Introducción a ArcGIS Runtime for Android

Antonio Remírez Remírez

Introducción a ArcGIS Runtime for Android

por Antonio Remírez Remírez

Introducción a ArcGIS Runtime for Android

por Antonio Remírez Remírez

Acerca del autor:

Experto certificado por esri en desarrollo web y administración de sistemas GIS empresariales con experiencia en formación en ESRI España:

www.esri.com/certificationVerification

Web application developer: JKWBESGCC1EEQT6R

Enterprise administration: 9CRLFWS11EF12QFC

Prefacio

El nuevo Runtime SDK para Android facilita el poder del GIS desde dispositivos con el sistema operativo libre Android. Mediante el entorno de desarrollo Eclipse y el lenguaje de programación Java, el curso le acercará la potencia de las nuevas tecnologías móviles.

Para la realización del curso es importante tener nociones de programación orientada a objetos y en especial del lenguaje de programación Java. No es un requisito indispensable tener conocimientos de desarrollo para la plataforma Android.

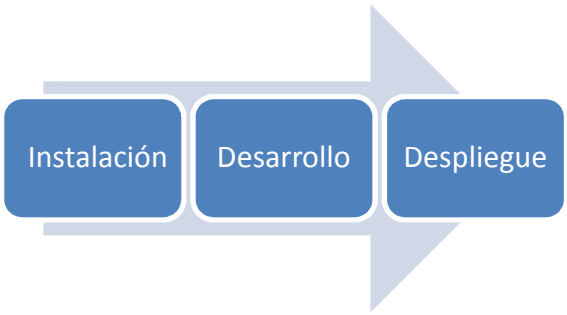
El formato del curso es secuencial, haciendo en los primeros temas hincapié en las bases del desarrollo Android, el entorno de desarrollo Eclipse y todo lo relacionado con la tecnología que es preciso conocer antes de comenzar a utilizar el Runtime SDK de ArcGIS para Android. En los siguientes temas, se profundiza en la programación más pura realizando ejemplos típicos de aplicaciones GIS sobre esta plataforma. Al final del curso, se estudia brevemente cómo realizar el *release* de las aplicaciones así como su despliegue final en el usuario.

Índice

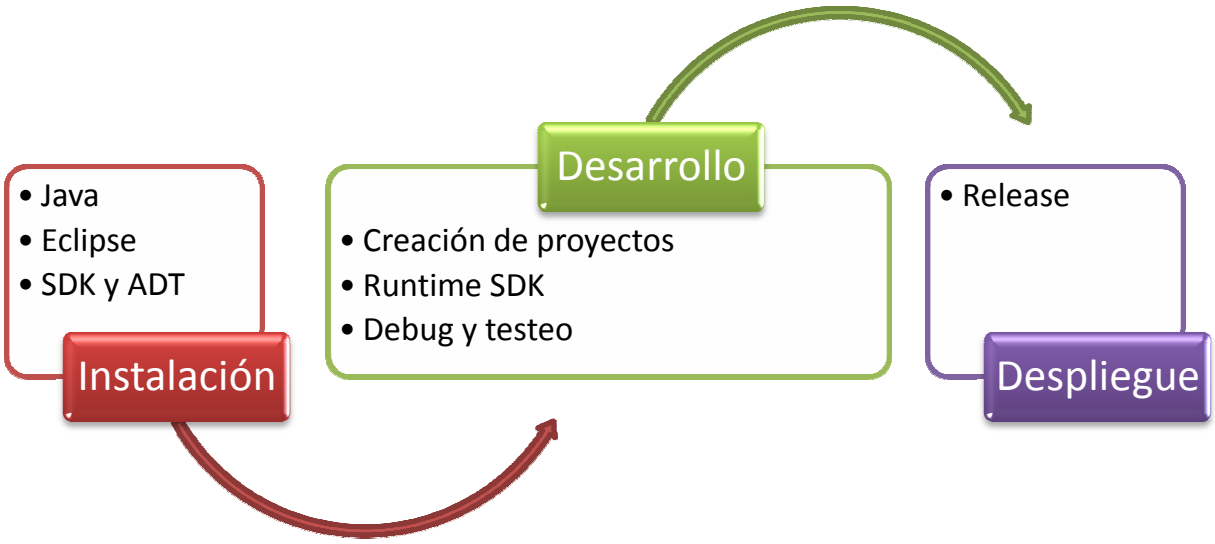
1.- Introducción al desarrollo Android

1.1. - Desarrollo Android

Como todo desarrollo de software, desarrollar para la plataforma Android tiene unas fases bien diferenciadas:



A lo largo del curso, se va a profundizar en cada una de las tres fases, haciendo hincapié en los en la fase de instalación del entorno y del desarrollo de proyectos:



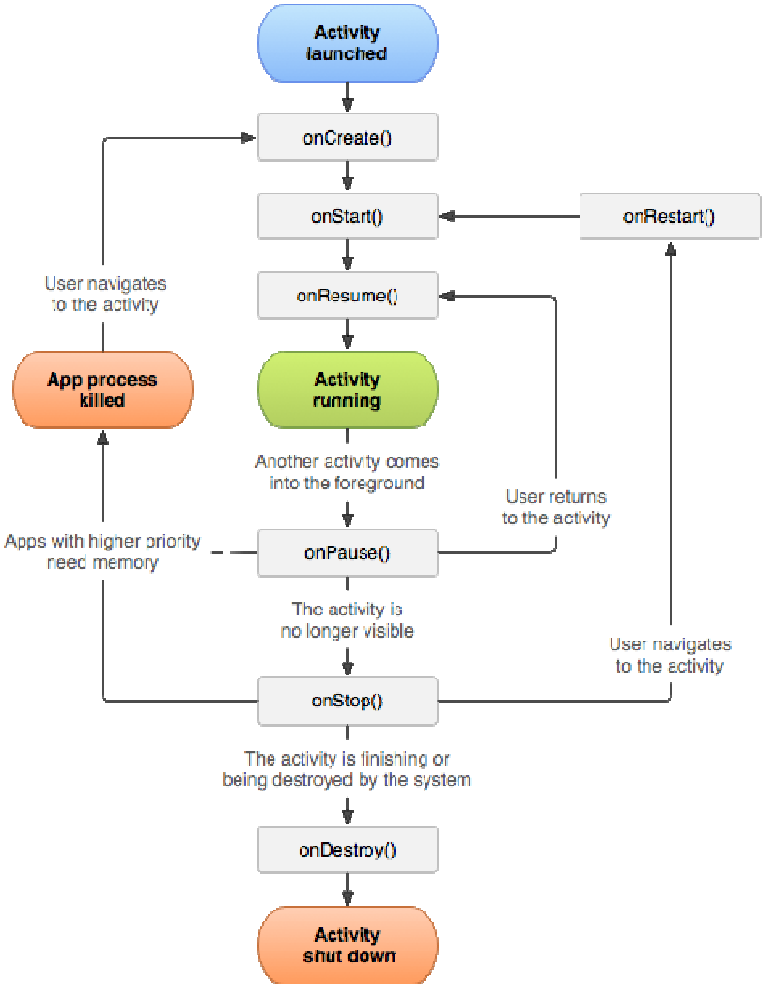
En los primeros temas se aborda cómo debe realizarse la instalación o cómo afecta el entorno de desarrollo al proyecto. En los siguientes, se profundiza en cómo crear proyectos con el Runtime SDK y se desarrollan prototipos de proyectos GIS explotando su funcionalidad. Por último, se muestra cómo realizar un *release* del proyecto y cómo es su despliegue.

1.2. – Activities

Podría decirse que el desarrollo de una aplicación Android, se lleva a cabo mediante el desarrollo de las sucesivas pantallas que componen nuestra aplicación. Imagine, por ejemplo, que abre una aplicación en un dispositivo Android y en su pantalla aparece una imagen y un botón. Al pinchar en el botón la aplicación le lleva a otra pantalla en la cual aparece un mensaje de texto. Para realizar esa aplicación, el programador ha tenido que desarrollar esas dos ventanas. Cada una de estas interfaces será lo que en Android se conoce como *Activity*.

Normalmente, la visión que se tiene de una *Activity* es precisamente la de una pantalla completa con la que interactúa el usuario pero es posible que se trate de una ventana flotante encima de otra *Activity* o incluso una *Activity* dentro de otra *Activity*.

Un desarrollador debe comprender que una *Activity* pasará por varios estados a lo largo del ciclo de vida de la aplicación (desde que un usuario lanza la aplicación hasta que la cierra) y que un número indefinido de estas *Activities* serán coexistirán en el dispositivo durante este ciclo. Tanto es así, que resulta importante conocer su propio ciclo de existencia:



Ciclo de vida definido en el API Reference de Android:
http://developer.android.com/reference/android/app/Activity.html

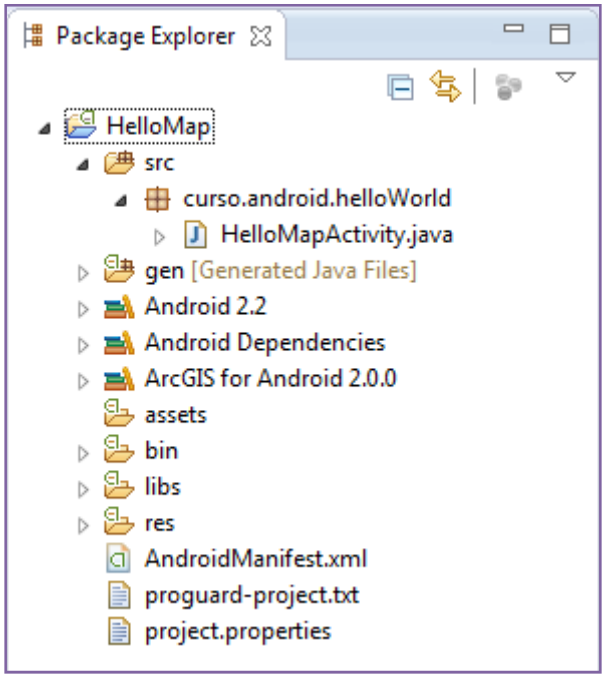
Los métodos que normalmente deben ser implementados en cada *Activity* son:

- **onCreate()**: se llama cuando la *Activity* se llama por primera vez y debe ser creada. Si la aplicación requiere de conectar con unos datos o con el estado de la aplicación la última vez que fue abierta, debe realizarse aquí. Siempre se sigue con un onStart().
- **onRestart()**: la actividad ha sido detenida y debe relanzase. De nuevo, siempre se sigue con un onStart().
- **onStart()**:se llama cuando la actividad se hace visible para el usuario.
- **onResume()**: se llama cuando la aplicación se ha cargado y está preparada para que el usuario interactúe con ella. Siempre se continúa con un onPause().
- **onPause()**: el sistema requiere de otra *Activity* y antes se ejecuta este método. Suele utilizarse para salvar cambios y detener elementos que puedan consumir CPU. Se recomienda que la implementación sea lo más ligera posible debido a que ninguna otra *Activity* puede realizar su correspondiente onResume() si hay un previo onPause() sin finalizar. Además, siempre irá seguido de su onResume() si la *Activity* vuelve a ser visible al usuario o de su onStop() si se hace invisible.
- **onStop()**: se llama cuando ya no es visible para el usuario. Esto puede deberse a que una nueva *Activity* ha comenzado o ha vuelto a ser visible al usuario o la actual ha sido destruida.
- **onDestroy()**: llamada final de la *Activity*, después es totalmente destruida. Puede suceder debido a que el sistema necesite liberar memoria o porque explícitamente se llama a este método.

1.3. – Arquitectura del proyecto

Es importante saber que el proyecto Android se convertirá en un fichero *.apk (Application PacKage file)*, que es una variante de los clásicos ficheros *.jar* de Java. Este es el tipo de fichero es el empaquetado necesario para la distribución de aplicaciones en la plataforma Android.

La arquitectura de los proyectos Android que hacen esto posible es la siguiente:



El código fuente

El código fuente de la aplicación, se sitúa, como en los proyectos Java, en la carpeta **src**, mientras que el código autogenerado se encuentra en la carpeta **gen**. Como puede verse en la imagen, en dichas carpetas encontraremos ficheros de código escritos en Java y con extensión *.java*. Nuestros ficheros Java se deben organizar en paquetes lo cual crea una estructura de clases que se refleja en nuestro disco como una estructura de carpetas donde se guardan los ficheros *.java*.

Al crear un proyecto, se verá más adelante, es necesario establecer junto al nombre de la aplicación, el paquete que se quiere crear. En la carpeta **gen** se crean automáticamente para el mismo paquete dos ficheros *.java* adicionales: R y BuildConfig. En el fichero R.java, se generan automáticamente por el compilador, identificadores para todos los recursos del proyecto. De este modo, es posible acceder mediante código a todos los recursos ya que quedan programados bajo clases estáticas.

Recursos

Un recurso es cualquier elemento del proyecto que no forma parte del código fuente del mismo. Habitualmente imágenes, audio, video, animaciones,…

Pero Android va más allá y trata de externalizar al máximo y contempla como recursos cadenas de texto, menús y algo muy interesante, el propio diseño de cada *Activity*, como se verá más adelante. De este modo se consigue que cualquier cambio en un recurso se haga de forma más eficiente que si estuvieran embebidos en el código fuente.

Todos estos recursos pueden encontrarse en la carpeta **res** del proyecto.

Es posible distinguir al menos 3 tipos de recursos: simples, imágenes y animaciones, y *layouts*:

Recursos simples:

También conocidos como **values,** se encuentran precisamente en dicha carpeta. Tiene como particularidad que se definen en ficheros XML y que el ADT proporciona un editor sencillo a base de diálogos.

Hay diferentes tipos:

- Cadenas de texto.
- Colores. Definidos en hexadecimal.
- Valores *booleanos*.
- Valores enteros.
- Identificadores únicos para todo el proyecto.
- *Arrays*.
- Estilos visuales.

Imágenes y animaciones

Los podemos encontrar en las carpetas **drawables** dentro de **res** si se trata de imágenes estáticas, **anim** para ficheros XML convertibles en animaciones, **raw** para ficheros de tipo *media* como *mp3* o ficheros *Ogg*.

Layouts

Se trata de ficheros en formato XML que se encuentran dentro de la carpeta **layout** y mediante los cuales se definen los elementos visuales y su disposición en la pantalla. Estos *layouts* son aplicados a las *Activities* separando de este modo el diseño de la funcionalidad.

De nuevo, el ADT nos permite su definición mediante ventanas de diálogo en lugar de la edición directa de los ficheros XML.

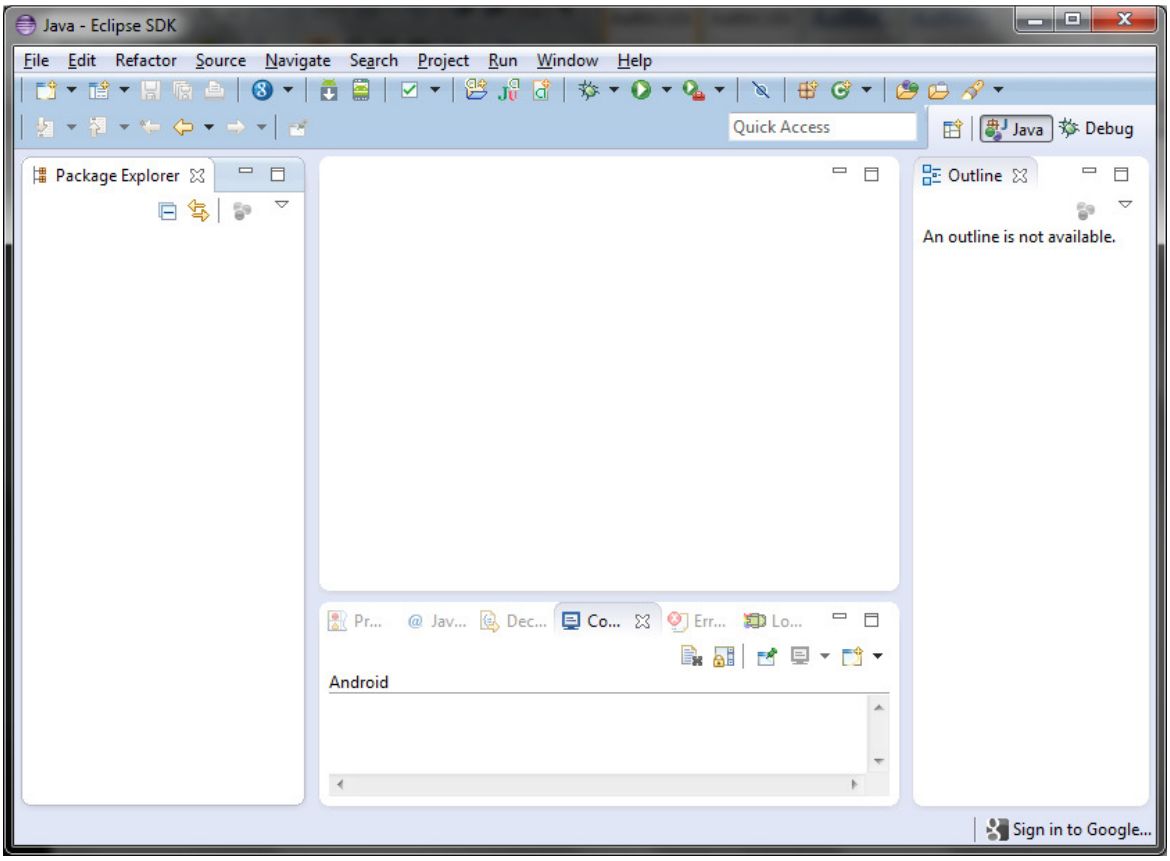
Más adelante se verá en profundidad la relación entre estos ficheros XML y las Activities pero conviene comprender que en Android se separa siempre presentación y funcionalidad, quedando la primera en los *layouts* en forma de ficheros XML y la segunda en *Activities* en forma de ficheros *.java*.

1.4. – Ejercicio: Creando un proyecto Android

Paso 1

Haga doble clic en el acceso directo a Eclipse del escritorio y espere a que arranque el programa.

