)  *fin* = 1;  Toast.*makeText*(getBaseContext(), **"El Caballo ganador es:"**+*fin*, Toast.***LENGTH\_LONG***).show();  }   @Override  **protected void** onProgressUpdate(Integer... values) {  **super**.onProgressUpdate(values);  **Caballo1**.setProgress(values[0]);  }   @Override  **protected void** onCancelled() {  **super**.onCancelled();   }  } *//----------------------------------------------------------------------------* **public class** TareaCaballo2 **extends** AsyncTask<Void, Integer, Void> {   @Override  **protected void** onPreExecute() {  **super**.onPreExecute();  **Caballo2**.setMax(100);  **Caballo2**.setProgress(0);  }   @Override  **protected** Void doInBackground(Void... params) {  **int** aux;  **for** (**int** i = 0; i <= 20000; i++) {  **for** (**int** j = 0; j <= 20000; j++) {  **if** (**numerosAordenar2**[j] > **numerosAordenar2**[j + 1]) {  aux = **numerosAordenar2**[j];  **numerosAordenar2**[j] = **numerosAordenar2**[j + 1];  **numerosAordenar2**[j + 1] = aux;  }   }  **if** (!isCancelled())  publishProgress((**int**) (((i + 1) / (**float**) (**numerosAordenar2**.**length** - 1)) \* 100));  **else  break**;  }  **return null**;  }   @Override  **protected void** onPostExecute(Void aVoid) {  **super**.onPostExecute(aVoid);  **if** (*fin* == 0)  *fin* = 2;  Toast.*makeText*(getBaseContext(), **"El Caballo ganador es:"**+*fin*, Toast.***LENGTH\_LONG***).show();  }   @Override  **protected void** onProgressUpdate(Integer... values) {  **super**.onProgressUpdate(values);  **Caballo2**.setProgress(values[0]);  }   @Override  **protected void** onCancelled() {  **super**.onCancelled();  }  } } |

El código que se muestra en Amarillo, con letras rojas es el que permite ejecutar los hilos en forma concurrente.

La ejecución de la aplicación se muestra a continuación:



**Utilización de WEB SERVICE EN ANDROID**

****

Un Web Service o Servicio Web es una aplicación que facilita la comunicación entre varios sistemas independientemente del lenguaje de programación o plataforma en que fueron desarrollados. Este debe tener una interfaz basada en un formato estándar entendible por las maquinas como lo es XML o JSON.

*Otra definición*

Es una tecnología que posibilita el intercambio de datos entre aplicaciones, que pueden estar en lenguajes o plataformas diferentes. Por ejemplo, los bancos tienen un conjunto de servicios web para mostrar información de sus cuentas y desde cualquier software que se cree que posibilite consumir estos servicios se podrá acceder a dicha información, sin importar el lenguaje en que esté escrito o la plataforma en la que se ejecute.

Otro ejemplo, la red social Facebook es una aplicación web construido con una determinada arquitectura y lenguajes de programación basados en el protocolo HTTP. Sin embargo, podemos usar esta red social en nuestro dispositivo Android.

*¿Cómo es posible esto, si la aplicación Android está construida con lenguaje Java?*

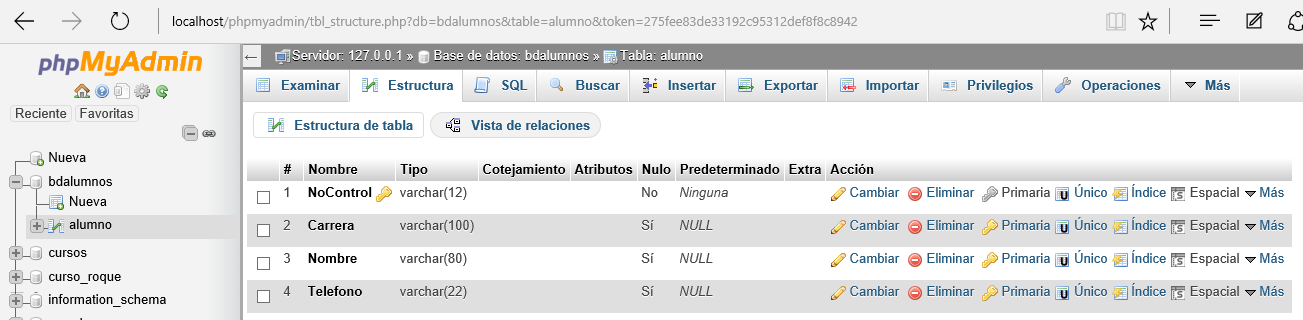
A través de un Web Service construido para gestionar todas aquellas operaciones sobre una base de datos alojada en los servidores de Facebook. Quiere decir que ambas aplicaciones usan como puente la web para acceder a un solo repositorio de datos.

Como ves, un Web Service se crea con funcionalidades que permitan obtener datos actualizados en tiempo real. El hecho de que sea dinámico incorpora el uso de un lenguaje web para la gestión HTTP que en este caso será Php.

Para ejemplificar el uso de servicios web vamos a crear una aplicación Android que permita dar de alta/bajas/cambios y consultas de alumnos de un servidor de base de datos en MySQL

Procedemos a instalar un servidor web, utilizaremos XAMPP, el cual instala Apache, PHP y MySQL para crear nuestro servicio web, usando JSON.

Entremos a MySQL y creamos una base de datos de nombre BDalumnos, y en está creamos la tabla alumno de acuerdo a la estructura que muestra la siguiente figura:



Inserte unos 6 registros en la tabla alumno para poder probar los Web Services.