El software Eclipse tiene una distribución de paneles clásica como esta:

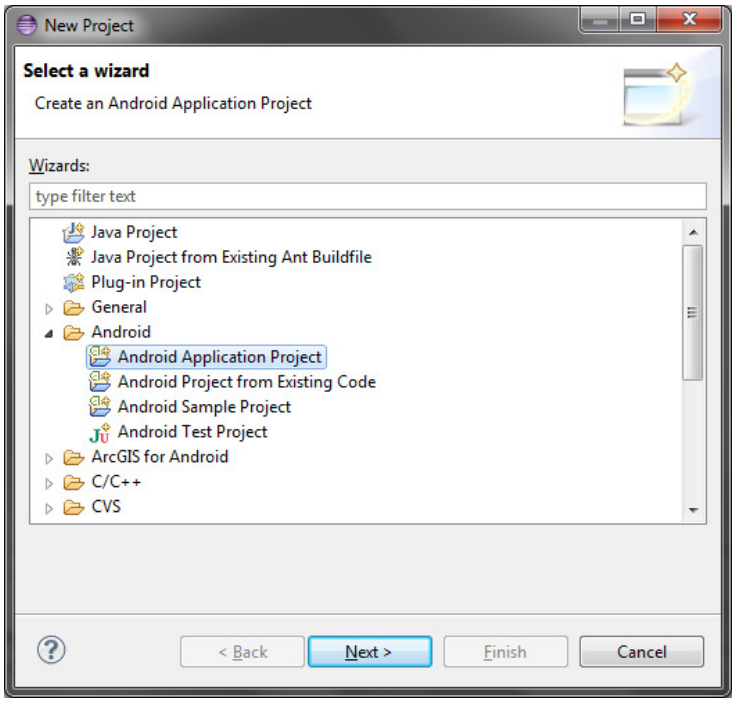


Varios de los paneles contienen valiosa información:

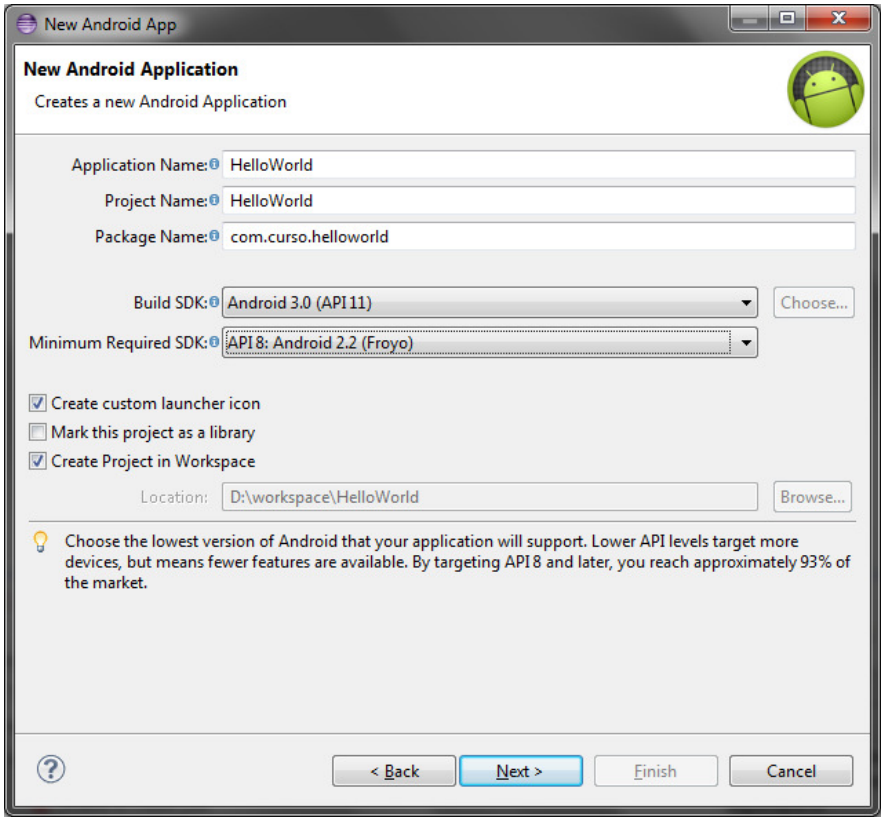
- **Package** Explorer: en esta sección se verán los ficheros de nuestro proyecto (ficheros Java, recursos, *layouts*,…). Normalmente, se localiza a la izquierda.
- **Outline**: en este panel se muestran los elementos del fichero de código que este abierto en cada momento de forma jerárquica. Normalmente, se localiza a la izquierda.
- **Console**: en este panel se muestran los mensajes enviados a la consola mediante código Java. Normalmente, se localiza en una pestaña del panel inferior.
- **Error Log**: el compilador muestra en este panel los diferentes errores o advertencias que localiza en el código.

Paso 2

Haga clic en el menú "**File > Project…**" y accederá al *wizard* de creación de proyectos de Eclipse:



Seleccione "**Android Application Project**" y haga clic en "**Next**". En esta ventana además del nombre del proyecto y la aplicación (normalmente, el mismo), hay varios conceptos importantes a tener en cuenta:



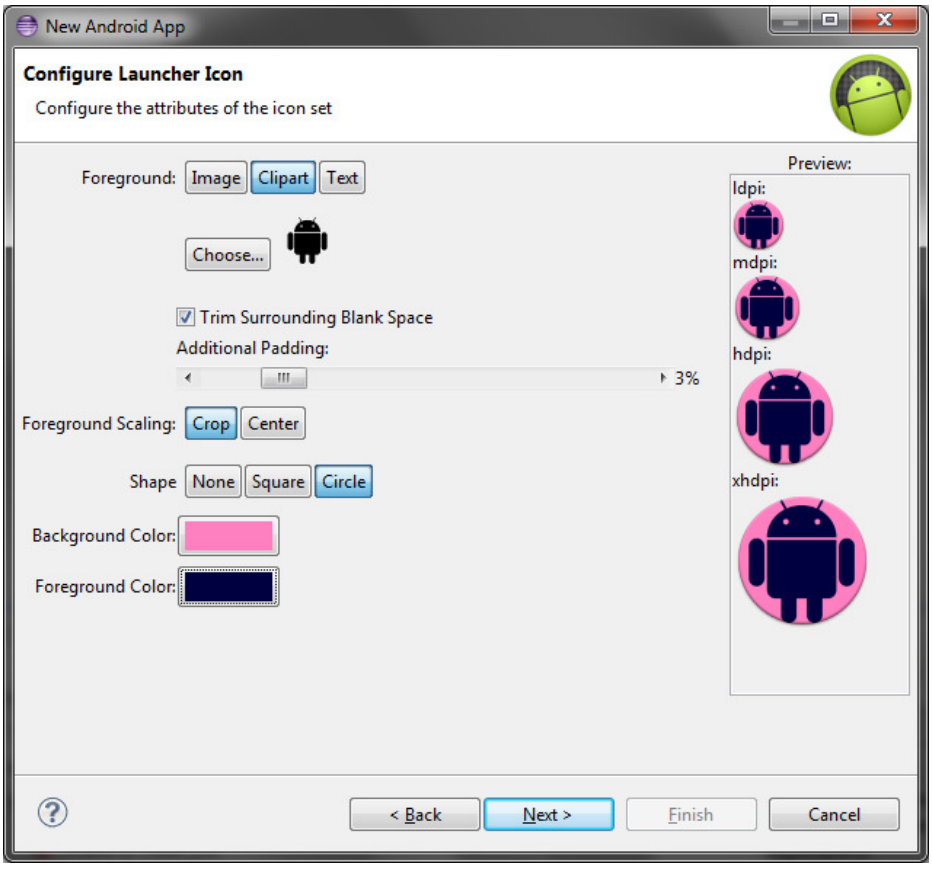
El primero es el nombre del paquete en el cual queremos incluir nuestro código fuente. En este caso hemos decidido que nuestro paquete sea “com.curso.helloworld”.

Las siguientes dos opciones a cumplimentar en el formulario son críticas, se trata del **Build SDK** o SDK de desarrollo y el **Minimum required SDK** o el mínimo SKD requerido. El Build SDK indica al compilador para qué versión de Android está programada la aplicación que estamos desarrollando, mientras que el Minimum required SDK indica qué versión de Android es la mínima capaz de ejecutar la misma. En este caso, se ha establecido como valor mínimo, Android 2.2 (Froyo) y como Build SKD, Android 3.0 (Honeycomb). De forma sencilla, podría decirse que la aplicación podrá ejecutarse en dispositivos móviles con una versión Android entre 2.2 y 3.0.

Seleccione el resto de parámetros por defecto y continúe.

Paso 3

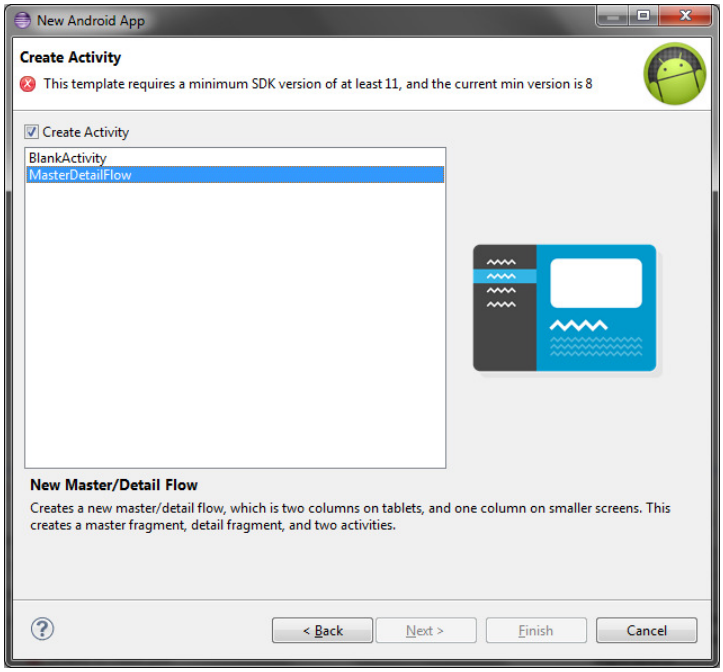
En esta ventana puede seleccionar el icono de su aplicación para los diferentes tamaños y resolución de pantalla. Puede seleccionar una imagen de su disco, una de las preestablecidas o crear una en base a un texto introducido.



Seleccione un icono a su gusto y continúe.

Paso 4

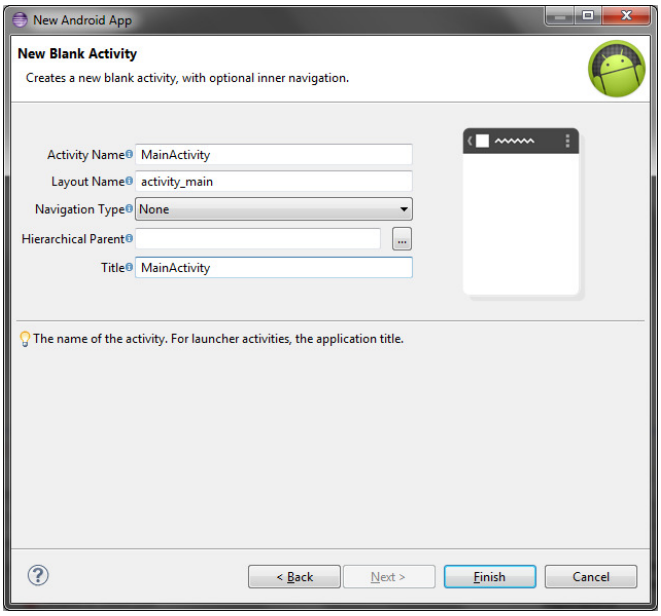
En esta ventana es posible crear una *Activity* por defecto que será la primera en lanzarse en nuestra aplicación. Puede seleccionar además una plantilla para esta *Activity* pero es necesario tener en cuenta que hay plantillas que requieren de una versión de Android elevada. En este caso hemos establecido la versión mínima para nuestra aplicación Android 2.2 (Froyo) y una de las plantillas del ADT, MasterDetailFlow, requiere una versión superior y dará un error:



Seleccione la plantilla por defecto, BlankActivity, y continúe.

Paso 5

En esta última ventana puede seleccionar los valores oportunos de la *Activity* a crear:

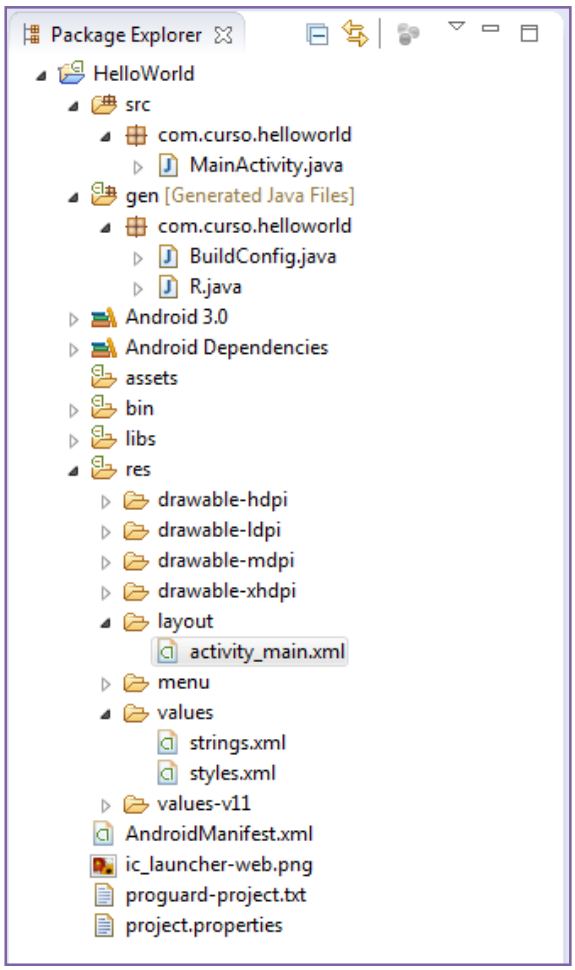


Deje los valores por defecto y pulse "**Finish**".

Paso 6

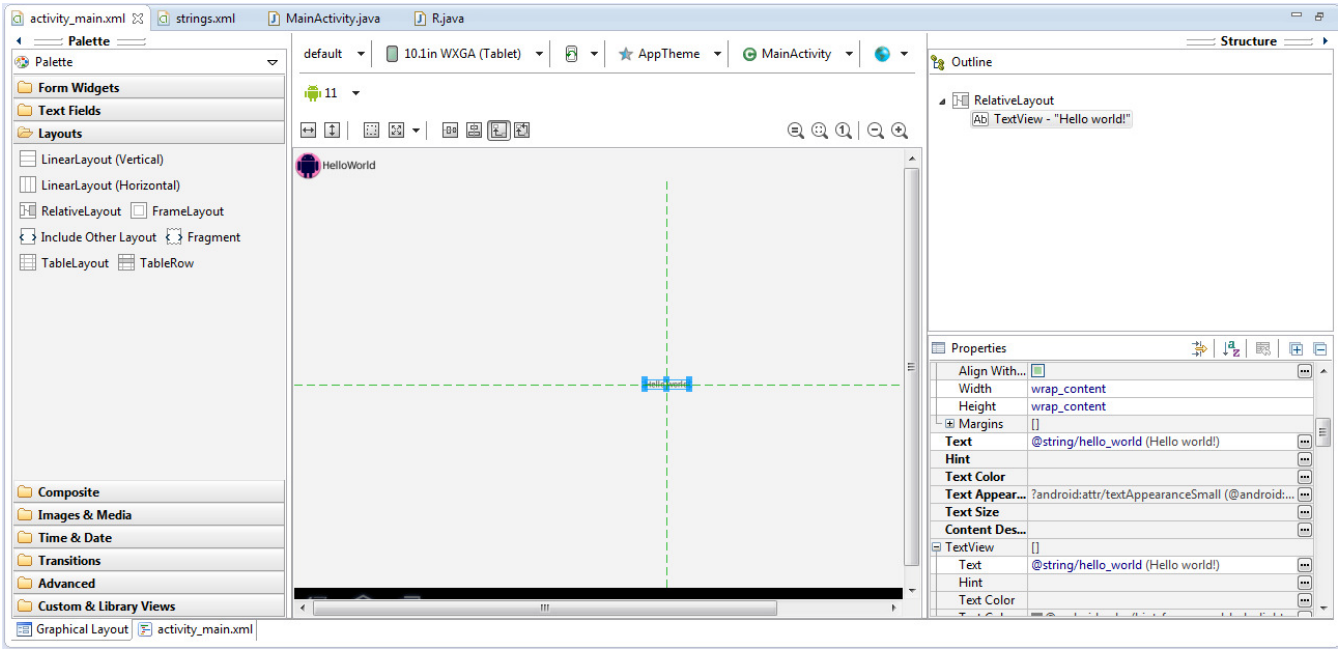
El proyecto ha sido creado así como la principal *Activity* del programa.

En el panel de exploración debería ver una jerarquía como la descrita anteriormente y que coincida con esta:



Compruebe que todos los componentes del proyecto han sido creados correctamente.

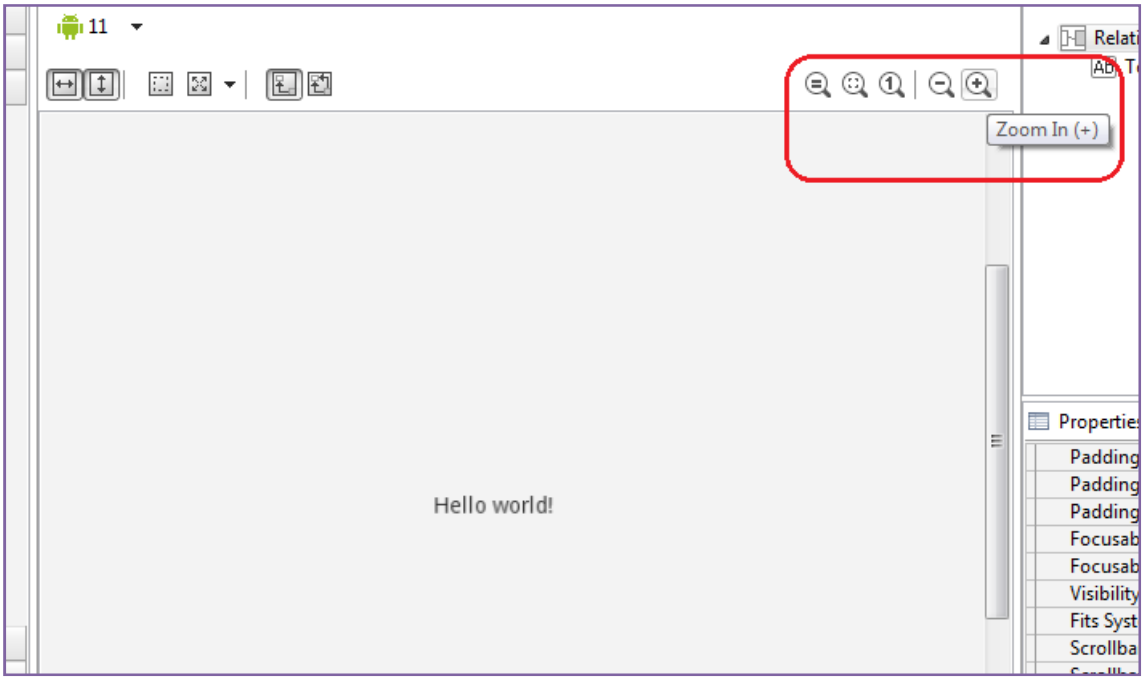
Una de las mayores aportaciones del ADT a Eclipse es la posibilidad de editar interfaces gráficas de forma sencilla mediante una interfaz basada en menús y paneles de atributos. Si no ha sido abierto automáticamente abra el *layout* cuyo nombre es activity_main.xml:



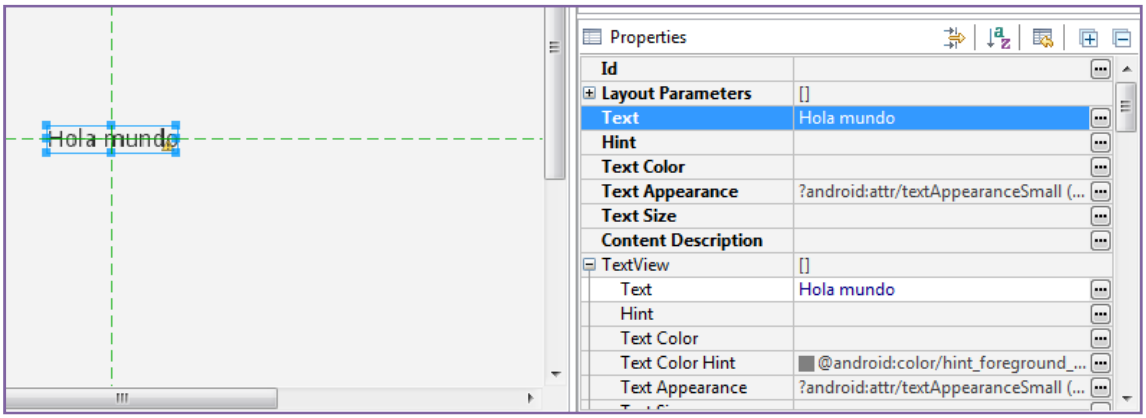
Por defecto, se crea para esta *Activity* un "RelativeLayout" y un "TextView" que se mostrará en el centro de la aplicación.

En el menú vertical de la izquierda, ordenados en carpetas, están todos los componentes gráficos que se pueden añadir a las *Activities*. En el centro, es posible retocar, ordenar y disponer gráficamente los elementos de forma sencilla. Y en la parte derecha, se accede a las propiedades de los elementos seleccionados en la interfaz central.