**¿Qué es y para qué sirve JSON?**

Obtenido de: https://geekytheory.com/json-i-que-es-y-para-que-sirve-json/



JSON (**JavaScript Object Notation**) es un formato para el intercambio de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. JSON nació como una alternativa a XML, el fácil uso en javascript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

Imaginemos que tenemos una frutería y que queremos obtener el nombre y la cantidad de fruta y verdura que tenemos. En un principio vamos a suponer que tenemos lo siguiente:

* Fruta:
* 10 manzanas
* 20 Peras
* 30 Naranjas
* Verduras
* 80 lechugas
* 15 tomates
* 50 pepinos

Para empezar, nos tenemos que familiarizar con la sintaxis de Json:

**JSON Nombre/Par de Valores**

Para asignar a un nombre un valor debemos usar los dos puntos ':' este separador es el equivalente al igual ('=') de cualquier lenguaje.

"Nombre" : "Alex Lora"

Los tipos de valores que podemos encontrar en Json son los siguientes:

* Un número (entero o float)
* Un string (entre comillas simples)
* Un booleano (true o false)
* Un array (entre corchetes [] )
* Un objeto (entre llaves {})
* Null

Objetos JSON

Los objetos JSON se identifican entre llaves, un objeto puede ser en nuestro caso una fruta o una verdura.

{ "NombreFruta":"Manzana" , "Cantidad":20 }

Arrays JSON

En un JSON puedes incluir arrays, para ellos el contenido del array debe ir entre corchetes []:

{

"Frutas": [

{ "NombreFruta":"Manzana" , "cantidad":10 },

{ "NombreFruta":"Pera" , "cantidad":20 },

{ "NombreFruta":"Naranja" , "cantidad":30 }

]

}

Una vez explicado el funcionamiento de la sintaxis JSON, vamos a aplicar nuestro ejemplo de la frutería:

{"Fruteria":

[

{"Fruta":

[

{"Nombre":"Manzana","Cantidad":10},

{"Nombre":"Pera","Cantidad":20},

{"Nombre":"Naranja","Cantidad":30}

]

},

{"Verdura":

[

{"Nombre":"Lechuga","Cantidad":80},

{"Nombre":"Tomate","Cantidad":15},

{"Nombre":"Pepino","Cantidad":50}

]

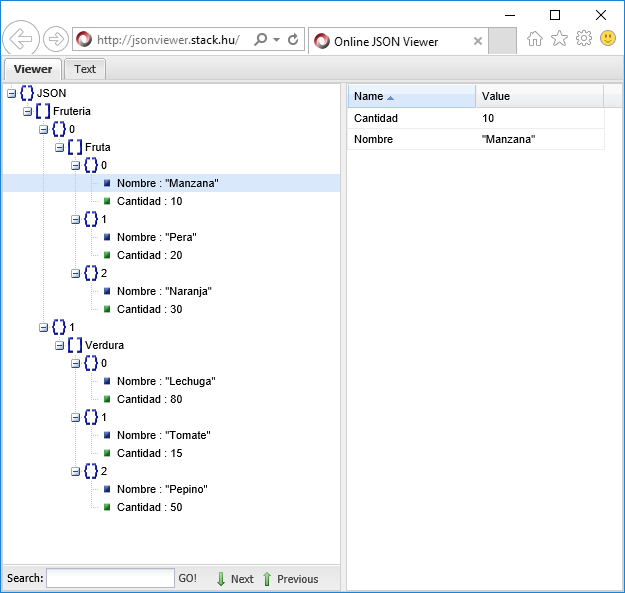
}

]

}

Como podemos observar, hemos creado un objeto llamado frutería y, dentro de ese objeto hemos almacenado un array de dos elementos. El primer elemento del array contiene un objeto llamado fruta y el segundo elemento del array contiene otro objeto llamado verdura. Estos objetos a su vez contienen un array cuyo contenido es el nombre y la cantidad de cada fruta o verdura.

**Existen herramientas online que ayudan a visualizar mejor un JSON**. Una de las mejores herramientas es la página [JSON Viewer](http://jsonviewer.stack.hu/). Si introducimos nuestro ejemplo observamos lo siguiente:



Una vez creada la tabla de alumno, nos vamos a la carpeta htdocs, que es la carpeta de publicación de páginas web y creamos la carpeta ww, en esta vamos a crear nuestros servicios web con JSON. Los servicios web que debes crear son:

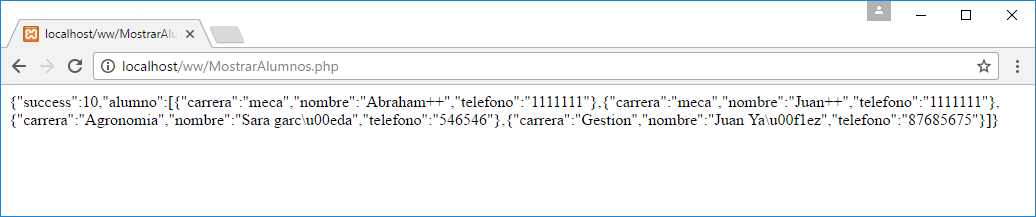
* MostrarAlumnos.php
* MostrarAlumno.php
* InsertEstudiante.php
* BorrarAlumno.php
* ActualizaAlumno.php

A continuación, mostramos el código de cada uno de estos servicios web:

MostrarAlumnos.php

|  |
| --- |
| <?php  /\*  \* El siguiente código muestra registros que se encuentran en la tabla de alumno  \* AGZ 3/11/2016  \*/  $response = array(); // arreglo para JSON response  $Cn = mysqli\_connect("localhost","root","","BDalumnos")or die ("server no encontrado");  mysqli\_set\_charset($Cn,"utf8"); //Para que permita acentos y ñ  $result = mysqli\_query($Cn,"SELECT carrera,nombre,telefono from alumno");    if (!empty($result)) {  if (mysqli\_num\_rows($result) > 0) {  $alumno = array();  $response["success"] = 10; // El success=10 es que encontro registros en la tabla alumno y fue exitoso  $response["alumno"] = array();  while ($ren = mysqli\_fetch\_array($result))  {  $alumno["carrera"] = $ren["carrera"];  $alumno["nombre"] = $ren["nombre"];  $alumno["telefono"] = $ren["telefono"];  array\_push($response["alumno"], $alumno);  }  // codifica la información en formato de JSON response  echo json\_encode($response);  } else {  // alumnos no encontrados  $response["success"] = 0; //No encontro información y el success = 0 indica no exitoso  $response["message"] = "Alumnos no encontrados";  echo json\_encode($response);  }  } else {  // alumnos no encontrados  $response["success"] = 0; //No encontro información y el success = 0 indica no exitoso  $response["message"] = "Alumno no encontrado";  echo json\_encode($response);  }  mysqli\_close($Cn);  ?> |