Seleccione el elemento de texto creado automáticamente. Si no lo ve correctamente, puede acercarse y moverse por la interfaz con el menú de navegación:

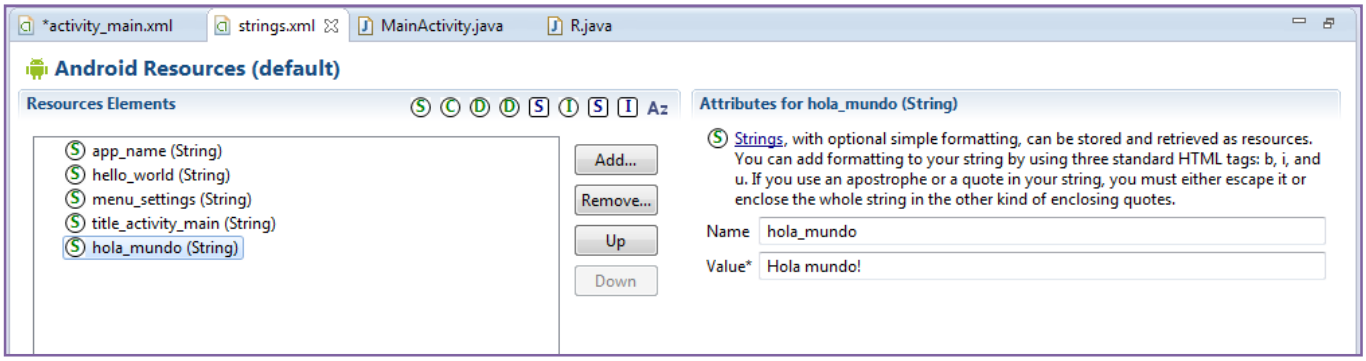


Seleccione el texto pinchando con el ratón (aparecerán dos líneas verdes) y compruebe cómo el menú de la derecha se actualiza para mostrar las propiedades del "TextView". Busque la propiedad "Text" y compruebe que su valor es: **@string/hello_world.** Es posible modificar este valor, otorgándole el texto que se desee, por ejemplo "Hola mundo":



Esta forma de establecer valores no la más óptima. Android permite externalizar valores de propiedades como cadenas de texto de forma sencilla en ficheros XML de recursos. Por defecto, en un proyecto Android se crea dentro de **"res > values**" un fichero strings.xml en el cual se localizan cadenas de texto reutilizables desde cualquier punto del proyecto.

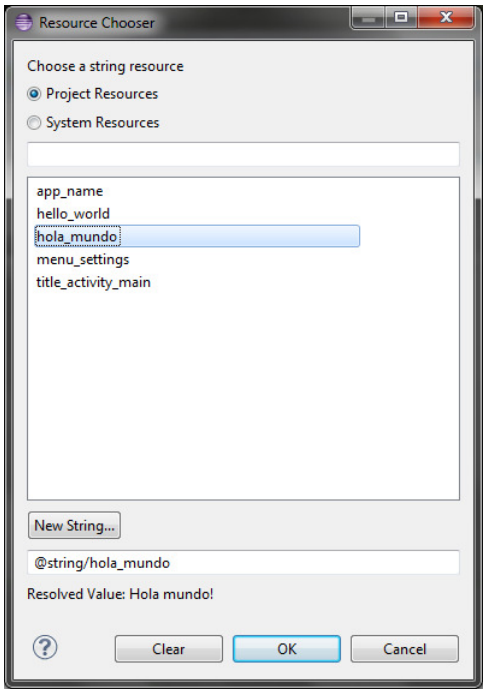
Abra el fichero strings.xml y un editor personalizado del ADT le permitirá editarlo de forma sencilla:



Es posible editar directamente el fichero como texto plano pero el uso del editor facilita la tarea. Pulse el botón "Add" y cree un nuevo recurso de tipo "String". Después, establezca los valores "Name" y "Value" como en el gráfico.

Salve los cambios realizados y vuelva a la *Activity* principal.

Seleccione de nuevo el elemento de texto y localice su propiedad "Text". Ahora, en lugar de introducir un texto de forma manual, pulse sobre el botón que se encuentra a la izquierda (tres puntos negros) y accederá al menú de selección de recursos:



Aparecerán todos los recursos disponibles en el proyecto, además de poder obtener los del sistema, entre los cuales estará el recurso recientemente creado “hola_mundo”. Selecciónelo y pulse “OK”.

El resultado será que el texto obtendrá su valor del fichero de recursos. De este modo es posible centralizar cadenas de texto reutilizables en diferentes partes del proyecto de forma sencilla.

En este punto, la aplicación podría ser ejecutada en un dispositivo con un sistema operativo Android entre las versiones 2.2 y 3.0 y mostraría una pantalla en la cual aparecería el mensaje “HolaMundo”. Más adelante se comprobará su ejecución:

2.- Introducción a ArcGIS Runtime SDK

2.1. – Runtime SDK

El Runtime SDK de ArcGIS para Android permite la ejecución de aplicaciones GIS de forma nativa en sistemas operativos Android 2.2 y superiores.

El Runtime es proporcionado en forma de *plugin* para Eclipse de tal forma que el desarrollo de aplicaciones se lleva a cabo de forma sencilla e integrada con el entorno de desarrollo más extendido para la plataforma Android. Este tipo de desarrollo permite:

- Utilizar mapas de ArcGIS Online y/o ArcGIS Server
- Ejecutar tareas avanzadas de geoprocesamiento
- Realizar consultas SQL sobre los datos
- Mostrar información en *popups*
- Integrar la localización GPS en su aplicación

En resumen, podría decirse que los principales recursos del Runtime son:

- **Mapas y capas**: tanto en su forma dinámica como cacheada. Incluso es posible incluir mapas base locales.
- **Capas gráficas**: facilitan que los usuarios realicen sus propios gráficos y también mostrar información en *popups*.
- **Tareas**: consultas, identificar elementos, localización de direcciones, geoprocesamientos,…

Además, es posible accede a una completa API Reference al estilo Javadoc desdes la web:

http://resources.arcgis.com/en/help/android-sdk/api/index.html

2.2. – Requisitos del sistema

El Runtime SDK necesita de ciertos requisitos en el entorno de desarrollo y el de ejecución:

Sistema operativo

El desarrollo de aplicaciones Android está soportado en los siguientes sistemas operativos:

- Windows XP (32bit), Vista (32- or 64-bit), y Windows 7 (32bit y 64bit)
- Mac OS X 10.5.8 o superior (sólo x86)
- Linux (Ubuntu Linux, Lucid Lynx)

Es posible encontrar una relación de sistemas operativos soportados en:
http://developer.android.com/sdk/index.html

IDE: Eclipse

El único entorno de desarrollo soportado por Esri y se necesita para cualquier desarrollo con el API desarrollada para ArcGIS. Además, para establecer este entorno desarrollo correctamente es necesario tener en cuenta ciertos pasos:

- Es necesario Java Development Kit (JDK) 6 o superior. No es suficiente para el desarrollo con contar con Java Runtime Environment (JRE).
- El entorno de desarrollo Eclipse a partir de la versión 3.6.2 (Helios).
- El plugin JDT de Eclipse, normalmente incluido en las versiones más recientes de Eclipse.
- Java Development Kit (JDK) 6 is required. A Java runtime environment (JRE) is not enough for development.
- El SDK de Android así como el *plugin* para Eclipse Android Development Tools (ADT).
- El Runtime SDK de ArcGIS para Android.

Algunos enlaces de interés:

- Java JDE: http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Eclipse downloads: http://www.eclipse.org/downloads/
- Android SDK: http://developer.android.com/sdk/index.html
- Android ADT: http://developer.android.com/tools/sdk/eclipse-adt.html
- ArcGIS Runtime SDK: http://www.esri.com/apps/products/download/index.cfm?fuseaction=download.all#ArcGIS_Runtime_SDK_for_Android

Plataformas Android

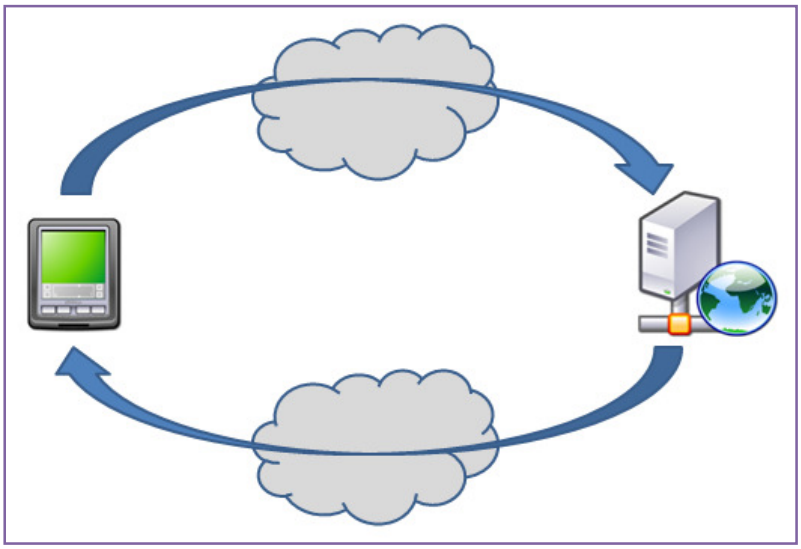
Esri soporta las plataformas Android 2.2 (API 8) y superiores. Por lo tanto, es necesario durante la creación del proyecto, establecer como SDK mínimo al menos la versión Android 2.2.

A partir de la versión 4.0.3 de Android (API 15), Esri añadió el uso de OpenGL ES 2.0 lo cual permite una aceleración superior en el renderizado de mapas y el consumo de menos batería en el dispositivo. Además, esto permite el uso de emuladores durante el desarrollo.

2.3. – Cliente-Servidor

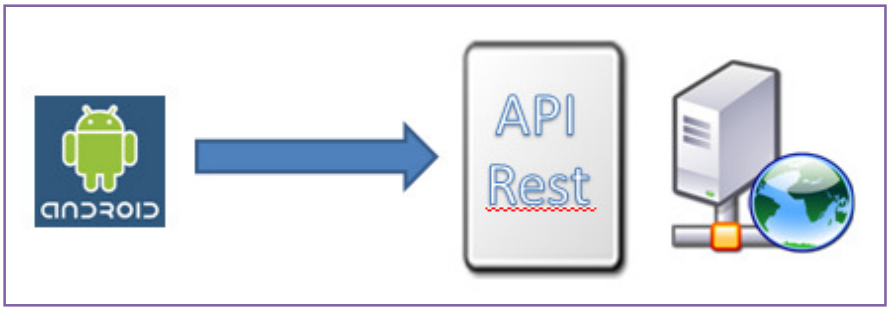
El Runtime SDK de ArcGIS para Android permite el consumo de servicios web de mapas alojados es ArcGIS Online, ArcGIS Server e incluso Bing Maps, pero también es posible alojar mapas cacheados en el propio dispositivo móvil.

Habitualmente, el GIS se basa en el paradigma Cliente-Servidor. En esta arquitectura, los datos, en este caso, los mapas, se encuentran alojados en un servidor y el dispositivo móvil, a través del Runtime SDK, realiza constantes peticiones al servidor para obtener la información solicitada por el usuario en cada momento. Las tareas de procesamiento pesado, geoprocesamiento, son realizadas de igual modo, normalmente en modo asíncrono.



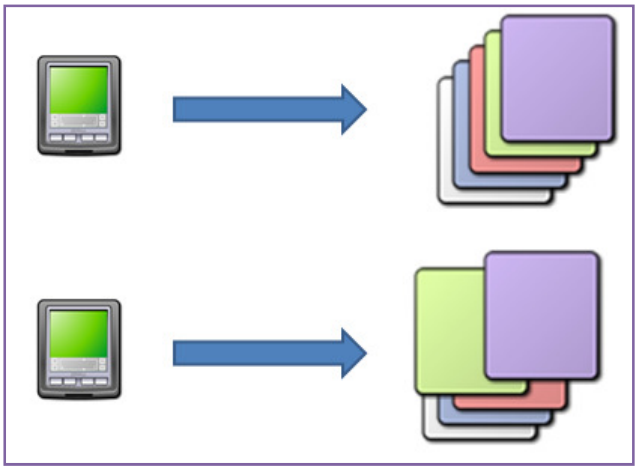
Es importante conocer ciertos detalles del funcionamiento del GIS en la web antes de desarrollar aplicaciones. Para un desarrollador que nunca ha trabajo con servicios GIS de mapas pero esté acostumbrado a trabajar consumiendo servicios web, puede resultar algo diferente el uso que hace el API de ArcGIS de los servicios de mapas dinámicos.

Toda la comunicación entre el cliente y el servidor se realiza mediante un API Rest en el servidor, facilitada por el API de ArcGIS:



Un servicio de mapas dinámico, como se verá más adelante, se puede entender como un servicio web que puede ser consumido por una aplicación para mostrar el mapa en pantalla. Ahora bien, en el mundo del GIS, un mapa se compone de un número determinado de capas que forman en su conjunto el mapa final. ArcGIS es capaz de dar acceso en sus servicios al mapa en su conjunto pero también a cada una de las capas que lo componen. Esta particularidad ayuda en especial a los desarrolladores, como se verá más adelante, a la hora de consultar datos de determinadas capas de un mapa.

De esta forma, una aplicación puede realizar peticiones a todo el servicio de mapas completo o puede realizar peticiones a una determinada capa:



En un nivel superior, una aplicación normalmente consume varios mapas al mismo tiempo formando un *mashup*.

2.4. – Ejercicio: Poniendo en marcha ArcGIS Runtime

Paso 1

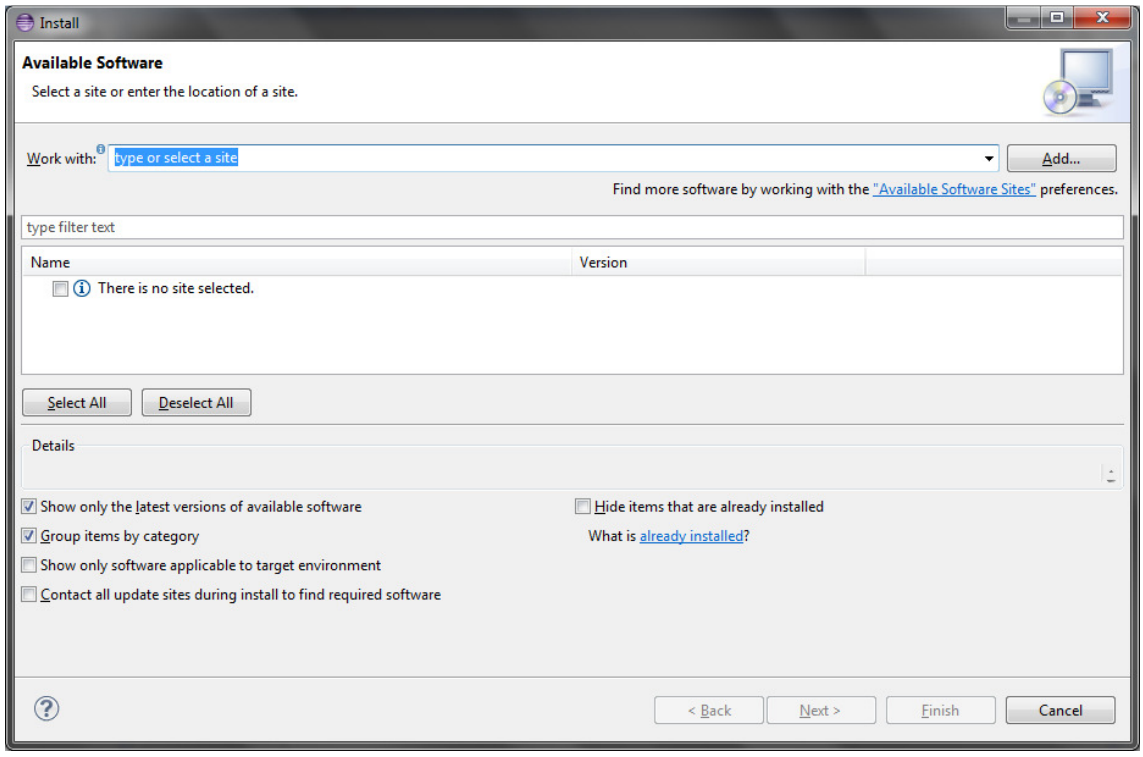
Antes de comenzar, es necesario instalar el Runtime SDK para Android de ArcGIS. Realmente, se trata de un *plugin* para Eclipse diseñado para que el desarrollo de aplicaciones GIS para la plataforma Android 2.2 y superiores sea lo más sencillo y robusto posible.

Comience la descarga del Runtime desde la página web de Esri: http://www.esri.com/apps/products/download/index.cfm?fuseaction=download.all#ArcGIS_Runtime_SDK_for_Android

Paso 2

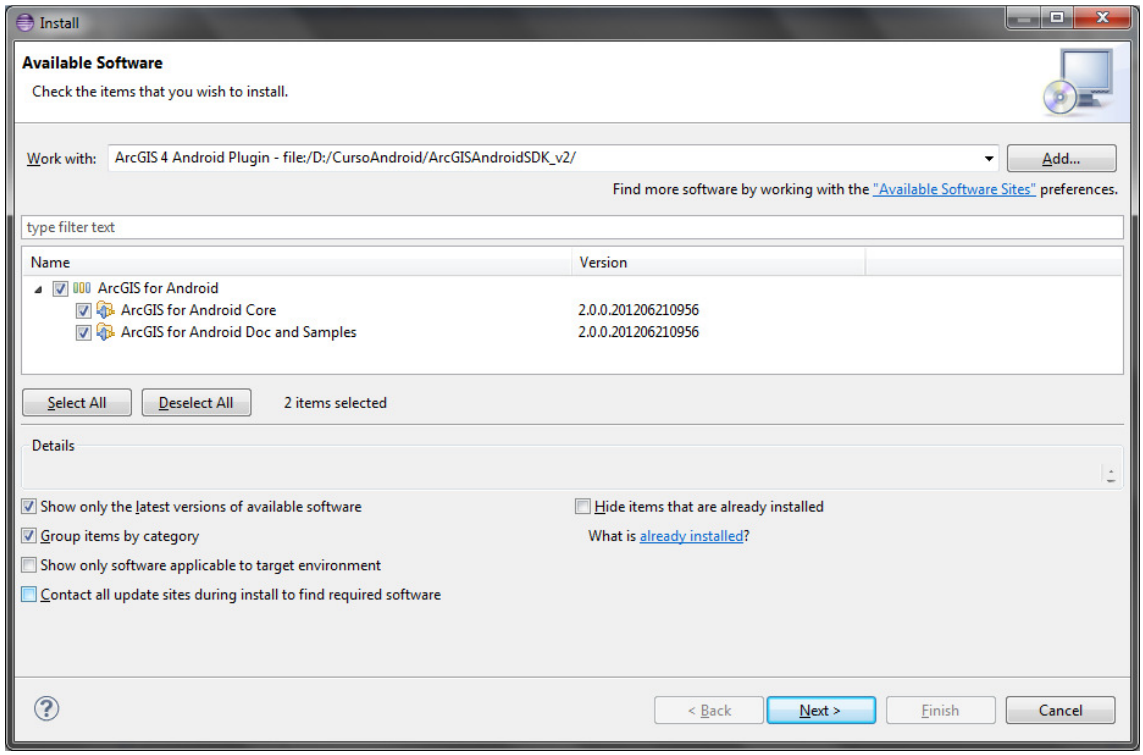
Una vez descargado el fichero *.zip*, descárguelo en una ruta y recuérdela.

Arranque Eclipse con el acceso directo del escritorio. Una vez puesto en marcha, haga clic en el menú "**Help > Install New Software**".

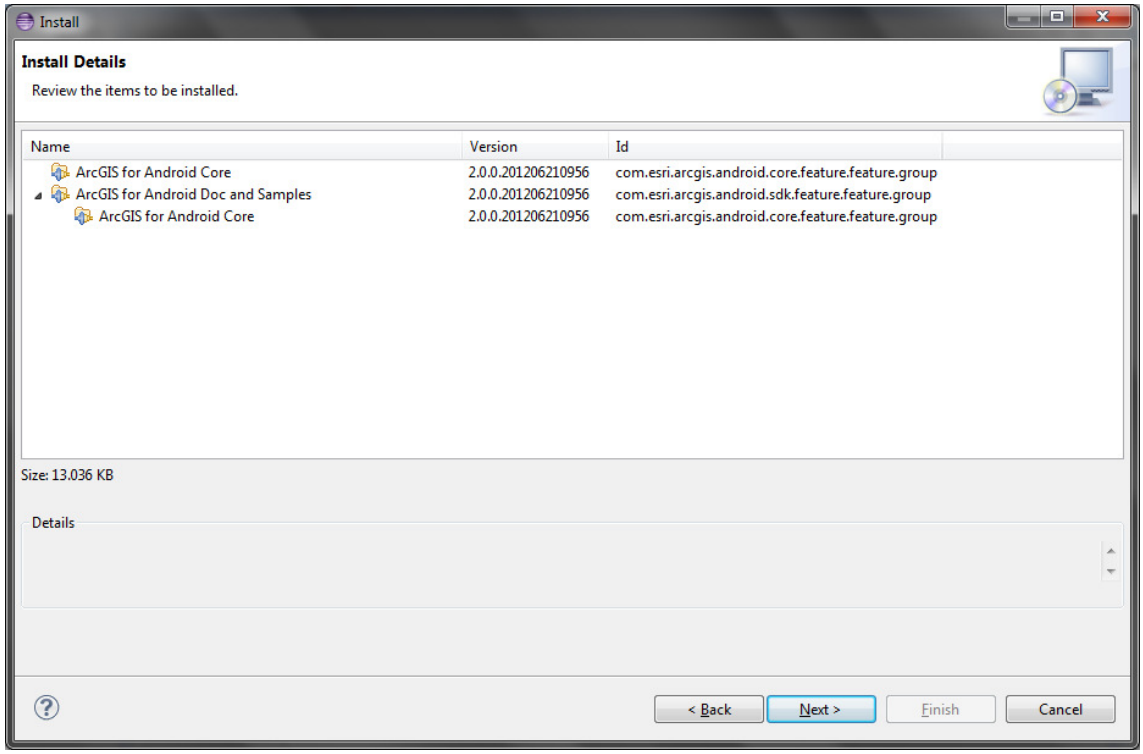


Pulse el botón "Add" arriba a la izquierda. En el siguiente formulario, pinche "Locate" para localizar la carpeta donde descomprimió el Runtime SDK y como nombre escriba "ArcGIS 4 Android Runtime". Acepte y continúe.

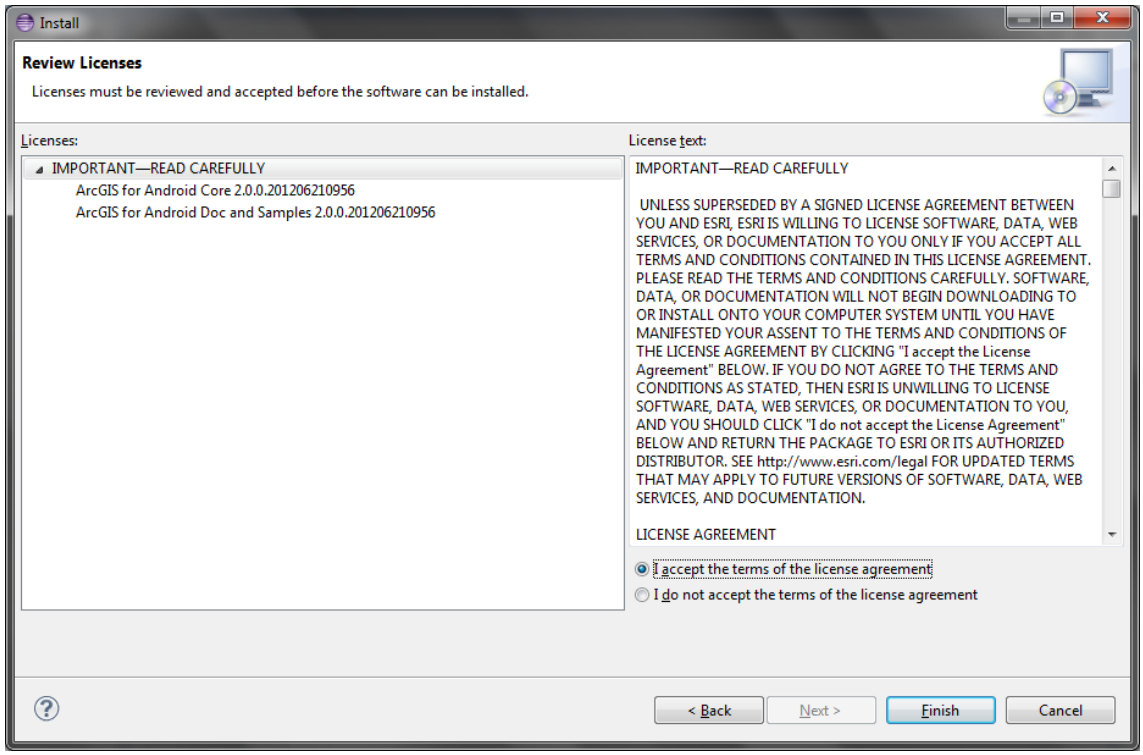
En la ventana de “Available Software”, extienda la categoría “ArcGIS for Android” para ver todo el software disponible, tanto el “Core” como la documentación y ejemplos. Seleccione ambos para que sean instalados en su sistema y pinche “Next”.



Utilice las siguientes ventanas para descargar e instalar el software necesario.



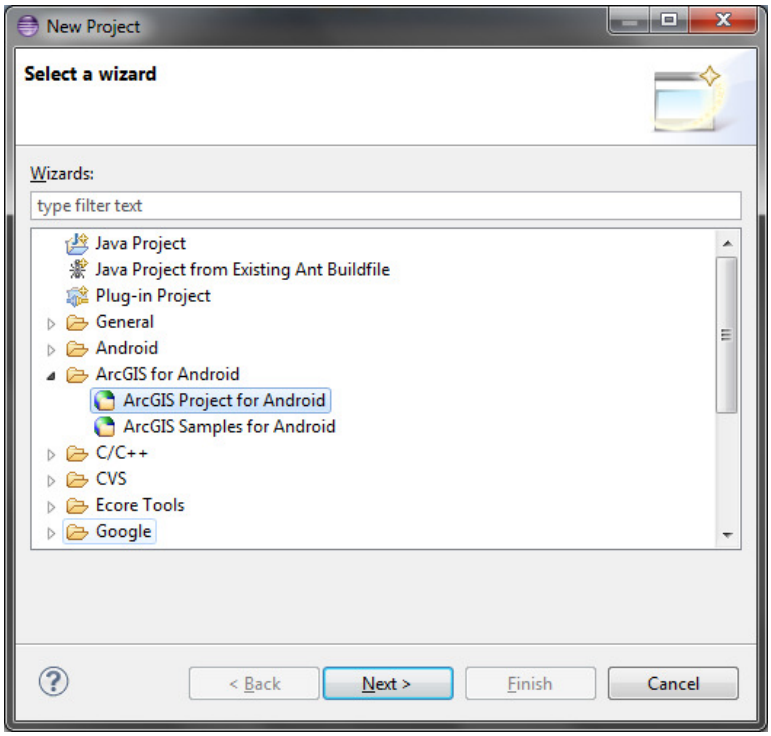
Acepte las condiciones de uso y pulse “Finish”.



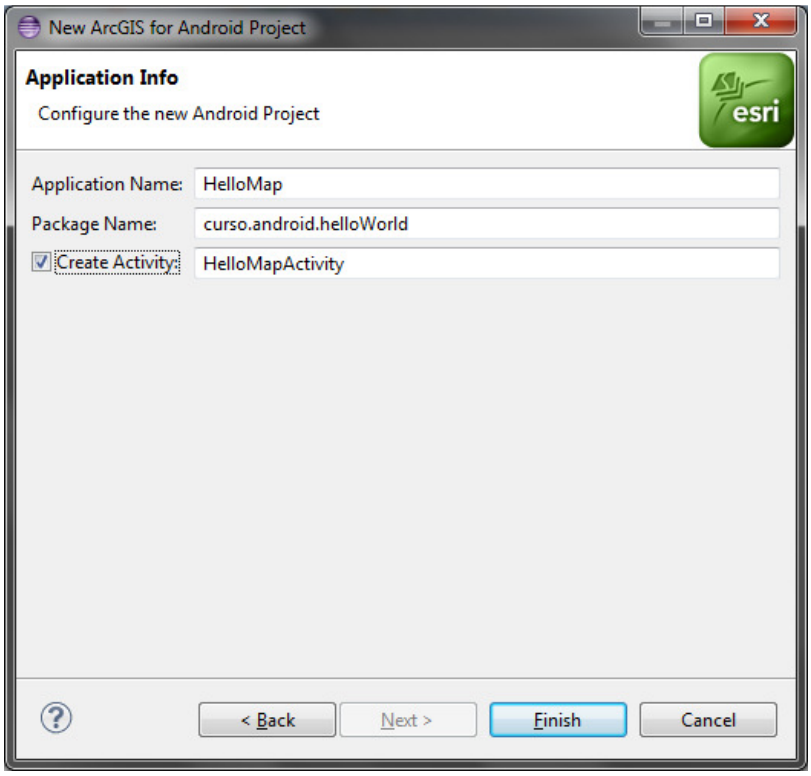
Paso 3

Una vez realizada la instalación, cierre Eclipse y vuelva a abrirlo.

Cree un nuevo proyecto y seleccione el *wizard* de ArcGIS Project for Android:

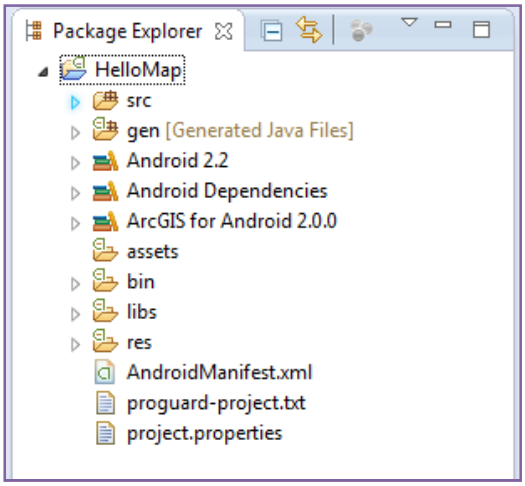


En la siguiente ventana, de un nombre a la aplicación, por ejemplo, “HelloMap”, un paquete, por ejemplo “curso.android.helloWorld” y seleccione la opción de crear una *Activity* por defecto.



Pulse “Finish” y espere que su proyecto termine de crearse.

Una vez tenga el proyecto, compruebe en el “Package Explorer” que todo ha sido creado correctamente.



Compruebe especialmente que las dependencias “ArcGIS for Android 2.0.0” han sido añadidas a su proyecto Android, con lo que podrá desarrollar desde este momento con el Runtime SDK.

En el siguiente tema, comenzará a desarrollar y ejecutar este tipo de aplicaciones.

3.- Entorno de desarrollo: Eclipse

3.1.- Eclipse y ADT

Para el desarrollo utilizando el Runtime SDK de Android, se utiliza el lenguaje de programación Java. En este curso, se emplea el entorno de desarrollo más extendido para dicho lenguaje, y muchos otros: Eclipse.

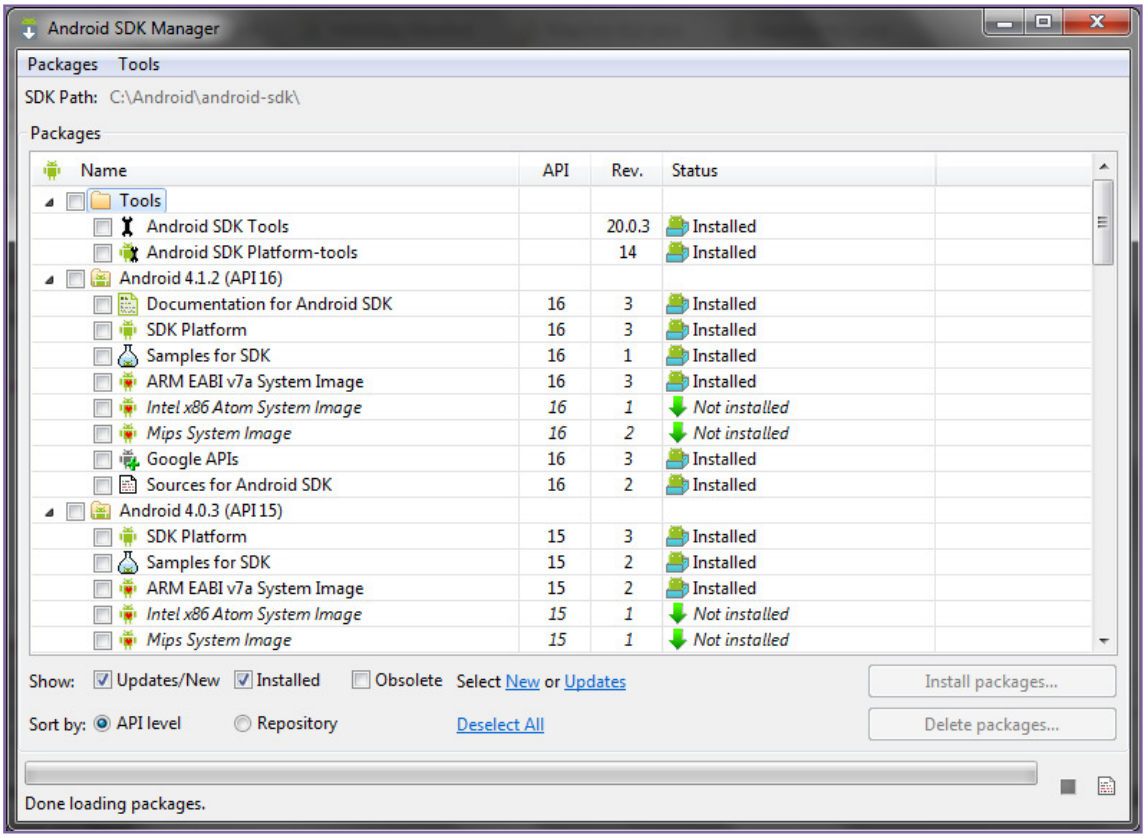
Eclipse es un entorno de desarrollo de código abierto multilenguaje y multiplataforma con una importante comunidad de desarrolladores aunque fue creado originariamente por IBM. Está basado en instalaciones mediante plugins, lo cual ha permitido que pueda utilizarse para desarrollar en otros lenguajes como C, C++ o Python.

Para poder desarrollar con Android en Eclipse, necesitamos instalar previamente el SDK de Android, que en realidad es el núcleo de librerías de Android y el ADT (Android Development Tools), el plugin desarrollado por Google. Aunque podría programarse sin el ADT, con tal sólo el núcleo de Android, es muy recomendable utilizar el ADT ya que nos provee de una serie de herramientas realmente útiles como editores gráficos, opciones para realizar *debug*, gestión de documentos XML,…

Para la instalación del ADT, puede encontrar información en la web de desarrollo Android: http://developer.android.com/sdk/installing/installing-adt.html

Una vez instalado el ADT, es posible ver en Eclipse dos botones integrados en una nueva barra de herramientas y que van a utilizarse a lo largo del curso:

El primero botón nos da acceso al SDK Manager de Android, mediante el cual podemos comprobar los componentes relacionados con Android instalados en nuestra máquina, instalar nuevos componentes o actualizar los que tenemos:



El segundo botón hace referencia al AVD Manager del cual se hablar más adelante.

3.2.- Emuladores o AVD

Una parte importante del desarrollo para móviles es la gestión de emuladores. Un emulador, es un software cuyo funcionamiento simula la implementación de un hardware y un sistema operativo de tal modo que no necesitamos tener un dispositivo físico para programar.

Al instalar el ADT para Eclipse, disponemos también de una herramienta fundamental para el desarrollo, el AVD (Android Virtual Device) manager. Se trata de un software que nos permite crear y gestionar emuladores para Android. Según su propia especificación, un AVD consiste en:

- Un hardware específico
- Un sistema operativo sobre dicho hardware
- Otras opciones gráficas de apariencia
- Un mapeo de memoria en la máquina donde se está desarrollando que simula ser la tarjeta SD del dispositivo móvil

A la hora de desarrollar software con Android, debemos recordar que el software desarrollado depende de la versión del sistema operativo (Android) sobre el cual se va a instalar y posteriores.

Como buenas prácticas debemos tener en cuenta los siguientes puntos:

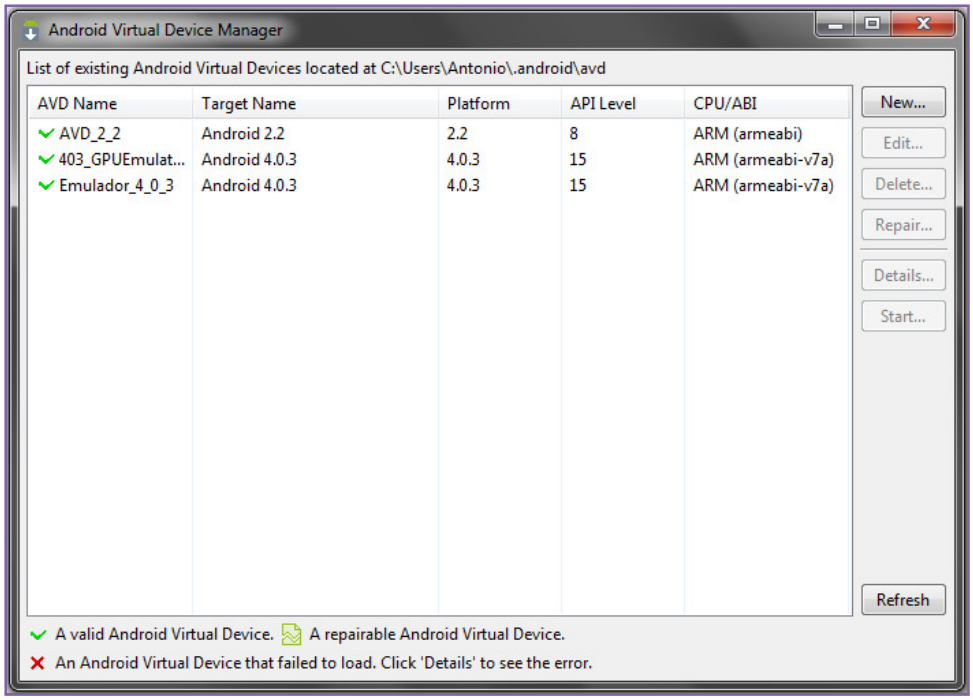
- Los AVD que utilicemos deben utilizar una versión del SDK de Android igual o mayor que la versión para la cual estamos desarrollando (minSdkVersion).
- Lo óptimo sería comprobar el funcionamiento de nuestro software no sólo en un AVD con la misma versión sino con las versiones superiores, de tal forma que aunque nuestros usuarios actualicen su sistema operativo, nuestra aplicación siga funcionando correctamente.
- Si utilizamos librerías de desarrollo externas al SDK de Android, debemos pensar que dichas librerías deben estar presentes en los AVDs y los dispositivos físicos finales.

Por último, también tenemos que tener en cuenta que un emulador no es tan completo como un dispositivo físico y que, por tanto, tiene ciertas limitaciones, de las cuales las más importantes son:

- No es posible simular conexiones USB al dispositivo
- No se soporta la determinación del estado de la tarjeta SD
- No se soportan conexiones Bluetooth
- No se soporta la determinación del estado de carga de la batería
- No se soporta el uso del acelerómetro del dispositivo

Aunque es posible evitar estas limitaciones mediante software adicional, en la mayoría de proyectos GIS no son limitaciones importantes.

En la barra de herramientas de Eclipse para acceder a AVD Manager, pinchamos el segundo botón y accedemos al mismo:



En este formulario podemos ver los AVDs disponibles, modificarlos, crear nuevos y también arrancar un dispositivo concreto. Una vez arrancado, obtendremos una emulación de este tipo:

Más adelante, se verá el sistema operativo ejecutando aplicaciones.