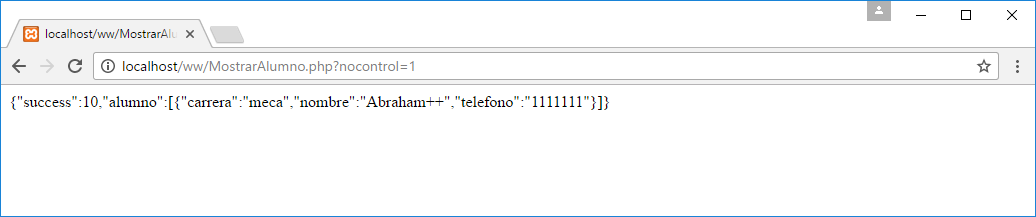
El resultado que arroja cuando invocamos al web service “MostrarAlumnos” es:



MostrarAlumno.php

|  |
| --- |
| <?php  /\*  \* El siguiente código localiza un alumno  \* AGZ 3/11/2016  \*/  $response = array();  $Cn = mysqli\_connect("localhost","root","","BDalumnos")or die ("server no encontrado");  mysqli\_set\_charset($Cn,"utf8");  // Checa que le este llegando por el método GET el nocontrol  if (isset($\_GET["nocontrol"])) {  $nocontrol = $\_GET['nocontrol'];  $result = mysqli\_query($Cn,"SELECT carrera,nombre,telefono from alumno WHERE Nocontrol = $nocontrol");  if (!empty($result)) {  if (mysqli\_num\_rows($result) > 0) {  $result = mysqli\_fetch\_array($result);  $alumno = array();  $alumno["carrera"] = $result["carrera"];  $alumno["nombre"] = $result["nombre"];  $alumno["telefono"] = $result["telefono"];      $response["success"] = 10; // El success=10 es que encontro al alumno y fue exitoso  $response["alumno"] = array();  array\_push($response["alumno"], $alumno);  // codifica la información en formato de JSON response  echo json\_encode($response);  } else {  // No Encontro al alumno  $response["success"] = 0; //No encontro información y el success = 0 indica no exitoso  $response["message"] = "Alumno no encontrado";  echo json\_encode($response);  }  } else {  $response["success"] = 0; //No encontro información y el success = 0 indica no exitoso  $response["message"] = "Alumno no encontrado";  echo json\_encode($response);  }  } else {  // required field is missing  $response["success"] = 0;  $response["message"] = "Faltan Datos";  // echoing JSON response  echo json\_encode($response);  }  mysqli\_close($Cn);  ?> |

Cuando invocamos el web service de “MostrarAlumno” arroja el siguiente resultado:



Como puede observar se le manda el parámetro nocontrol por el método GET.

InsertEstudiante.php

|  |
| --- |
| <?php  /\*  \* El siguiente código inserta un registros en la tabla del alumno, recibiendo los datos  \* por el método POST  \* AGZ 3/11/2016  \*/  $Cn = mysqli\_connect("localhost","root","","BDalumnos") or die ("Error al conectarse con servidor");  mysqli\_set\_charset($Cn,"utf8");  $response = array();    if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {  $body = json\_decode(file\_get\_contents("php://input"),true);  $nocontrol = $body['nocontrol'];  $carrera = $body['carrera'];  $nombre = $body['nombre'];  $telefono = $body['telefono'];    $result = mysqli\_query($Cn,"INSERT INTO alumno (nocontrol,carrera,nombre,telefono) VALUES ('$nocontrol','$carrera','$nombre','$telefono') ");    if($result){  $response["success"] = 20; // El success=20 es que inserto el registros y fue exitoso  $response["message"] = "Se inserto correctamente";  echo json\_encode($response);  }else{  $response["success"] = 0; //No encontro información y el success = 0 indica no exitoso  $response["message"] = "Ocurrio un error durante la insercion de alumno";  echo json\_encode($response);  }  }else{  $response["success"] = 0; //No encontro información y el success = 0 indica no exitoso  $response["message"] = "Se necesitan datos para la inserncion";  echo json\_encode($response);  }  mysqli\_close($Cn);  ?> |

BorrarAlumno.php

|  |
| --- |
| <?php  /\*  \* El siguiente código elimina un alumno, recibiendo por el método POST el nocontrol  \* AGZ 3/11/2016  \*/  $Cn = mysqli\_connect("localhost","root","","BDalumnos")or die ("server no encontrado");  mysqli\_set\_charset($Cn,"utf8");  $response = array();  if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {  $body = json\_decode(file\_get\_contents("php://input"),true);  $nocontrol = $body['nocontrol'];  $result = mysqli\_query($Cn,"DELETE FROM alumno WHERE nocontrol = '$nocontrol'");    // checa si el registro fue borrado o no  if (mysqli\_affected\_rows($Cn) > 0) {  // successfully updated  $response["success"] = 40; // El success=40 es que elimino el alumno y fue exitoso  $response["message"] = "Alumno se elimino correctamente";  // echoing JSON response  echo json\_encode($response);  } else {  // no encontro el alumno  $response["success"] = 0; // success = 0 es que no fue exitoso el borrado  $response["message"] = "Alumno no encontrado";  echo json\_encode($response);  }  } else {  // falto el nocontrol por el método POST  $response["success"] = 0; // success = 0 es que no fue exitoso el borrado  $response["message"] = "Requiere el numero de control";  echo json\_encode($response);  }  mysqli\_close($Cn);  ?> |