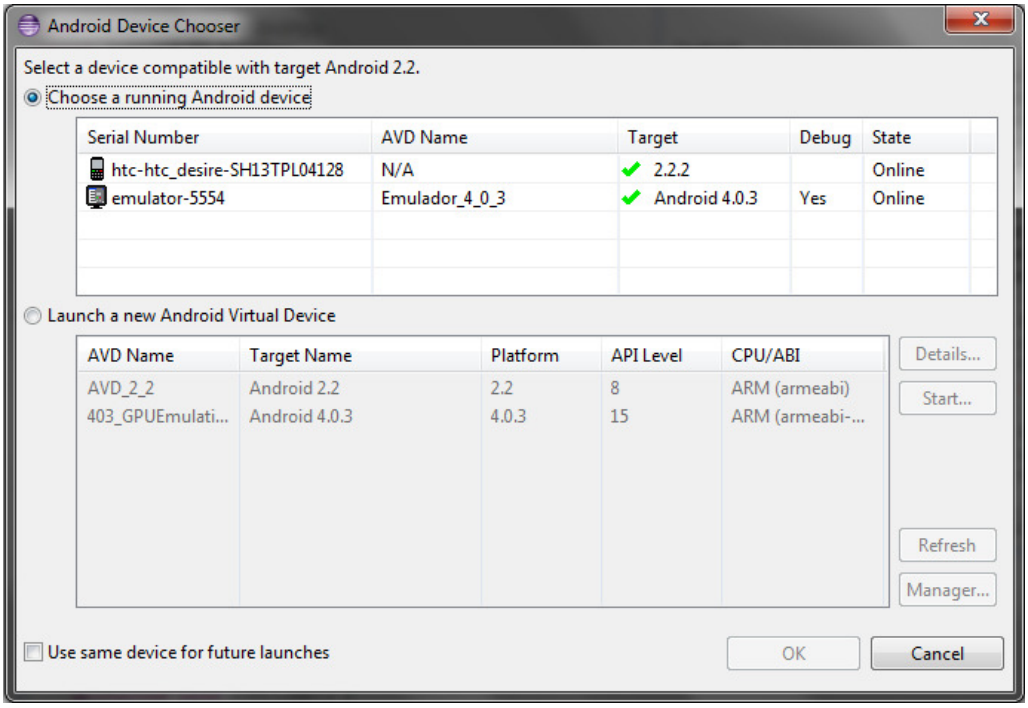
3.3.- Desarrollo con dispositivos físicos

Aunque en este curso, se va a utilizar un emulador para el desarrollo, es importante recordar que en última instancia siempre resulta importante debuggear el programa en un dispositivo físico real.

Para ello, tenemos que tener en cuenta ciertos puntos:

- La aplicación en desarrollo debe estar marcada como *debuggable* en el Android Manifest (puede encontrar más información en el Anexo 1 al final de este documento).
- El dispositivo debe aceptar la opción de *debug*. Esta opción está disponible en los dispositivos Android en "Settings > Applications > Development" o bien en "Settings > Developer options".
- El sistema operativo en la máquina de desarrollo debe tener instalados los drivers adecuados. En Mac OS X viene por defecto y para Windows puede encontrar los drivers aquí: http://developer.android.com/tools/extras/oem-usb.html

Una vez realizados estos pasos, cuando lance una desarrollo Android en Eclipse, accederá al panel Android Device Chooser que le permitirá escoger entre los dispositivos conectados y los AVDs creados:



En este caso hay dos dispositivos ya disponibles, un dispositivo físico conectado al ordenador y un emulador. El dispositivo físico es un móvil HTC Desire con un sistema operativo Android 2.2.2 (Froyo) instalado mientras que el emulador consta de una versión Android 4.0.3 (Ice Cream Sandwich).

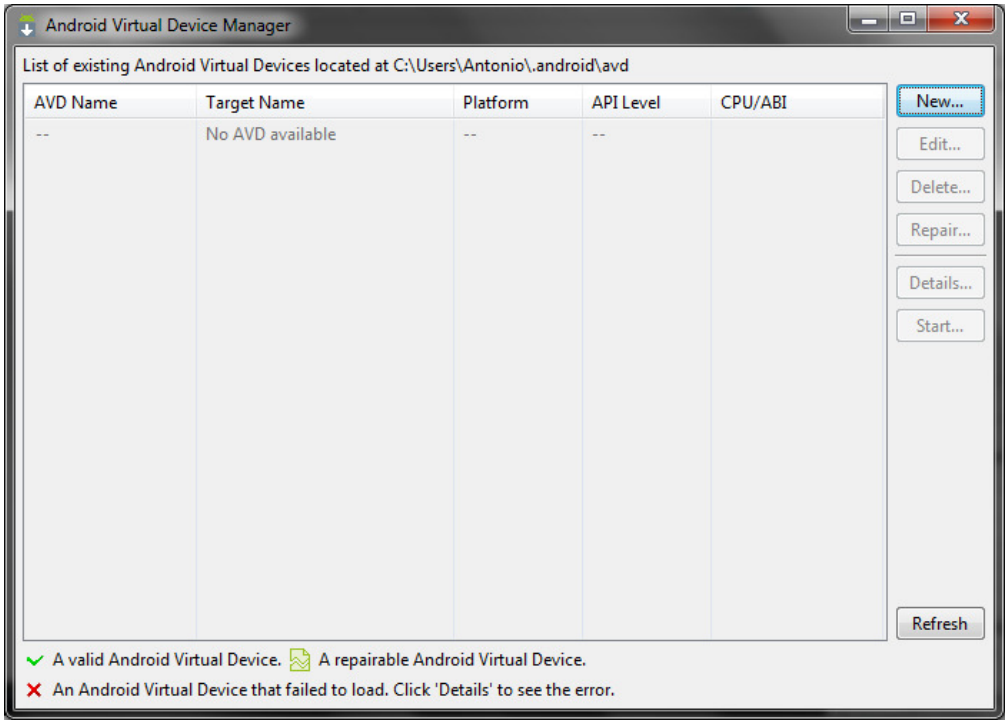
3.4.- Ejercicio: ArcGIS en AVD

Paso 1

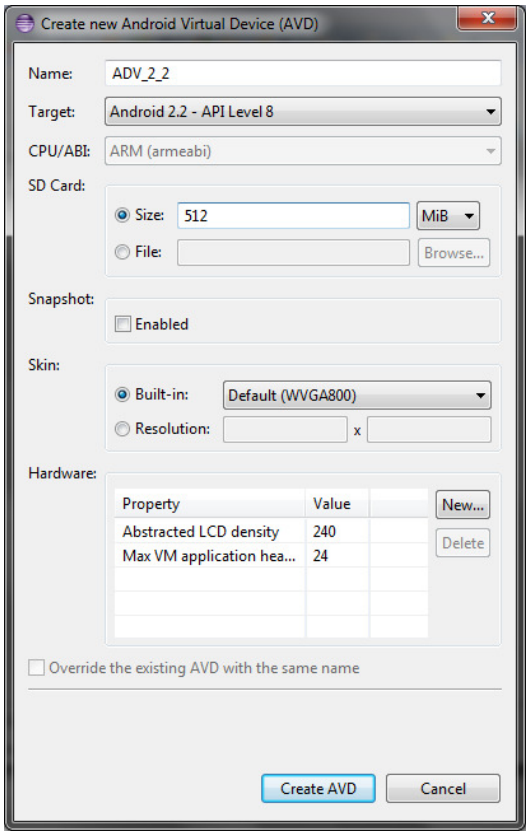
En este ejercicio va a crear un proyecto con el Runtime SDK de ArcGIS para Android y lo ejecutará en un emulador AVD.

Debido a que en este momento no cuenta con ningún emulador en su máquina, comenzará por crear uno. Arranque Eclipse y localice el AVD Manager:

Haga clic en él y accederá a la ventana de gestión de AVD:

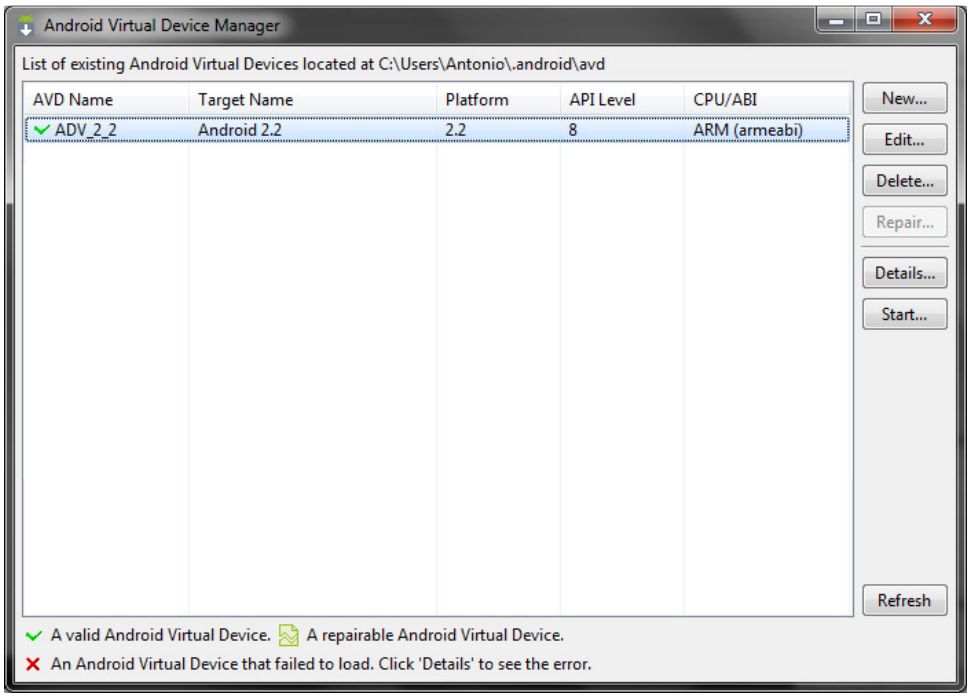


Pinche en el botón “New…” para crear una nuevo AVD:



De un nombre a su AVD y seleccione la versión Android 2.2 (API 8) como el target de la misma. Además, debe dar un tamaño a la tarjeta SD virtual de su AVD. Recuerde que este tamaño será reservado de su disco duro para el uso exclusivo del emulador. En este caso, un tamaño de 512MB es más que suficiente.

Una vez finalizado, pinche en “Create AVD” y espere a que sea creado:



Una vez creado el AVD, puede ser arrancado desde esta ventana o puede ser arrancado cuando se ejecute un proyecto. Para ahorrar tiempo, conviene arrancar un AVD que va a ser utilizado durante una larga sesión de desarrollo y no cerrarlo entre ejecución y ejecución. Esto es posible ya que al contrario que en otros entornos de desarrollo, como Visual Studio, la ejecución de una aplicación, no bloquea el entorno.

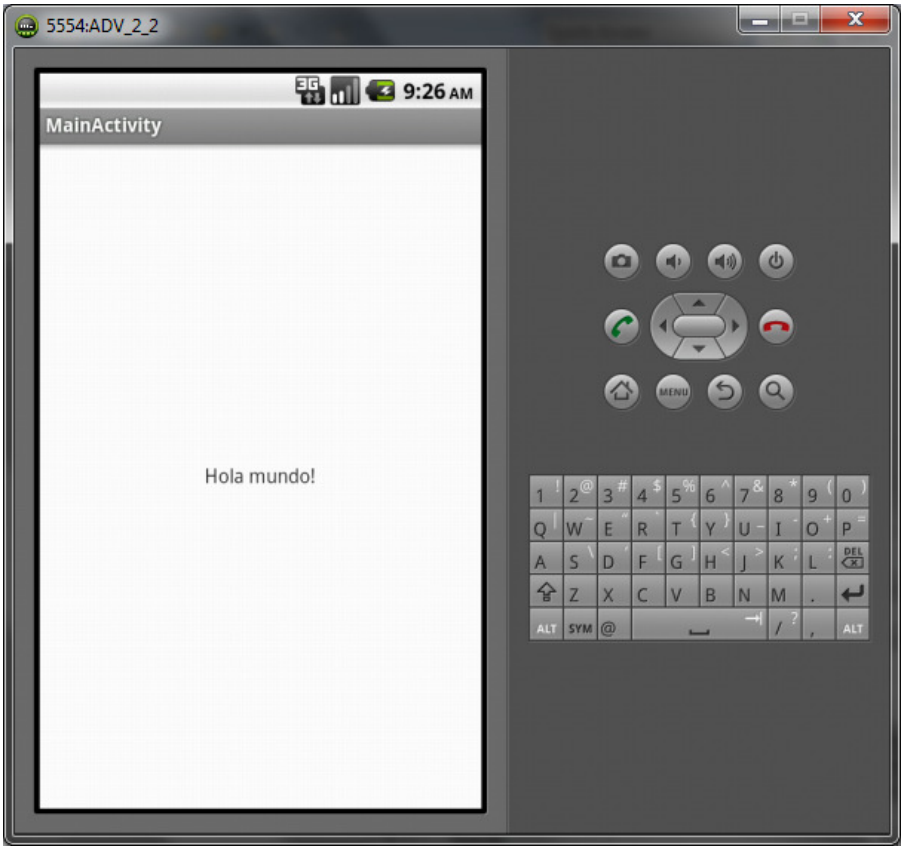
Seleccione su nuevo AVD y pinche "Start". El emulador será arrancado en una nueva ventana. Minimícelo y vuelva a Eclipse. Puede cerrar la ventana AVD Manager sin que el emulador se detenga.

Paso 2

Si no tiene abierto el proyecto "HolaMundo" realizado anteriormente, ábralo de nuevo.

Ahora que cuenta con un emulador, puede lanzar su proyecto. Localice el botón "Run" para ejecutar el proyecto:

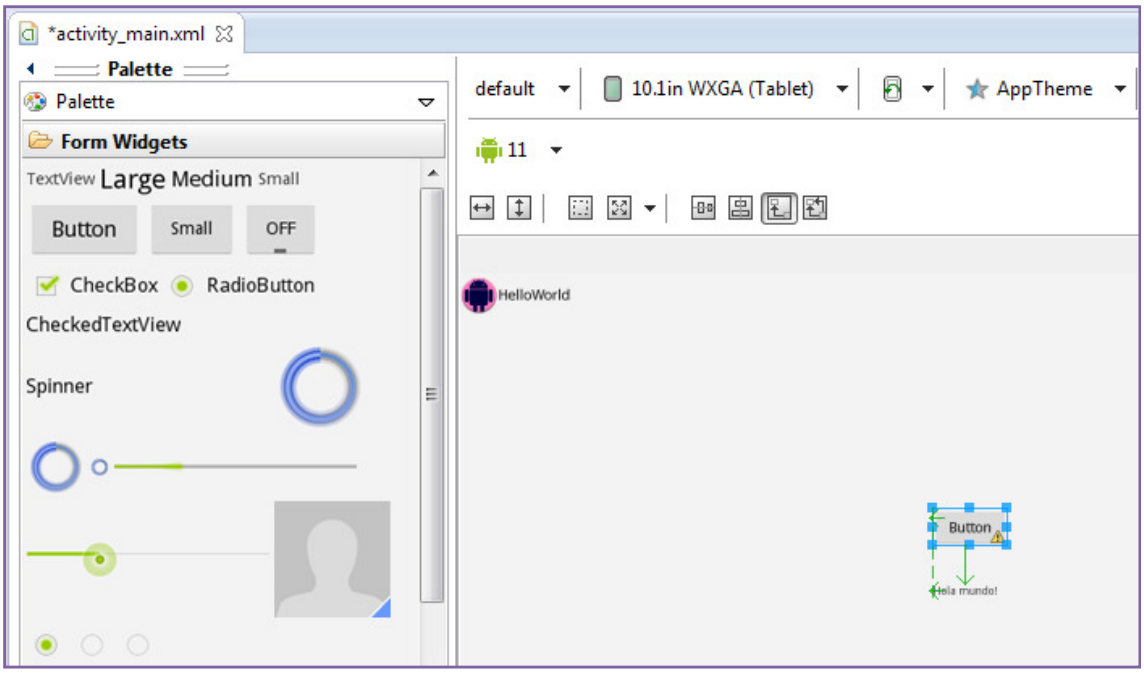
Pinche el botón y al ser preguntado cómo lanzar la aplicación, seleccione su nuevo emulador y la aplicación se instalará y ejecutará en el AVD:



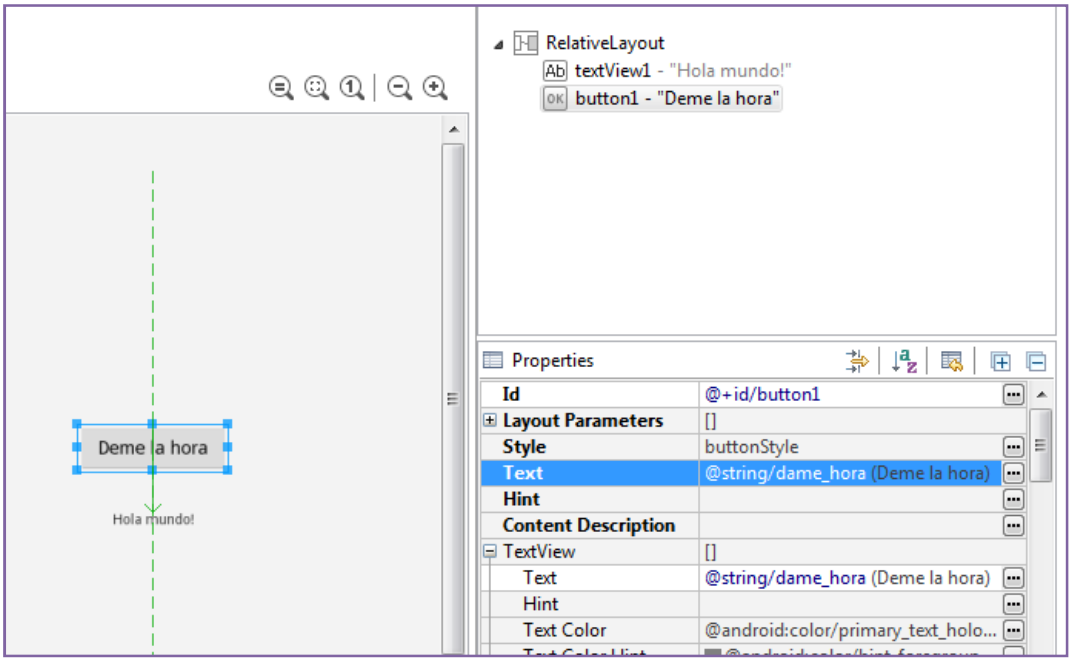
Paso 3

Minimice el AVD y regrese a Eclipse. Localice el *layout* principal, activity_main.xml, y ábralo.

Se va a modificar el *layout* añadiendo algo de funcionalidad al mismo. Para ello, en el menú de componentes, a la izquierda, busque la carpeta “Form Widgets” y arrastre un componente “Button” a la interfaz:



Verá una advertencia en el botón indicando que el texto del mismo es *hardcoded*, es decir, no ha sido externalizado en un fichero de recursos. En capítulos anteriores, ya vio cómo realizar este proceso mediante el fichero strings.xml. Cree un nuevo recurso de texto cuyo valor sea “Deme la hora” y aplíquelo a la propiedad “Text” del botón:



El objetivo es obtener la hora del sistema cada vez que el usuario haga clic en el botón. Hasta el momento, ha definido la interfaz necesaria: el botón y el texto, el cual será sobrescrito cada vez que se haga clic.

Una vez realizada creada la interfaz, debe ser creada la funcionalidad. Para ello abra la Activity principal de su proyecto y añada dentro de la clase "MainActivity" un método que capturará el evento clic y obtendrá la hora exacta del sistema:

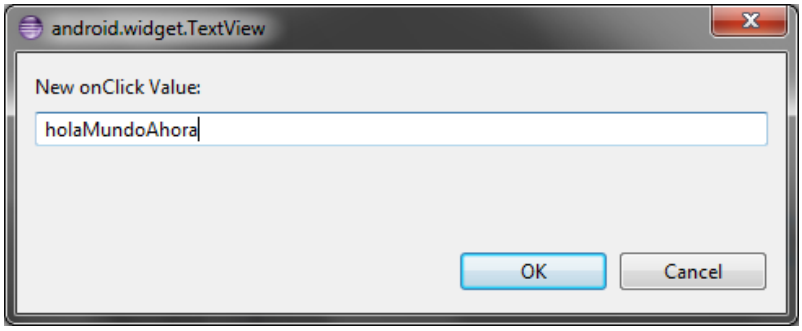
```
public void holaMundoAhora(View view) {

}
```

Como parámetro del método llega un objeto de la clase "View". Por defecto, no ha importado el paquete donde se aloja dicha clase a su código por lo que se marcará en rojo. Eclipse ayuda a la importación de clases con el atajo de teclado: **CTRL + SHIFT + O**. Verá como la clase View es importada y puede continuar sin errores.