ActualizaAlumno.php

|  |
| --- |
| <?php  /\*  \* El siguiente código actualiza los datos de un alumno,  \* recibiendo por el método POST el nocontrol, carrera, nombre y telefono  \* AGZ 3/11/2016  \*/  $Cn = mysqli\_connect("localhost","root","", "BDalumnos")or die ("server no encontrado");  $response = array();  if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {    $body = json\_decode(file\_get\_contents("php://input"),true);  $nocontrol = $body['nocontrol'];  $carrera = $body['carrera'];  $nombre = $body['nombre'];  $telefono = $body['telefono'];    $result = mysqli\_query($Cn,"UPDATE alumno SET carrera='$carrera',nombre='$nombre',telefono='$telefono' WHERE Nocontrol='$nocontrol' ");  // checa si el registro fue actualizado  if ($result)  {  // Acualización exitosa  $response["success"] = 30;  $response["message"] = "Alumno se actualizo correctamente.";    // echoing JSON response  echo json\_encode($response);  }  else  {  $response["success"] = 0; // Terminación no exitosa  $response["message"] = "problema al aztualizar";  echo json\_encode($response);  }  }  else  {  // requiere información de entrada por el método POST  $response["success"] = 0; // Terminación no exitosa  $response["message"] = "Faltaron datos";  echo json\_encode($response);  }  mysqli\_close($Cn);  ?> |

Para continuar con nuestro ejemplo creamos una aplicación de Android de nombre WebService, con un LayOut de nombre MainActivityWS. A continuación, se muestra la interfaz de la aplicación:



El código se presenta a continuación:

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  tools:context="com.example.adminagz.webservice.MainActivityWS"**>    <**LinearLayout  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"**>   <**Button  android:layout\_width="0dp"  android:layout\_weight="1"  android:layout\_height="wrap\_content"  android:text="Consultar"  android:id="@+id/consultar"** />   <**Button  android:layout\_width="0dp"  android:layout\_weight="1"  android:layout\_height="wrap\_content"  android:text="Consultar ID"  android:id="@+id/consultarid"** />   <**Button  android:layout\_width="0dp"  android:layout\_weight="1"  android:layout\_height="wrap\_content"  android:text="Insertar"  android:id="@+id/insertar"** />   <**Button  android:layout\_width="0dp"  android:layout\_weight="1"  android:layout\_height="wrap\_content"  android:text="Borrar"  android:id="@+id/borrar"** />   <**Button  android:layout\_width="0dp"  android:layout\_weight="1"  android:layout\_height="wrap\_content"  android:text="Actualizar"  android:id="@+id/actualizar"** />  </**LinearLayout**>   <**LinearLayout  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"**>   <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Identificador"  android:id="@+id/tid"** />   <**EditText  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/eid"** />  </**LinearLayout**>  <**LinearLayout  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"**>   <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Nombre"  android:id="@+id/tnombre"** />   <**EditText  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/enombre"** />  </**LinearLayout**>  <**LinearLayout  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"**>   <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Carrera"  android:id="@+id/tcarrera"** />   <**EditText  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/ecarrera"** />  </**LinearLayout**>   <**LinearLayout  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"**>   <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Telefono"  android:id="@+id/ttelefono"** />   <**EditText  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/etelefono"** />  </**LinearLayout**>   <**LinearLayout  android:orientation="vertical"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_gravity="center\_horizontal"**>    <**TextView  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:text=""  android:id="@+id/resultado"** />  </**LinearLayout**>  </**LinearLayout**> |

A continuación, mostramos el código de la actividad “MainActivityWS”:

|  |
| --- |
| **package** com.example.adminagz.webservice;  **import** android.os.AsyncTask; **import** android.os.Bundle; **import** android.support.design.widget.FloatingActionButton; **import** android.support.design.widget.Snackbar; **import** android.support.v7.app.AppCompatActivity; **import** android.support.v7.widget.Toolbar; **import** android.view.View; **import** android.view.Menu; **import** android.view.MenuItem; **import** android.widget.Button; **import** android.widget.EditText; **import** android.widget.TextView; **import** android.widget.Toast;  **import** org.json.JSONArray; **import** org.json.JSONException; **import** org.json.JSONObject;  **import** java.io.BufferedInputStream; **import** java.io.BufferedReader; **import** java.io.BufferedWriter; **import** java.io.DataInputStream; **import** java.io.DataOutputStream; **import** java.io.IOException; **import** java.io.InputStream; **import** java.io.InputStreamReader; **import** java.io.OutputStream; **import** java.io.OutputStreamWriter; **import** java.net.HttpURLConnection; **import** java.net.MalformedURLException; **import** java.net.URI; **import** java.net.URL; **import** java.net.URLConnection;  **import** javax.net.ssl.HttpsURLConnection;  **public class** MainActivityWS **extends** AppCompatActivity **implements** View.OnClickListener {   Button **consultar**;  Button **consultarporid**;  Button **insertar**;  Button **actualizar**;  Button **borrar**;  EditText **identificador**;  EditText **nombre**;  EditText **carrera**;  EditText **telefono**;  TextView **resultado**;   String **IP** = **"http://192.168.1.67/ww"**; *// Localización de los webservices* String **GET** = **IP** + **"/MostrarAlumnos.php"**; *// Servicio web para consultar los alumnos* String **GET\_BY\_ID** = **IP** + **"/MostrarAlumno.php"**; *// Servicio web para consultar un solo alumno* String **DELETE** = **IP** + **"/BorrarAlumno.php"**; *// Servicio web para borrar un alumno de MySQL* String **INSERT** = **IP** + **"/InsertEstudiante.php"**; *// Servicio web para insertar un alumno de MySQL* String **UPDATE** = **IP** + **"/ActualizaAlumno.php"**; *// Servicio web para actualizar un alumno de MySQL* ObtenerWebService **hiloconexion**; *// declaración del hilo* @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main\_activity\_ws***);  Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  setSupportActionBar(toolbar);   *// Liga las variables con los objetos del la interfaz* **consultar** = (Button)findViewById(R.id.***consultar***);  **consultarporid** = (Button)findViewById(R.id.***consultarid***);  **insertar** = (Button)findViewById(R.id.***insertar***);  **actualizar** = (Button)findViewById(R.id.***actualizar***);  **borrar** = (Button)findViewById(R.id.***borrar***);  **identificador** = (EditText)findViewById(R.id.***eid***);  **nombre** = (EditText)findViewById(R.id.***enombre***);  **carrera** = (EditText)findViewById(R.id.***ecarrera***);  **telefono**= (EditText)findViewById(R.id.***etelefono***);  **resultado** = (TextView)findViewById(R.id.***resultado***);   **consultar**.setOnClickListener(**this**);  **consultarporid**.setOnClickListener(**this**);  **insertar**.setOnClickListener(**this**);  **actualizar**.setOnClickListener(**this**);  **borrar**.setOnClickListener(**this**);   FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.***fab***);  fab.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View view) {  Snackbar.*make*(view, **"Replace with your own action"**, Snackbar.***LENGTH\_LONG***)  .setAction(**"Action"**, **null**).show();  }  });  }   @Override  **public boolean** onCreateOptionsMenu(Menu menu) {  *// Inflate the menu; this adds items to the action bar if it is present.* getMenuInflater().inflate(R.menu.***menu\_main\_activity\_w***, menu);  **return true**;  }   @Override  **public boolean** onOptionsItemSelected(MenuItem item) {  *// Handle action bar item clicks here. The action bar will  // automatically handle clicks on the Home/Up button, so long  // as you specify a parent activity in AndroidManifest.xml.* **int** id = item.getItemId();   *//noinspection SimplifiableIfStatement* **if** (id == R.id.***action\_settings***) {  **return true**;  }   **return super**.onOptionsItemSelected(item);  }   *// Evento clic de lo botones* @Override  **public void** onClick(View v){  **switch** (v.getId()){  **case** R.id.***consultar***:  **hiloconexion** = **new** ObtenerWebService(); *// Genera la instancia del hilo* **hiloconexion**.execute(**GET**,**"1"**); *// Parametros del doInBackGround* **break**;  **case** R.id.***consultarid***:  **hiloconexion** = **new** ObtenerWebService(); *// Genera la instancia del hilos* String cadenaLlamada1 = **GET\_BY\_ID** + **"?nocontrol="** + **identificador**.getText().toString();  **hiloconexion**.execute(cadenaLlamada1,**"1"**); *// Parametros del doInBackGround* **break**;  **case** R.id.***insertar***:  **hiloconexion** = **new** ObtenerWebService(); *// Genera la instancia del hilos* **hiloconexion**.execute(**INSERT**,**"3"**,**identificador**.getText().toString(),**carrera**.getText().toString(),**nombre**.getText().toString(),**telefono**.getText().toString()); *// Parametros del doInBackGround* **break**;  **case** R.id.***actualizar***:  **hiloconexion** = **new** ObtenerWebService(); *// Genera la instancia del hilos* **hiloconexion**.execute(**UPDATE**,**"3"**,**identificador**.getText().toString(),**carrera**.getText().toString(),**nombre**.getText().toString(),**telefono**.getText().toString()); *// Parametros del doInBackGround* **break**;  **case** R.id.***borrar***:  **if** (**identificador**.getText().toString().trim().length() == 0) {  **identificador**.requestFocus();  Toast.*makeText*(getBaseContext(),**"Has olvidado rellenar el No de control"**, Toast.***LENGTH\_LONG***).show();  }**else** {  **hiloconexion** = **new** ObtenerWebService(); *// Genera la instancia del hilos* **hiloconexion**.execute(**DELETE**, **"5"**, **identificador**.getText().toString()); *// Parametros del doInBackGround* }  **break**;  **default**:  **break**;  }  }  *// Clase AsyncTask para manejar el Hilo  // hay que sobreescribir estos métodos doInBackground, onPostExecute* **public class** ObtenerWebService **extends** AsyncTask<String,Void,String>{  *// Este método recibe el JSON en s que resulta de la ejecución de doInBackground* @Override  **protected void** onPostExecute(String s) {  String devuelve=**""**;  **try** {  JSONObject respuestaJSON = **new** JSONObject(s.toString());  String resultJSON = respuestaJSON.getString(**"success"**); *//Obtiene el primer campo de JSON que es string y se llama estado  //resultado.setText(s.toString());* **switch** (resultJSON){ *// 0 = error, 10=Consulta 20=Insert 30= Update 40=Delete* **case "0"**:  String messageJSON1 = respuestaJSON.getString(**"message"**);  **resultado**.setText(messageJSON1);  **break**;  **case "10"**:  JSONArray alumnosJSON = respuestaJSON.getJSONArray(**"alumno"**);  *// Lee el segundo parametro que es un arreglo que trae la información de los alumnos  // idalumno, nombre, direccion* **if** (alumnosJSON.length()>1){  **for** (**int** i = 0; i < alumnosJSON.length(); i++) {  devuelve = devuelve + alumnosJSON.getJSONObject(i).getString(**"telefono"**) + **" "** +  alumnosJSON.getJSONObject(i).getString(**"carrera"**) + **" "** +  alumnosJSON.getJSONObject(i).getString(**"nombre"**) + **"\n"**;  }  **resultado**.setText(devuelve);  }  **else** {  **carrera**.setText(alumnosJSON.getJSONObject(0).getString(**"carrera"**));  **nombre**.setText(alumnosJSON.getJSONObject(0).getString(**"nombre"**));  **telefono**.setText(alumnosJSON.getJSONObject(0).getString(**"telefono"**));  }  **identificador**.requestFocus();  **break**;  **default**:  String messageJSON2 = respuestaJSON.getString(**"message"**);  Toast.