Conceptualmente, la clase "View" representa el bloque mínimo de una interfaz Android. Esto quiere decir, que los elementos de la UI habituales (botones, *text fields*, *checkbox*,…) derivan de dicha clase. Por lo tanto, dicho método es genérico y podría aplicarse a cualquier elemento del *layout*. En este caso, lo utilizaremos para que se ejecute cuando el usuario pinche sobre el botón "Dame la hora" por lo que el "View" que se recibirá como parámetro será precisamente dicho botón.

Puede encontrar más información en el API Reference de Android:
http://developer.android.com/reference/android/view/View.html#attr_android:onClick

Para asociar el evento clic del botón con este método, vaya de nuevo al *layout*, haga clic con el botón derecho sobre el botón y localice "**Other properties > Inherited from View > OnClick…**" y escriba el método "holaMundoAhora" que acaba de desarrollar:



También puede acceder al texto XML del *layout* y añadir en el botón la línea de código:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="39dp"
    android:text="@string/dame_hora"
    android:onClick="holaMundoAhora"      />
```

De cualquiera de los dos modos, cuando suceda el evento clic del botón, se ejecutará el método “holaMundoAhora” y recibirá como parámetro dicho botón.

Ahora es necesario que complete el código del método “holaMundoAhora” de su *Activity*. Puede hacerlo en 3 pasos. El primer paso consiste en obtener el “TextView” donde originalmente ha escrito “Hola Mundo!”. Esta cadena de texto está representada en un fichero de recursos, con lo que accederá al recurso a través de la clase “R” automáticamente generada por el ADT y que recordará que almacena referencias a todos los elementos del proyecto. Además, utilizará la método heredada “findViewById()” que le permite localizar en Android cualquier elemento de interfaz de usuario:

```
TextView textView = (TextView) findViewById(R.id.textView1);
```

Como segundo paso, es necesario obtener del sistema la hora actual. Hay muchas formas de realizar este paso, una de las más sencillas es utilizando la clase “Time”:

```
Time now = new Time();
now.setToNow();
```

Por último, aplique al “TextView” un nuevo texto compuesto por la cadena original “Hola Mundo!” seguido de la hora actual formateada adecuadamente. Para obtener el texto de la cadena original, necesitará de nuevo de la clase “R” así como del método heredado “getString()”:

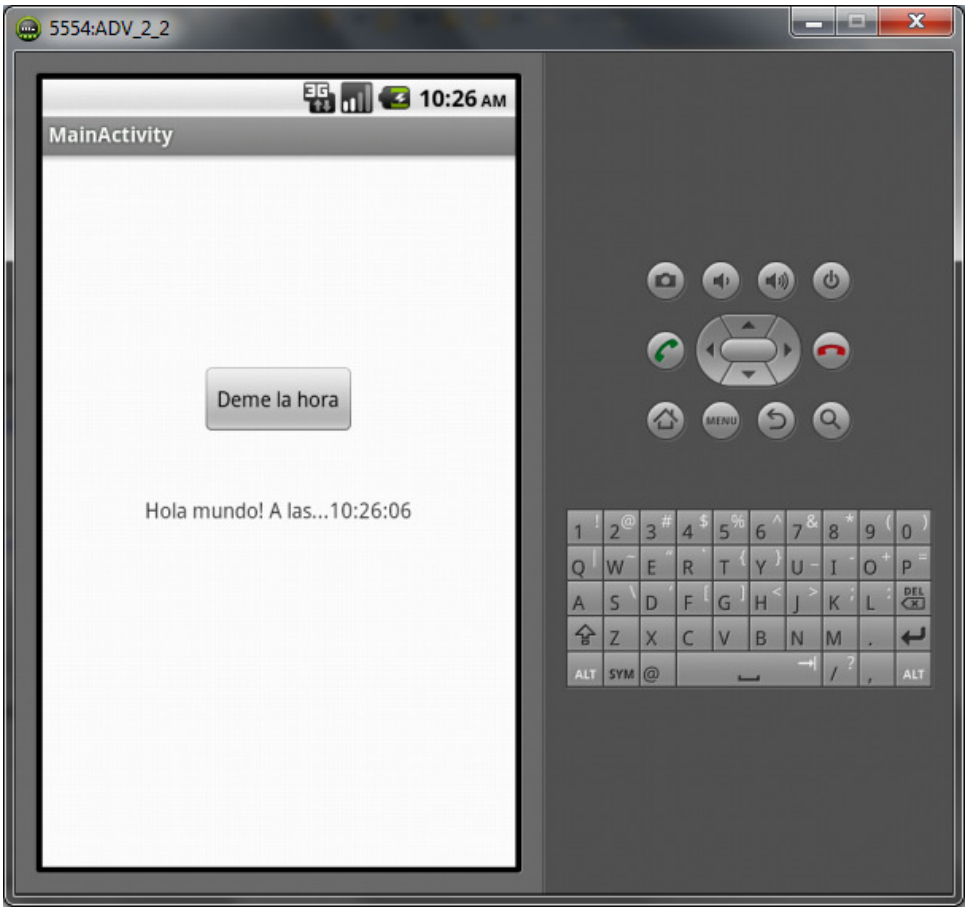
```
textView.setText(getString(R.string.hola_mundo) + " A las..." + now.format("%k:%M:%S"));
```

En definitiva, su método debería quedar así:

```
public void holaMundoAhora(View view) {
	// Obtenemos el TextView del texto "HolaMundo"
	TextView textView = (TextView) findViewById(R.id.textView1);

	// Creamos una instancia de tiempo y la asociamos al actual
	Time now = new Time();
	now.setToNow();

	// Obtenemos el string original de "Hola mundo!" y le añadimos la hora
actual
	textView.setText(getString(R.string.hola_mundo) + " A las..." +
now.format("%k:%M:%S"));
}
```

Salve todos los cambios realizados en los diferentes ficheros.

Ahora está listo para ejecutar el programa. Compruebe que el AVD está arrancado y lance la aplicación. Si Eclipse no detecta ningún emulador o es la primera vez que ejecuta una aplicación, le preguntará en qué sistema quiere ejecutarlo. En este caso, simplemente, seleccione su AVD y continúe:



Compruebe la correcta ejecución del programa y vea cómo se actualiza el “TextView” cada vez que haga clic sobre el botón.

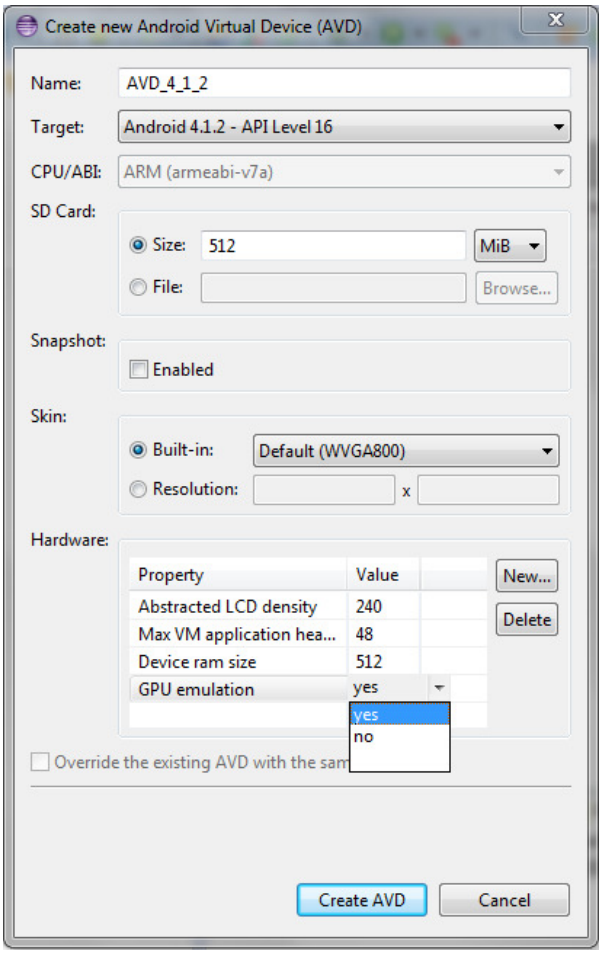
Paso 4

Una vez realizado y ejecutado un proyecto Android simple, tratará de crear un proyecto con el Runtime SDK de ArcGIS para Android. Como primer intento, utilizará uno de los proyectos de ejemplo que ha instalado previamente con el Runtime.

Cierre el emulador si lo tiene abierto. Este primer AVD creado, no es útil para los proyectos con el Runtime debido a que para la renderización de mapas, se utiliza OpenGL ES 2.0 con aceleración mediante GPU y se soporta la ejecución de estos proyectos desde la versión 4.0.3 de Android por lo que será necesario crear un AVD nuevo para esta versión.

Recuerde que esto no implica que su aplicación tenga que funcionar en un dispositivo con esta versión de la plataforma, puede utilizar el Runtime SDK para versiones Android 2.2 y superiores. A efectos prácticos, esto quiere decir que sus proyectos Android que utilicen el Runtime requerirán una versión mínima 2.2 (Froyo) y como *Build Target* necesitará una versión al menos 4.0 si es que quiere hacer *debug* en un emulador. Como práctica útil, puede crear sus proyectos con un valor mínimo 2.2 y como *target*, la mayor versión de Android que tenga instalada en su equipo.

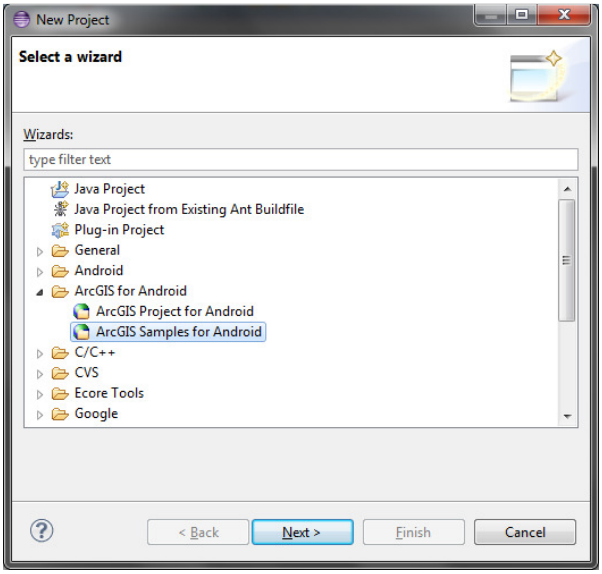
Una vez tenido esto en cuenta, acceda al AVD Manager y cree un nuevo AVD para la versión 4.1 con un requisito de adicional. En la sección de "Hardware" de la ventana de creación de un nuevo AVD pinche sobre "New" y busque en el desplegable de propiedades "GPU Emulation". Acepte y luego modifique este nuevo parámetro con el valor "true":



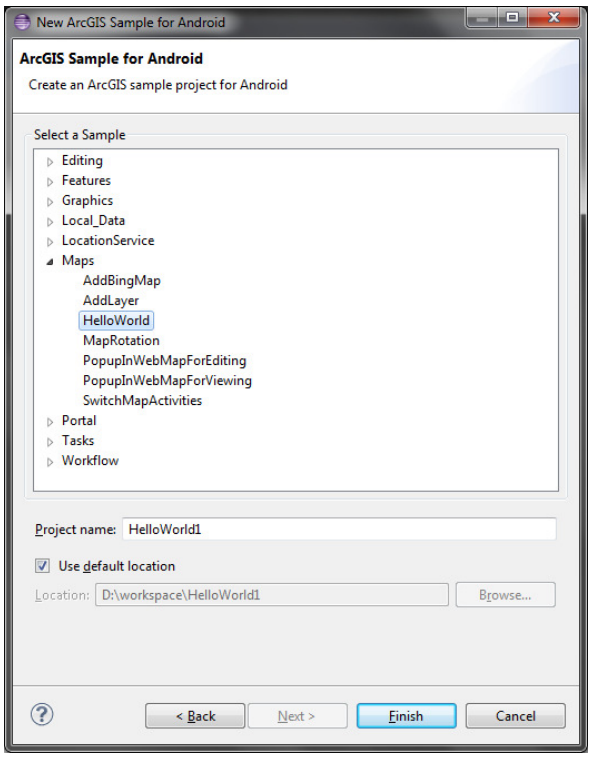
Cree su nuevo AVD e inícielo.

Paso 5

Desde el menú “File” de Eclipse cree un nuevo proyecto y en la ventana de selección del tipo de proyecto, busque la carpeta “ArcGIS for Android” y seleccione un nuevo proyecto de ejemplo:



El Runtime le provee de un número importante de proyectos de ejemplo que puede utilizar y modificar para crear su propio proyecto. Como primer caso, busque el proyecto de ejemplo “Hello World” y finalice:

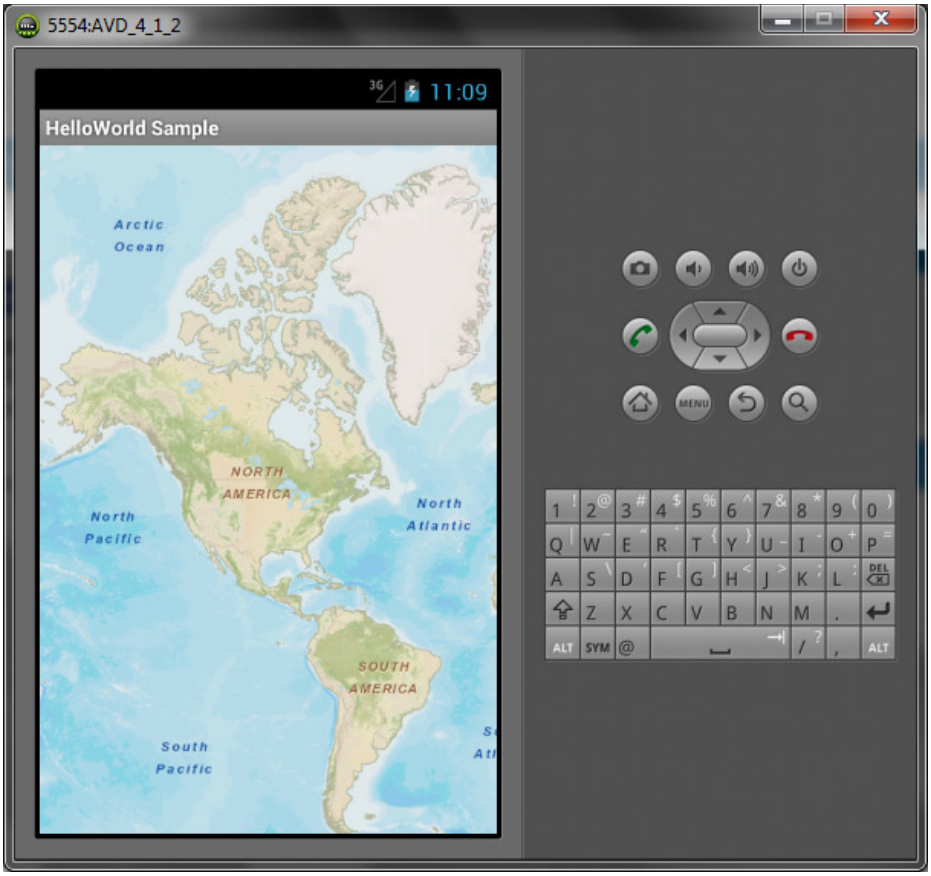


Es posible que reciba un error tras crear el proyecto indicándole que la versión de compilación es incorrecta. Para solventar este problema haga clic con el botón derecho desde el *Package Explorer* sobre el nuevo proyecto, busque el submenú "Android Tools" y haga clic en "Fix Project Properties".

Además, puede cerrar o eliminar el proyecto realizado en el ejercicio anterior si lo tiene cargado en Eclipse.

Una vez solventados estos problemas, puede visualizar el código y comprobar cómo se ha realizado el proyecto y cómo se cargan las librerías del Runtime en formato .jar en el proyecto dentro de la carpeta "ArcGIS for Android 2.0.0".

Cuando esté preparado, ejecute el proyecto como una aplicación Android en el AVD que tiene inicializado. Es posible que tarde unos minutos en cargarse la aplicación y el mapa debido a las limitaciones del emulador:



Navegue por el mapa y compruebe que funciona correctamente.

4.- Consumir servicios web

El paradigma Cliente-Servidor, revisado anteriormente, aplicado al escenario del mundo GIS, se convierte en la necesidad de consumir servicios web, en su mayoría servicios de mapas, utilizando las facilidades del Runtime SDK de ArcGIS.

4.1.- Tipos de servicios disponibles

Los tipos de servicios de mapas que hay disponibles para su uso son los siguientes:

Tipo	API Class	Rendimiento
Mapa cacheado	ArcGISTiledMapServiceLayer BingMapsLayer	★★★
Mapa dinámico	ArcGISDynamicMapServiceLayer ArcGISImageServiceLayer	★★
Graphics Layer	GraphicsLayer	★
Feature Layer	ArcGISFeatureLayer	★

Mapa cacheado

Esri soporta mapas cacheados tanto de ArcGIS Server como de Bing Maps de Microsoft. Un servicio de mapa cacheado provee de imágenes predefinidas a ciertas escalas según un esquema de cacheado alojado en el servidor. Gracias al Runtime SDK, es posible realizar aplicaciones Android que entienden estos esquemas y son capaces de realizar las solicitudes de imágenes necesarias para el usuario de forma sencilla.

Aunque estos mapas pueden contener diferentes capas, al ser pregeneradas en el servidor, no es posible modificar la visibilidad de algunas de ellas. Estos mapas suelen ser estáticos y varían muy poco en el tiempo ya que si cambiaran, sería necesario regenerar las imágenes del servidor cada poco tiempo. Ejemplos clásicos de mapas cacheados son mapas polítos, callejeros, fotos aéreas,…

Este tipo de mapa es el más rápido de los disponibles debido a que las imágenes están alojadas en el servidor, y este sólo tiene que enviarlas al dispositivo. Este tipo de servicios suelen utilizarse para los mapas base, mapas que se posiciones en la parte inferior de tal forma que otros datos puedan ser renderizados encima de este.

Dynamic map service layers

No siempre es posible tener un mapa cacheado en el servidor. En muchas ocasiones los datos varían constantemente y necesitamos representarlos de este modo. Por ejemplo, un servicio de mapas meteorológico que varía cada día o un servicio de tráfico que se actualiza cada hora.

En estos casos, en lugar de simplemente almacenar las imágenes, el servidor tendrá que generar una imagen con cada petición que recibe de la aplicación.

Otra ventaja de los mapas dinámicos es la posibilidad de acceder a las capas que forman el mapa. Aunque, por defecto, en la clase "ArcGISDynamicMapServiceLayer" no hay métodos para filtrar cada una de las capas, es posible realizar filtros mediante las denominadas definition expressions, parecidas a las consultas, y también es posible realizar consultas mediante la clase "QueryTask".

Es importante conocer el sistema de proyección de los servicios de mapa debido a que es posible que al combinar mapas cacheados y mapas dinámicos se generen problemas de diferentes tipos. Es uso de *mashups* de este estilo es muy común en las aplicaciones GIS, teniendo un mapa base cacheado con una serie de mapas dinámicos o *graphics layers* sobre ellos.

Graphics layers

Los servicios cacheados devuelven a la aplicación imágenes que están almacenadas en el servidor mientras que los servicios dinámicos generan una nueva imagen con los datos que necesita la aplicación en cada momento. En ambos casos, la información está alojada en el servidor y la aplicación para obtener nueva información requiere de la comunicación con el servidor ya que es este quien la almacena.

Sin embargo, una *graphics layer* es una capa que pertenece del lado del cliente, es decir, es un elemento de la aplicación. En estas capas gráficas, el posible dibujar elementos de un mapa a través de un *renderer* o un *symbol* como se verá en el siguiente tema.

Normalmente, este tipo de capas, pueden considerarse como un contenedor donde elementos gráficos pueden ser pintados y que en primera instancia está vacío y se irán introduciendo en ellas los elementos que sean resultado de consultas o geoprocesamientos. Así, cuando se realizan consultas de datos y el resultado se muestra en el mapa, habitualmente, ese resultado se aloja en una capa gráfica.

Un número elevado de capas gráficas en una aplicación o demasiados elementos gráficos en una capa gráfica, puede provocar que el rendimiento de la aplicación sea menor ya que esto fuerza la aplicación a hacer mayor uso de memoria para mantener todos los elementos y mayor uso de CPU para renderizarlos.

Feature layers

Un *feature layer* es en realidad un tipo de *graphics layer*, es decir, la primera clase hereda de la segunda. La diferencia reside en que mientras que una capa gráfica siempre necesita de una clase intermedia que se comunique con el servidor para obtener información, por ejemplo una consulta "QueryTask", una *feature layer* sí puede comunicarse con el servidor para obtener datos que luego guarda en memoria en la aplicación. Podría decirse, por tanto, que se trata de una automatización de una *graphics layer*.

Estas capas son capaces de comunicarse con el servidor para traer información de forma automática, mientras que las capas gráficas visualizan datos normalmente a petición del usuario en base a las consultas o geoprocesamientos que se dispongan para tal efecto. Esta comunicación con el servidor se puede realizar de tres modos diferentes en función de la necesidad que se tenga:

- **Snapshot**: en cuanto la *feature layer* es cargada en el mapa, esta pide al servidor todos los elementos de la capa del mapa que se quiere renderizar.
- **OnDemand**: en este modo, la *feature layer* es capaz de solicitar al servidor aquellos elementos que coincidan con la extensión del mapa que se muestre en cada momento. Es decir, si el usuario navega por el mapa, automáticamente la *feature layer* irá solicitando al servidor los elementos que vayan correspondiendo.
- **Selection**: con este modo sólo se solicitan las elementos que el usuario seleccione. Suele utilizarse cuando la información es muy pesada y los otros modos no funcionan de forma óptima.

Mapas base locales

Es posible añadir un mapa base cacheado en local. Para ellos simplemente basta con generar una caché de ArcGIS Server e incluirla en la tarjeta SD del teléfono. De este modo, es posible minimizar el uso del ancho de banda disponible para la aplicación.

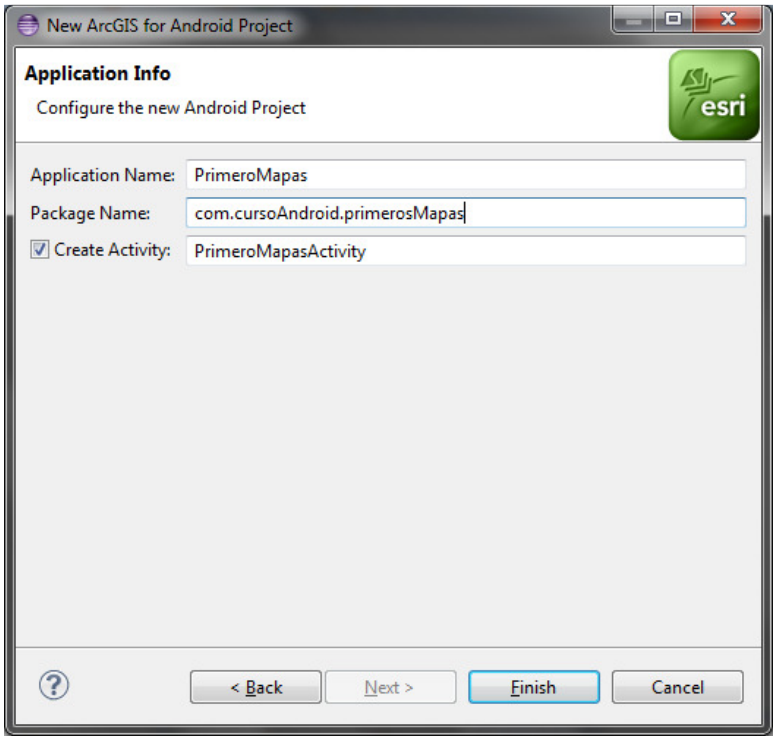
La clase que se necesita para poder realizar esta operación es ArcGISLocalTiledLayer y funciona igual que cualquier otro servicio con la diferencia de que el lugar de establecer un *url*, se establece la ruta donde está almacenada esa caché en la tarjeta SD.

4.2.- Ejercicio: Consumir diferentes tipos de mapa

Paso 1

En este ejercicio va a crear un proyecto con el Runtime SDK de ArcGIS para Android en el cual establecerá un mapa base y por encima un servicio de mapa dinámico. Por último establecerá una extensión inicial para los datos.

Cree un nuevo proyecto en Eclipse y seleccione el tipo de proyecto "ArcGIS Project for Android". De un nombre al proyecto y al paquete de su código. Además, acepte crear una actividad por defecto:



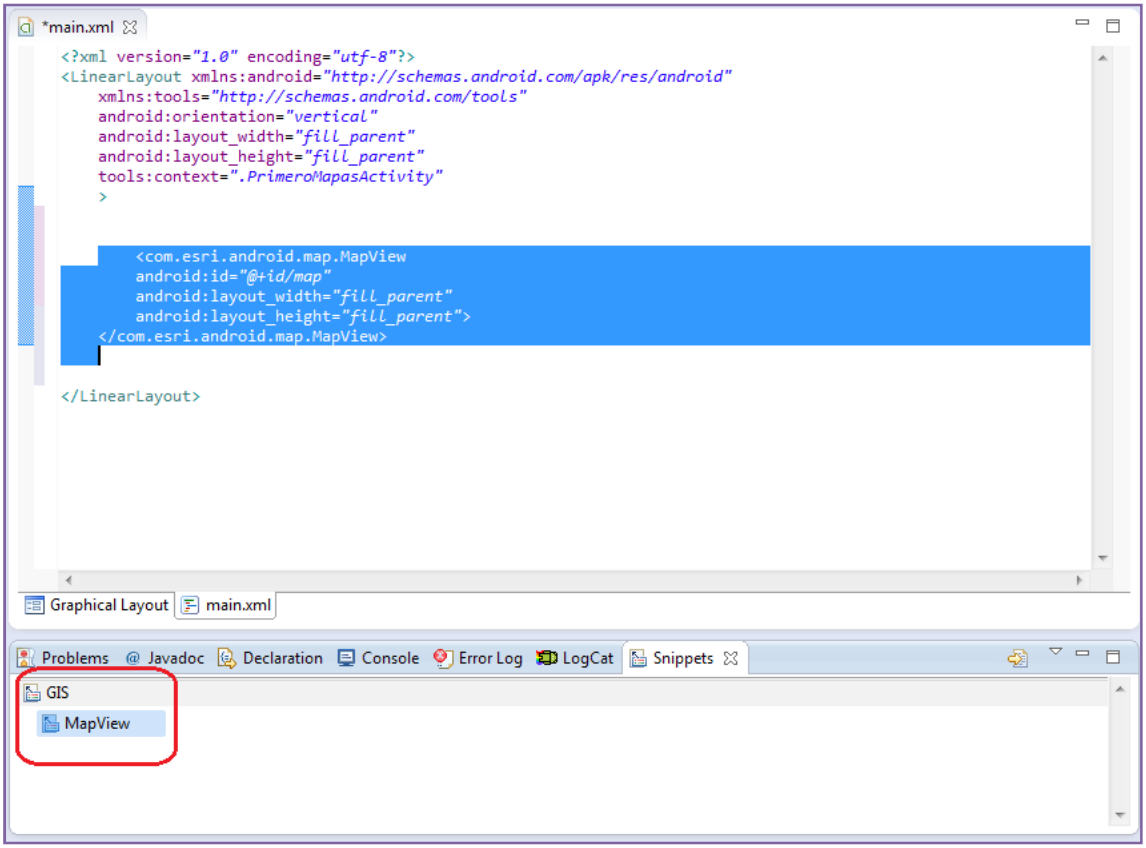
Paso 2

Una vez creado el proyecto, abra el fichero *layout* creado por defecto en la carpeta "res > layout" y acceda al código XML directamente en lugar de utilizar el editor gráfico.

Encontrará que el *layout* consta de un "LinearLayout" y, dentro, un único "TextView". Elimine este último y sustitúyalo por un "MapView" como este:

```
<com.esri.android.map.MapView
      android:id="@+id/map"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
</com.esri.android.map.MapView>
```

Debido a que no es posible en este momento añadir un “MapView”, que es la clase que representa un mapa, es habitual copiar y pegar este trozo de código. Una práctica útil es la creación de *snippets* en Eclipse. Un *snippet* es un de código que se almacena para utilizarlo de nuevo en el futuro. Para crear un *snippet*, seleccione el código en Eclipse y haga clic con el botón derecho sobre la selección. Haga clic en la opción “Add to Snippets…”. El almacenamiento de estos *snippets* se basa en categorías por lo que si no tiene ninguna, le pedirá que primero cree una y le dé un nombre a su nuevo *snippet*. Una vez realizado este paso, desde la ventana de “Snippets”, podrá añadirlo cómodamente arrastrándolo al editor:



Guarde los cambios y localice ahora el fichero con la *Activity* creada automáticamente. Verá que dentro de la clase hay un objeto “MapView” que en el método “onCreate” se instancia. No es necesario instanciarlo de este modo ya que lo ha definido en el *layout*, por lo que puede sustituirla por la siguiente línea de código:

```
mMapView = (MapView)findViewById(R.id.map);
```

La clase “MapView”, es una representación de un mapa en la interfaz gráfica, es decir, se trata de un componente que el usuario verá en su pantalla dentro de una *Activity*. Como ha visto en temas anteriores, debe tratarse por lo tanto de un objeto de tipo “View” y que podemos localizarlo mediante el método “findViewById()”. Además, en el *layout*, le hemos dado un identificador al que hemos llamado “map” y, como ha visto en temas anteriores, en la clase autogenerada “R” se almacenan todos los identificadores de todos los componentes del proyecto, haciendo posible obtener sus referencias en cualquier parte del mismo. En realidad, la clase “MapView” es un “GroupView”, un conjunto de elementos de tipo “View”.

Paso 3

Una vez que ha conseguido la referencia correcta al mapa, es necesario añadir capas al mismo ya que hasta el momento el mapa está vacío.

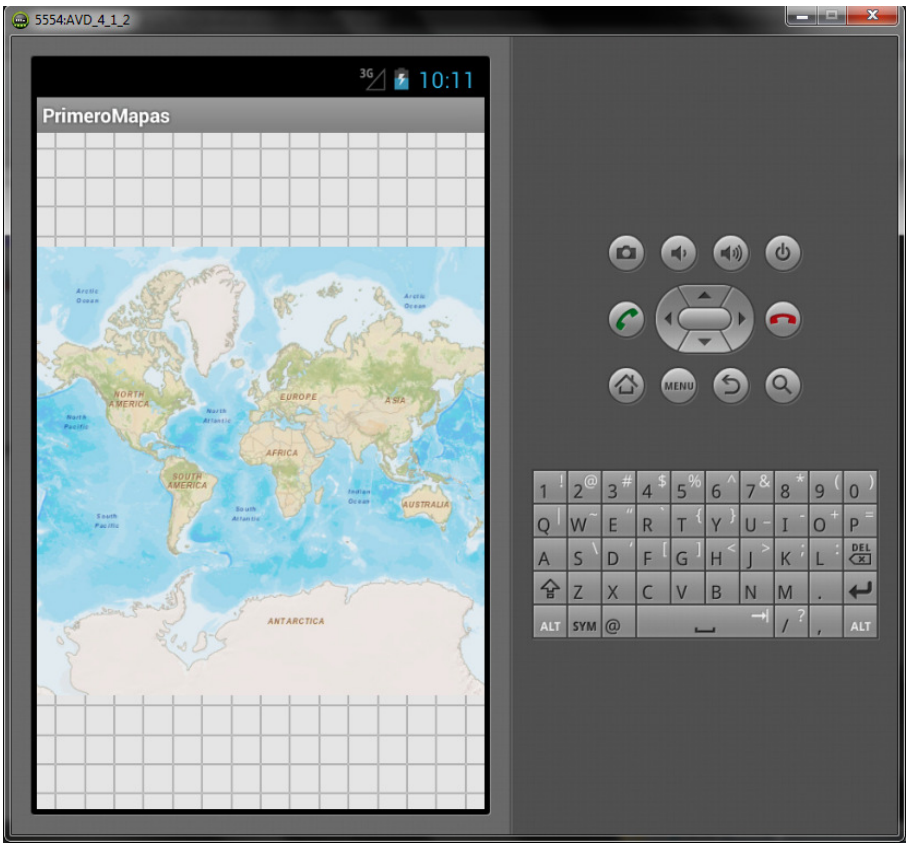
Primero creará una capa de un servicio de mapas cacheado y que utilizará como mapa base. Para ello añada la línea de código:

```
ArcGISTiledMapServiceLayer tiled = new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/serv
ices/World_Street_Map/MapServer");
```

No ha importado el paquete de la clase "ArcGISTiledMapServiceLayer" a su código por lo que le dará un error. Para solventarlo importe el paquete oportuno. Puede utilizar el atajo de teclado "CRTL + SHIFT + O". Por último añada esta capa al mapa:

```
mMapView = (MapView)findViewById(R.id.map);

ArcGISTiledMapServiceLayer tiled = new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/serv
ices/World_Street_Map/MapServer");

mMapView.addLayer(tiled);
```

Ejecute el proyecto en un emulador y obtendrá como resultado la visualización del mapa cacheado:



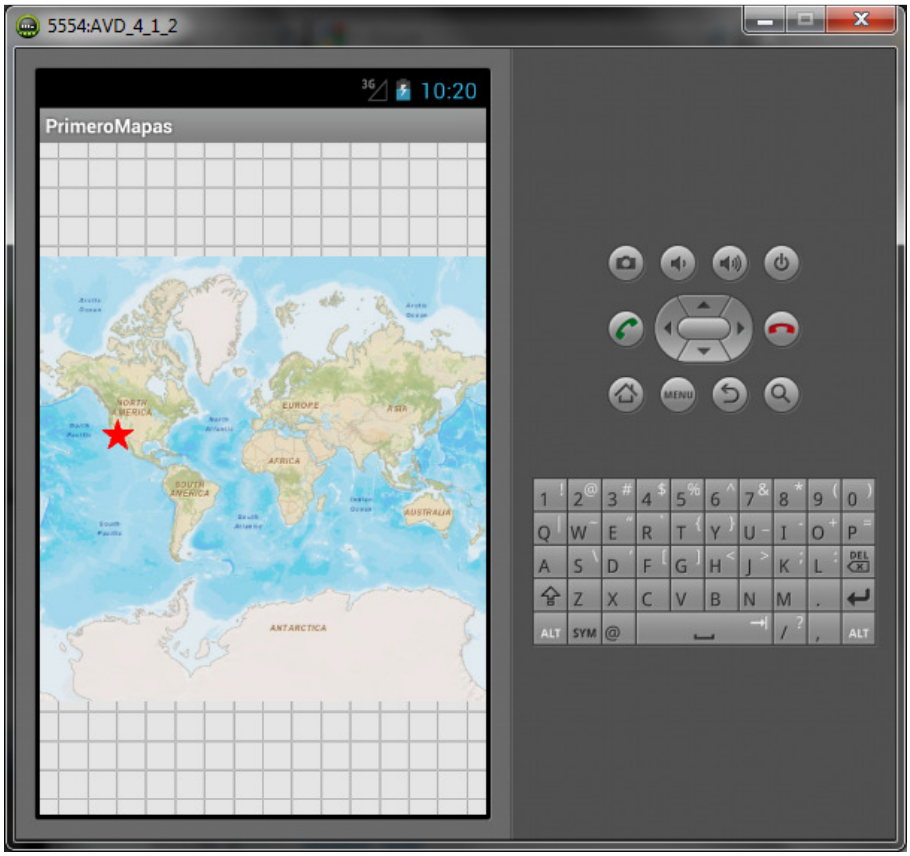
Paso 4

Ahora creará una capa de un servicio dinámico y la añadirá al mapa después del mapa cacheado de tal forma que se renderizará encima, ya que el proceso de renderizado es de abajo arriba por lo que el primer servicio añadido al mapa será el inferior y el último servicio añadido será el superior.

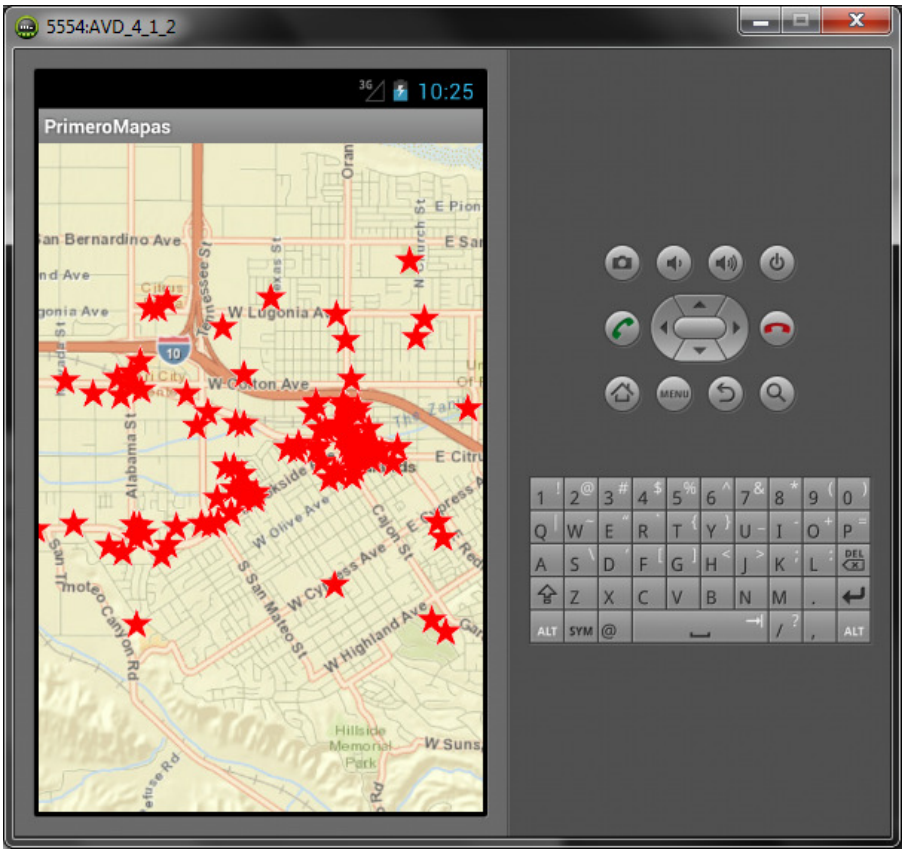
Cree un servicio dinámico del siguiente modo y ejecute de nuevo el proyecto.