*makeText*(getBaseContext(),messageJSON2, Toast.***LENGTH\_LONG***).show();  **identificador**.setText(**""**);  **carrera**.setText(**""**);  **nombre**.setText(**""**);  **telefono**.setText(**""**);  **identificador**.requestFocus();  **break**;  }  }  **catch** (JSONException e){  e.printStackTrace();  }  }   *// Este método es el que manda ejecutar los servicios web  // y recibe como parametro el nombre del servicio web a ejecutar un el numero  // 1 si ejecuto un boton de Consulta de alumnos  // 3 si ejecuto un boton de Inserta o Actualizar  // 5 si ejecuto el boton de Borrar* @Override  **protected** String doInBackground(String... params) {  String cadena = params[0];  URL url = **null**;  String devuelve = **""**;   **if** (params[1]==**"1"**) *// Si es 1 va por el primer webservice que es ObtenerAlumnos* {  StringBuilder result = **new** StringBuilder();  **try** {  url = **new** URL(cadena);  *// Abrimos la conexión hacia el servicio web alojado en el servidor* HttpURLConnection connection = (HttpURLConnection) url.openConnection();  connection.setRequestProperty(**"User-Agent"**, **"Mozilla/5.0"** +  **" (Linux; Android 1.5; es-ES) Ejemplo HTTP"**);  **int** respuesta = connection.getResponseCode();  *// Preguntamos si se pudo conectar al servidor con exito* **if** (respuesta == HttpURLConnection.***HTTP\_OK***) {  InputStream in = **new** BufferedInputStream(connection.getInputStream()); *// se prepara la cadena de entrada* BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(in));  *// El siguiente proceso de hace por que JSONObject necesita un string para  // concatenar lo que envio el servicio web de regreso qu es un JSON* String line;  **while** ((line = reader.readLine()) != **null**) {  result.append(line);  }  connection.disconnect();  **return** result.toString(); *// Regresa un JSON al método onPostExecute* }  }*// Atrapamos la posibilidad de que se genere un error* **catch** (MalformedURLException e){  e.printStackTrace();  }**catch** (IOException e){  e.printStackTrace();  }  **return** result.toString();  } **else if** (params[1]==**"3"**) *// Se ejecuta por el Boton de Insertar o Actualizar* {  **try** {  HttpURLConnection urlConn;   DataOutputStream printout;  DataInputStream input;  url = **new** URL(cadena);  *// Abrimos la conexión hacia el servicio web alojado en el servidor* urlConn = (HttpURLConnection) url.openConnection();  urlConn.setDoInput(**true**);  urlConn.setDoOutput(**true**);  urlConn.setUseCaches(**false**);  urlConn.setRequestProperty(**"Content-Type"**, **"application/json"**);  urlConn.setRequestProperty(**"Accept"**, **"application/json"**);  urlConn.connect();  *// Creando parametros que vamos a enviar* JSONObject jsonParam= **new** JSONObject();  jsonParam.put(**"nocontrol"**,params[2]);  jsonParam.put(**"carrera"**,params[3]);  jsonParam.put(**"nombre"**,params[4]);  jsonParam.put(**"telefono"**,params[5]);  *//Envio de parametros por el método post* OutputStream os = urlConn.getOutputStream();  BufferedWriter writer = **new** BufferedWriter(**new** OutputStreamWriter(os,**"UTF-8"**));  *// Escribe los datos a través de los métodos flush() y write()* writer.write(jsonParam.toString());  writer.flush();  writer.close();  **int** respuesta = urlConn.getResponseCode();   StringBuilder result = **new** StringBuilder();  *// Preguntamos si se pudo conectar al servidor con exito* **if** (respuesta == HttpURLConnection.***HTTP\_OK***)  {  *// El siguiente proceso de hace por que JSONObject necesita un string para  // concatenar lo que envio el servicio web de regreso qu es un JSON* String line;  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(urlConn.getInputStream()));  **while**((line=br.readLine()) != **null**)  {  result.append(line);  }  urlConn.disconnect();  devuelve = result.toString(); *// Regresa un JSON al método onPostExecute* }  }  **catch** (MalformedURLException e){  e.printStackTrace();  }**catch** (IOException e){  e.printStackTrace();  }**catch** (JSONException e){  e.printStackTrace();  }  **return** devuelve;  }**else if** (params[1]==**"5"**) *// Se ejecuta por el Boton de Borrar* {  **try** {  HttpURLConnection urlConn;   DataOutputStream printout;  DataInputStream input;  url = **new** URL(cadena);  *// Abrimos la conexión hacia el servicio web alojado en el servidor* urlConn = (HttpURLConnection) url.openConnection();  urlConn.setDoInput(**true**);  urlConn.setDoOutput(**true**);  urlConn.setUseCaches(**false**);  urlConn.setRequestProperty(**"Content-Type"**, **"application/json"**);  urlConn.setRequestProperty(**"Accept"**, **"application/json"**);  urlConn.connect();  *// Creando parametros que vamos a enviar* JSONObject jsonParam= **new** JSONObject();  jsonParam.put(**"nocontrol"**,params[2]);  *//Envio de parametros por el metodo post* OutputStream os = urlConn.getOutputStream();  BufferedWriter writer = **new** BufferedWriter(**new** OutputStreamWriter(os,**"UTF-8"**));  *// Escribe los datos a través de los métodos flush() y write()* writer.write(jsonParam.toString());  writer.flush();  writer.close();  **int** respuesta = urlConn.getResponseCode();   StringBuilder result = **new** StringBuilder();  *// Preguntamos si se pudo conectar al servidor con exito* **if** (respuesta == HttpURLConnection.***HTTP\_OK***)  {  *// El siguiente proceso de hace por que JSONObject necesita un string para  // concatenar lo que envio el servicio web de regreso qu es un JSON* String line;  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(urlConn.getInputStream()));  **while**((line=br.readLine()) != **null**)  {  result.append(line);  }  urlConn.disconnect();  devuelve = result.toString(); *// Regresa un JSON al método onPostExecute* }  }  **catch** (MalformedURLException e){  e.printStackTrace();  }**catch** (IOException e){  e.printStackTrace();  }**catch** (JSONException e){  e.printStackTrace();  }  **return** devuelve;  }  **return null**;  }  }  } |

**Tareas Asincronas (Hilos)**

Obtenido de: https://miguelangellv.wordpress.com/2011/01/31/creando-tareas-asincronas-en-android-con-asynctask/

En determinadas ocasiones, tenemos programas que tardan bastante rato en realizar una tarea en concreto y queremos que esta no bloquee la aplicación mientras se está ejecutando.

Para solventar esto normalmente se hace uso de los Threads (hilos). Para ello, creamos un hilo adicional para nuestra tarea y continuamos con el hilo normal para el resto de la aplicación.

El problema es que la interfaz gráfica (UI) de Android no permite llamadas desde otros hilos que no sea el suyo, así que si necesitamos realizar alguna actualización (aumentar barra de porcentaje, añadir información…) nuestra aplicación se cerrará.

Hay distintos métodos para poder acceder al hilo de la UI y actualizarla desde este (Handler, runOnUiThread..), pero yo explicaré el que me ha parecido más limpio, seguro y fácil, las AsyncTasks.

Extendiendo de AsyncTask (tarea asíncrona) podemos crear una clase que tenga entre 1 y 4 métodos según necesitemos.

Además, la clase AsyncTask dispone de tres tipos de parámetros distintos, que tendremos que especificar cuándo declaremos nuestra clase. Veamos un ejemplo:

|  |  |
| --- | --- |
|  | public class MiTarea extends AsyncTask<Params, Progress, Result> {  } |

Los tres parámetros son los siguientes:

* Params:  Datos que pasaremos al comenzar la tarea
* Progress: Parámetros que necesitaremos para actualizar la UI.
* Result: Dato que devolveremos una vez terminada la tarea.

En nuestro proyecto la clase AsyncTask es:

|  |  |
| --- | --- |
|  | **public class** ObtenerWebService **extends** AsyncTask<String, Void, String>{ |

Si alguno de los parámetros no lo necesitamos, podremos substituirlo por Void.

Los métodos que tendremos que escribir de nuestra clase son los siguientes, aunque solo es obligatorio el primero:

|  |  |
| --- | --- |
|  | protected Result doInBackground(Params... p) {  } |

Este método será el encargado de realizar la tarea en segundo plano. Como vemos, recibe un número cualquiera de parámetros del tipo Params, así que debemos tratar a p como un array.

Este método se ejecuta en otro hilo, por lo que no podremos modificar la UI desde él. Para ello, usaremos los tres métodos siguientes.

|  |  |
| --- | --- |
|  | protected void onPreExecute() {  } |

Este método se ejecutará antes de doInBackground, por lo que podremos modificar la interfaz para indicar el comienzo de la tarea (colocar un cargando, desactivar botones…).

|  |  |
| --- | --- |
|  | protected void onProgressUpdate (Progress... values){  } |

Este método permitirá actualizar la interfaz mientras se ejecuta la tarea asíncrona. Para ello, desde doInBackground deberemos llamar a ﻿publishProgress y pasarle los parámetros oportunos.

|  |  |
| --- | --- |
|  | protected void onPostExecute (Result result){  } |

Este último método, como habréis supuesto, se ejecuta tras terminar doInBackground y recibe como parámetro lo que este devuelva.

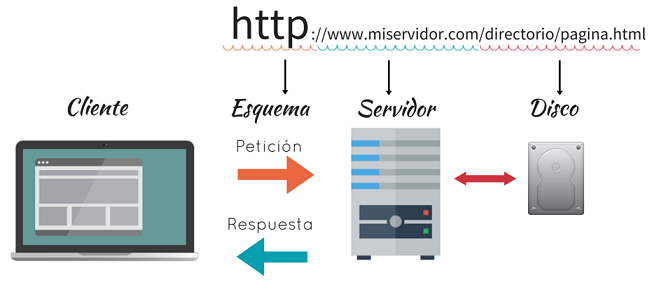
# **HTTP En Android con el Cliente HttpURLConnection**

Obtenido de: http://www.hermosaprogramacion.com/2015/01/android-httpurlconnection/

Las conexiones que haremos desde nuestras aplicaciones Android hacia los servidores web siguen un estándar internacional llamado *Hipertext Transfer Protocol* HTTP. Este protocolo consiste en reglas sencillas de transferencia de recursos o archivos entre equipos interconectados a una red.

Al equipo que hace la petición para enviar u obtener datos se le llama *Cliente* y al que contiene el recurso o el espacio para almacenar es llamado *Servidor*.

La comunicación se establece a través de una petición de envío, la cual contiene los datos del cliente, como el sistema operativo que usa, el navegador web desde donde se hace la petición, la ubicación del archivo solicitado (URL), etc.

[](http://www.hermosaprogramacion.com/wp-content/uploads/2015/01/http-protocolo-peticion.png)

Una petición puede tener múltiples objetivos dependiendo del método que se elija. Los tipos de peticiones más comunes son el **Retorno de datos** y la **Publicación de datos**. Técnicamente se les conoce como los métodos GET y POST.

La búsqueda de una página web a través de la URL es un buen ejemplo de una petición GET, donde el cliente especifica la URL y el servidor retorna en la información HTML necesaria para que el navegador realice su respectivo parsing.

El ejemplo más popular del método POST se refleja en el envío de información desde un formulario hacia la base de datos del servidor. Aquí hacemos lo contrario, dictaminamos los datos y el servidor los recibe para almacenarlos y darles persistencia.

Cada vez que entras a hermosa programación desde tu navegador la comunicación HTTP sería similar a esta:

GET /index.html

Host: www.hermosaprogramación.com

User Agent: Mozilla/4.0 (Compatible; MSIE 7.0; Windows NT 6.0)

Accept: \*/\*

**Importante**

Debes dar permiso a tu aplicación Android para conectarse a protocolos de internet, para lo cual agrega el siguiente permiso en tu archivo AndroidManifest.xml:

<**uses-permission  
 android:name="android.permission.INTERNET"**>  
</**uses-permission**>

## Enseguida se muestra la imagen de como que el archivo de AndroidManifest.xml, remarcando en azul el código que se agrego:

## 

## Usar el cliente HttpURLConnection

La clase HttpUrlConnection del paquete java.net.\* permite a nuestros dispositivos android asumir las características de un cliente HTTP ligero. Su funcionamiento está condicionado a las versiones superiores a *Gingerbread*. Para versiones anteriores debes usar el cliente HttpClient de Apache.

### *Abrir la conexión hacia el servidor*

El primer paso para iniciar la comunicación es abrir la conexión hacia el recurso alojado en el servidor. Para ello se usa el método openConnection() de la clase URL. El resultado que se obtenga debe ser casteado a HttpUrlConnection para que el cliente sea instanciado:

url = **new** URL(cadena);  
*// Abrimos la conexión hacia el servicio web alojado en el servidor*HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
connection.setRequestProperty(**"User-Agent"**, **"Mozilla/5.0"** +  
 **" (Linux; Android 1.5; es-ES) Ejemplo HTTP"**);

### *Obtener datos con el método GET*

Si deseas descargar datos desde la URL especificada simplemente usas el método getIntpuStream() para obtener el flujo de datos asociado al recurso que se encuentra en esa dirección:

url = **new** URL(cadena);  
*// Abrimos la conexión hacia el servicio web alojado en el servidor*HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
connection.setRequestProperty(**"User-Agent"**, **"Mozilla/5.0"** +  
 **" (Linux; Android 1.5; es-ES) Ejemplo HTTP"**);  
**int** respuesta = connection.getResponseCode();  
*// Preguntamos si se pudo conectar al servidor con exito***if** (respuesta == HttpURLConnection.***HTTP\_OK***) {  
 InputStream in = **new** BufferedInputStream(connection.getInputStream()); *// se prepara la cadena de entrada* BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(in));  
 *// El siguiente proceso de hace por que JSONObject necesita un string para  
 // concatenar lo que envio el servicio web de regreso qu es un JSON* String line;  
 **while** ((line = reader.readLine()) != **null**) {  
 result.append(line);  
 }  
 connection.disconnect();  
 **return** result.toString(); *// Regresa un JSON al método onPostExecute*}

Es importante que al finalizar tus operaciones de conexión liberes la memoria asociada a la instancia de la conexión realizada. Para ello usa el método disconnect(), el cual pone a disposición de nuevo una futura reconexión.

Recuerda que un tipo InputStream debe ser decodificado para interpretar su contenido, ya sea texto plano, imagen, *JSON*, audio, etc. Dependiendo del objetivo así mismo debes usar los métodos y técnicas correspondientes.

### *Postear información con el método POST*

Si deseas publicar información en un servidor debes abrir la conexión al igual que con GET. Luego se indica a la conexión que se permite el envío de datos hacia el servidor con el método setDoOutput().

Seguidamente se declaran los datos que se enviarán al destino, para los cuales debes declarar el tamaño que ocuparán para ser transmitidos por el flujo.

Si su tamaño es fijo, entonces usa el método setFixedLengthStreamingMode(), quién recibe como parámetro la cantidad de bytes.

Si el tamaño es incierto (normalmente esta situación se da en transmisiones Streaming), entonces usa setChunkedStreamingMode().

**try** {  
 HttpURLConnection urlConn;  
  
 DataOutputStream printout;  
 DataInputStream input;  
 url = **new** URL(cadena);  
 *// Abrimos la conexión hacia el servicio web alojado en el servidor* urlConn = (HttpURLConnection) url.openConnection();  
 urlConn.setDoInput(**true**);  
 urlConn.setDoOutput(**true**);  
 urlConn.setUseCaches(**false**);  
 urlConn.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 urlConn.setRequestProperty(**"Accept"**, **"application/json"**);  
 urlConn.connect();  
 *// Creando parametros que vamos a enviar* JSONObject jsonParam= **new** JSONObject();  
 jsonParam.put(**"nocontrol"**,params[2]);  
 jsonParam.put(**"carrera"**,params[3]);  
 jsonParam.put(**"nombre"**,params[4]);  
 jsonParam.put(**"telefono"**,params[5]);  
 *//Envio de parametros por el método post* OutputStream os = urlConn.getOutputStream();  
 BufferedWriter writer = **new** BufferedWriter(**new** OutputStreamWriter(os,**"UTF-8"**));  
 *// Escribe los datos a través de los métodos flush() y write()* writer.write(jsonParam.toString());  
 writer.flush();  
 writer.close();  
 **int** respuesta = urlConn.getResponseCode();  
  
 StringBuilder result = **new** StringBuilder();  
 *// Preguntamos si se pudo conectar al servidor con exito* **if** (respuesta == HttpURLConnection.***HTTP\_OK***)  
 {  
 *// El siguiente proceso de hace por que JSONObject necesita un string para  
 // concatenar lo que envio el servicio web de regreso qu es un JSON* String line;  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(urlConn.getInputStream()));  
 **while**((line=br.readLine()) != **null**)  
 {  
 result.append(line);  
 }  
 urlConn.disconnect();  
 devuelve = result.toString(); *// Regresa un JSON al método onPostExecute* }  
}  
**catch** (MalformedURLException e){  
 e.printStackTrace();  
}**catch** (IOException e){  
 e.printStackTrace();  
}**catch** (JSONException e){  
 e.printStackTrace();  
}

}

Finalmente se obtiene acceso al sistema de archivos del servidor con getOutputStream(), el cual retorna en un flujo de datos en donde escribirás los datos de los comentarios a través de los métodos flush() y write().