```
ArcGISDynamicMapServiceLayer dynamic = new
ArcGISDynamicMapServiceLayer("http://trainingcloud.arcgis.com/ArcGIS/rest/ser
vices/Redlands_PointsOfInterest/MapServer");

mMapView.addLayer(dynamic);
```

La ejecución del proyecto en este momento debería superponer los dos mapas de este modo:



El segundo mapa que ha cargado representa una serie de puntos de interés de la ciudad de Redlands, California. Aunque pueda parecer un único punto, esto es debido a que el mapa se encuentra a una escala muy pequeña y los puntos están tan próximos entre sí que parecen un único elemento. Pero si se acerca lo suficiente, puede ir haciendo doble clic sobre el punto hacia el que quiere desplazarse, verá que en realidad hay un conjunto amplio de puntos:



Sería ideal por lo tanto establecer que nuestro "MapView" se iniciara en un *extent* que permitiera al usuario ver el mapa de este modo. Establecer este *extent* inicial es posible tanto desde el *layout* como desde el código Java.

Resulta más "limpio" realizar esta tarea desde el XML del *layout*:

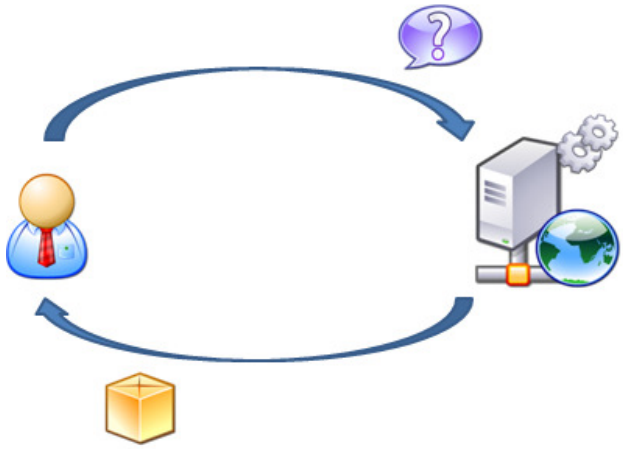
```
<com.esri.android.map.MapView
       android:id="@+id/map"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       initExtent = "-13040839.738 4038380.333 -13048483.441 4032647.556">
</com.esri.android.map.MapView>
```

Añada la línea al XML y ejecute el proyecto para obtener un mapa inicial como el visto anteriormente.

5.- Consultando los datos

5.1.- Consultas

En este tema comenzará a realizar consultas. Ya ha revisado el paradigma Cliente-Servidor para el caso de los mapas pero una consulta se realiza de igual modo, el usuario final desea obtener cierta información y la aplicación le provee de los mecanismos necesarios para consultarla con el servidor. De este modo, la consulta llegará al servidor, este la procesará y enviará una respuesta al usuario, normalmente en forma de "FeatureSet", un paquete que incluye un conjunto de elementos que cumplen con consulta del usuario:



Una consulta se compone de dos objetos con los que trabajar: un objeto "QueryTask" y un objeto "Query". El primero de ellos es el objeto del Runtime capaz de comunicarse con el servidor a través de la URL de la capa del mapa sobre la que se quiere lanzar la consulta. El segundo, el objeto "Query", es el encargado de almacenar el objeto de la consulta en sí, suele entenderse como el objeto que almacena la clausula "where" a aplicar sobre los datos para obtener información, aunque en realidad la consulta puede no ser una consulta clásica de tipo SQL sino que también puede tratarse de una consulta espacial.

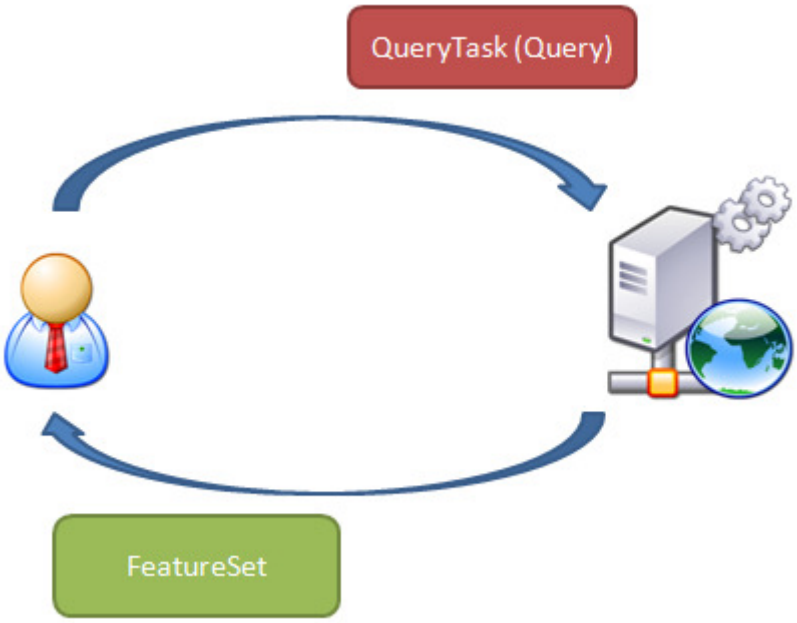
Hay varios métodos interesantes en la clase "Query" que es necesario tener en cuenta, y pueden consultarse en el API Reference, a la hora de lanzar una consulta:

- **setGeometry():** utilizado para realizar un filtro espacial
- **setOutFields():** utilizado para indicar al servidor que campos de la capa se quiere obtener en la respuesta de la consulta
- **setOutSpatialReference():** utilizado para indicar el sistema de referencia en el cual se desea que el servidor devuelva las geometrías de los elementos obtenidos
- **setReturnGeometry():** es necesario indicar al servidor si como respuesta a la consulta se requieren las geometrías de los elementos o no son necesarias las geometrías, necesitando tan sólo las información alfanumérica

- **setText():** en ArcGIS Server es posible especificar al publicar una capa, un campo por defecto, o *display field*, y este método permite realizar consultas de tipo SQL únicamente sobre ese campo, siendo estas óptimas pero no pudiendo consultar otros campos
- **setWhere():** para indicar la consulta SQL que se realizar sobre la capa

Una vez entendidos los objetos que se utilizan al realizar una consulta de tipo "Query", es importante conocer los pasos a realizar.

1. Establecer una interfaz de usuario para realizar las consultas. Por ejemplo, un botón que al ser pulsado, lance una consulta contra el servidor.
2. Implementar la "QueryTask" con la URL de la capa a consultar.
3. Implementar el objeto "Query" con las propiedades necesarias para la consulta.
4. Implementar una clase que extienda la clase de Android "AsyncTask" y que permita la ejecución asíncrona de consultas.



5.2.- Renderers

Un objeto *symbol* es una simbología que se aplica a una geometría. Por ejemplo, una coordenada, un punto, puede representarse como un triángulo rojo, en este caso, sería necesario crear un *symbol* con esas propiedades. Por el contrario, un *renderer* es una forma de aplicar simbologías a una capa gráfica.

Cuando realizamos una consulta, el servidor devuelve un "FeatureSet", un conjunto de elementos gráficos, a los cuales hay que aplicarles una simbología para que puedan ser renderizados en el mapa. Por lo tanto, un elemento gráfico, todo aquello que se dibuja en el mapa, se compone de una geometría, que es implícita del elemento, y una simbología que el desarrollador de la aplicación le da.

Es posible recorrer los elementos del "FeatureSet" que devuelve el servidor y uno a uno otorgarles una simbología para que sean renderizados, o de forma más sencilla, aplicar un *renderer* a la capa gráfica a la que serán añadidos dichos elementos.

Básicamente, en una capa gráfica es posible introducir 3 tipos diferentes de geometrías: puntos, líneas y polígonos. Pero un *renderer*, sólo puede aplicarse a un tipo de geometría, es decir, un *renderer* instanciado para simbolizar líneas y que sea aplicado a una capa gráfica en la cual hay puntos, este los ignorará y no serán renderizados en el mapa.

Hay 3 tipos de *renderer*:

- ***SimpleRenderer:*** todos los elementos son renderizados con la misma simbología.
- ***UniqueValueRenderer:*** a partir de un campo de los elementos gráficos, es posible dar diferentes simbologías para cada uno de los diferentes valores que tenga cada uno de ellos. Por ejemplo, en una capa de puntos de interés, cada tipo de punto (Banco, Recinto deportivo, Restaurante,..) tendrá una simbología diferente.
- ***ClassBreaksRenderer:*** para un campo numérico, gracias a este *renderer*, es posible establecer rangos de valores y simbolizar cada rango con diferentes simbologías. Por ejemplo, simbolizar la población en las diferentes comunidades autónomas en base a su población.

5.4. - *Callouts*

Los *callouts* permiten mostrar información de forma sencilla relacionada con los datos del mapa. Son a Android lo que los *popups* a la web.

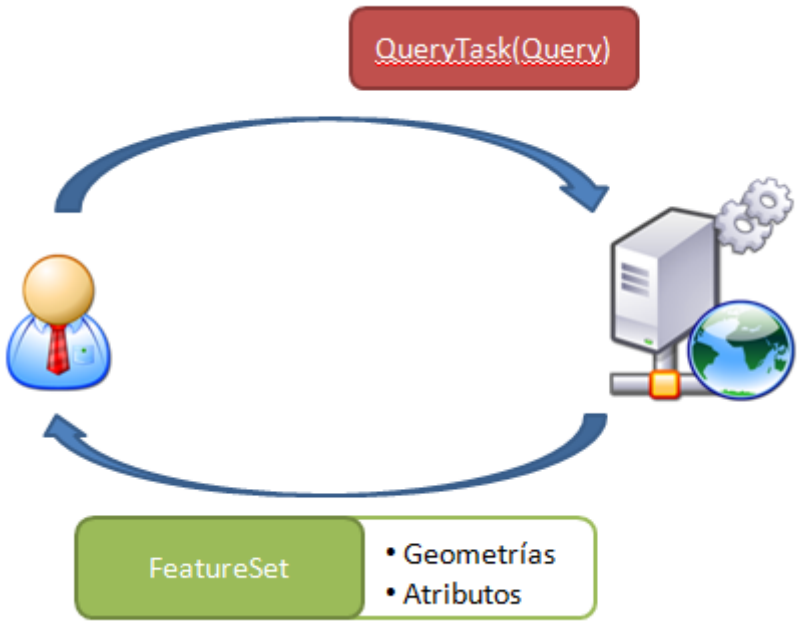
La mayor particularidad de este componente del Runtime SDK es que debido al funcionamiento interno de Android, hay que tener en cuenta que un *callout* necesariamente tendrá que ser una "View" y, por lo tanto, necesitaremos tanto definir el *layout* de esa vista que queremos tener y también un fichero que defina el estilo del *callout*.

Además, habrá que definir la forma mediante la cual el usuario final indicará a la aplicación que quiere obtener información de un elemento, normalmente, haciendo un único clic, mejor dicho, un único "tap" en la pantalla.

Debe tener en cuenta que para la información debe estar disponible en local para que funcione con una velocidad adecuada, esto quiere decir que los datos que se muestren en el *callout*, debe estar en una capa gráfica o bien una *feature layer*.

Al definir el *callout* mediante un *layout* de Android, es posible hacerlo con todas las características que esto incluye como añadir imágenes personalizadas en base a los atributos de cada elemento.

Por tanto, revisando el gráfico descrito anteriormente, podemos revisarlo añadiéndole un nivel más de profundidad:



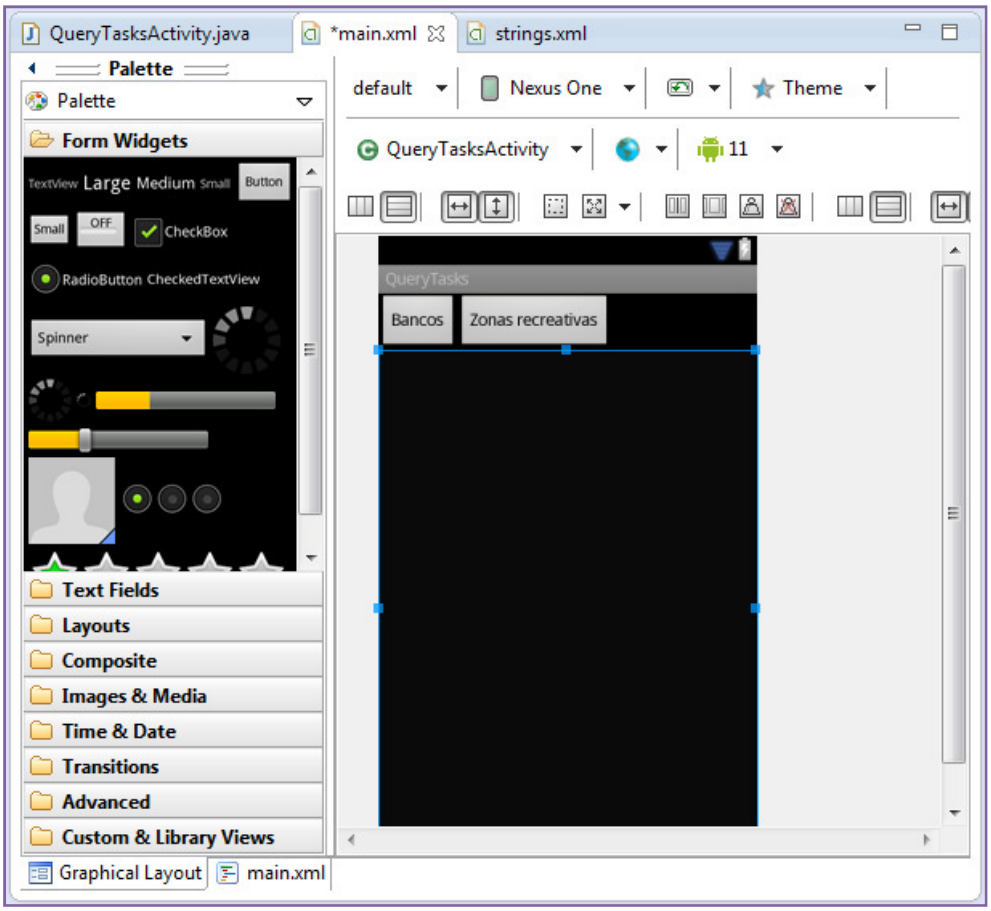
5.5.- Ejercicio: *QueryTask*

Paso 1

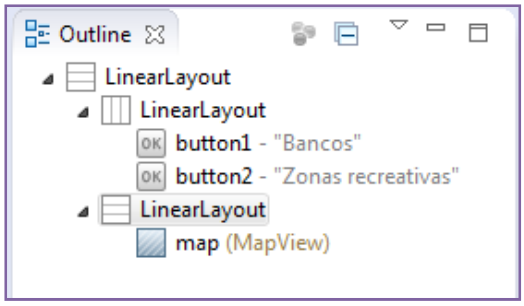
En este ejercicio va a realizar consultas a una capa alojada en ArcGIS Server de tipo QueryTask. El objetivo es que la aplicación muestre un mapa y dos botones. Cada uno de los botones lanza una consulta diferente sobre una capa de puntos que en principio no es visible para el usuario. De esta forma, cuando el usuario haga clic en el primer botón, lanzará una consulta de tipo SQL para obtener los puntos que representan bancos, mientras que el segundo botón mostrará al usuario los puntos que representen zonas recreativas.

Para ello, cree un nuevo proyecto en Eclipse de ArcGIS para Android como ha aprendido en temas anteriores.

Lo primero que ha de hacer es crear una interfaz de usuario. Abra el *layout* principal para modificarla. Para este ejercicio necesita crear 2 botones en la parte superior de la vista y un "MapView" debajo de ellos. Desde el propio editor proporcionado por el ADT, es posible dividir el elemento "Layout" creado por defecto incluyendo en él otros 2 objetos "Layout", el primero horizontal y el segundo vertical. Incluya en el primero de ellos, dos botones. La apariencia de la interfaz será como esta:



Los textos de los botones debe incluirlos como aprendió anteriormente en el fichero de recursos strings.xml de tal modo que el *outline* de la interfaz quedará así:



Recuerde que el mapa lo salvó como un snippet en un ejercicio anterior. Además, los datos que utilizará pertenecen al área de Redlands, California, por lo que puede utilizar el *extent* inicial que ya utilizó anteriormente, quedando finalmente el *layout* de este modo:

```
<?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:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context=".QueryTasksActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/banks_button" />
        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="top"
            android:text="@string/recreative_button" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
        <com.esri.android.map.MapView
             android:id="@+id/map"
             android:layout_width="fill_parent"
             android:layout_height="fill_parent"
             initExtent = "-13040839.738 4038380.333 -13048483.441
4032647.556">
             </com.esri.android.map.MapView>
    </LinearLayout>

</LinearLayout>
```

Paso 2

Una vez definida la interfaz vaya al código Java de la actividad creada por defecto. En el método "onCreate" recupere el "MapView" y añada un mapa base:

```
// Obtenemos el MapView
mMapView = (MapView) findViewById(R.id.map);

// Añadimos un mapa base al mapa
ArcGISTiledMapServiceLayer tiled = new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/MapServer");

mMapView.addLayer(tiled);
```

Recuerde que debe importar los paquetes necesarios para las clases que vaya utilizando (CTRL + SHIFT + O).

Añada ahora dos miembros de la clase además del "MapView" que ya tiene creado por defecto:

```
MapView mMapView ;
GraphicsLayer gl;
String targetServerURL =
"http://trainingcloud.arcgis.com/ArcGIS/rest/services/Redlands_PointsOfInterest/MapServer/0";
```

El objeto de tipo "GraphicsLayer" será la capa gráfica donde mostrará el resultado de sus consultas, mientras que la cadena de texto es la URL donde se encuentra la capa de puntos de interés sobre la que realizará las mismas.

La capa gráfica, al igual que el resto de capas que añade al mapa, será instanciada en el método "onCreate" de la actividad pero es necesario crear un *renderer* para ella. Comience por crear un "SimpleRenderer", es decir, un *renderer* que aplicará a todos los elementos una simbología. Esta simbología también necesita ser creada y, en este caso, utilizará un círculo rojo de 10 puntos de tamaño:

```
// Creamos una capa grafica
gl = new GraphicsLayer();
// Para la capa grafica necesitamos un renderer.
// En este caso necesitamos un renderer de puntos ya que en esta capa
añadiremos puntos de interes de redlands puntuales
SimpleRenderer sr = new SimpleRenderer(new SimpleMarkerSymbol(Color.RED, 10,
SimpleMarkerSymbol.STYLE.CIRCLE));
gl.setRenderer(sr);
// La añadimos al mapa
mMapView.addLayer(gl);
```

Paso 3

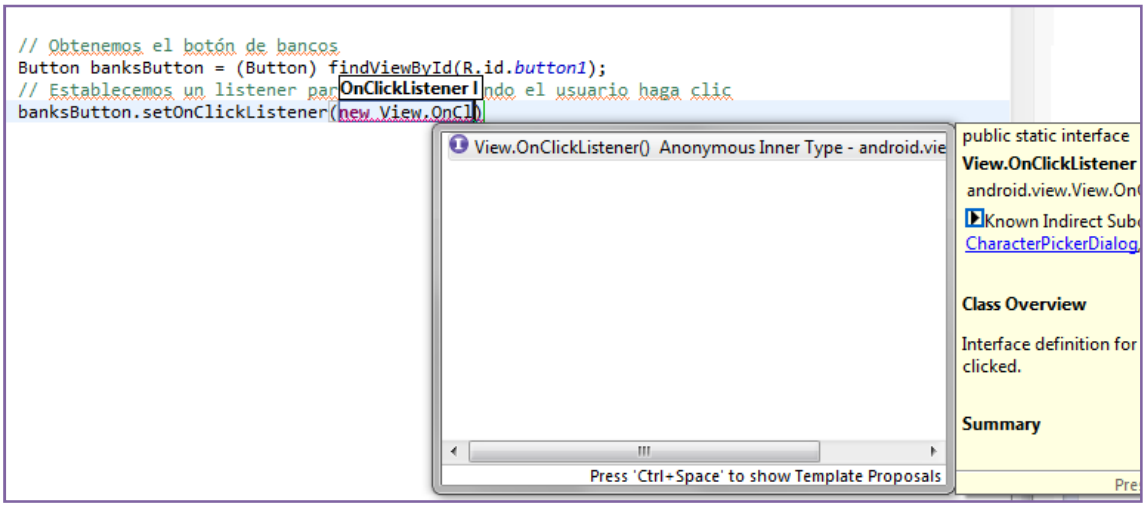
El mapa está preparado para su uso. Ahora necesita capturar cuándo hace el usuario clic en los diferentes botones para así realizar las consultas correspondientes.

La forma más sencilla de hacer esto, es utilizando *listeners*. La clase "View" proporciona mecanismos sencillos de este estilo para facilitar la captura de los eventos que suceden cuando el usuario interactúa con la aplicación.

En este caso, necesitará un *listener* para el evento "onClick" del botón pero primero necesitará obtener el botón del mismo modo que ha obtenido el mapa:

```
// Obtenemos el botón de bancos
Button banksButton = (Button) findViewById(R.id.button1);
// Establecemos un listener para capturar cuando el usuario haga clic
banksButton.setOnClickListener(new View.OnClickListener() {

	public void onClick(View v) {
		// TODO Auto-generated method stub

	}
});
```

Aunque el código no parece muy intuitivo, Eclipse genera la mayor parte del mismo. Pruebe a escribir parte del código y verá cómo Eclipse le va indicando qué debe escribir:



Si no lo consigue, pida ayuda a su profesor. El código generado quedará así:

```
// Obtenemos el botón de bancos
Button banksButton = (Button) findViewById(R.id.button1);
// Establecemos un listener para capturar cuando el usuario haga clic
banksButton.setOnClickListener(new View.OnClickListener() {

	public void onClick(View v) {
		// TODO Auto-generated method stub

	}
});
```

Paso 4

Llegado a este punto, tiene una aplicación con un mapa y un botón para el cual captura el evento clic (puede repetir el paso anterior para el otro botón) y en ese evento clic, debe enviar al servidor una consulta concreta.

La forma más sencilla de hacer tareas asíncronas, es aprovechar una clase de Android llamada "AsyncTask" y que es genérica, es decir, se diseño precisamente para que fuera utilizada en cualquier tipo de proyecto. Esto hace que sea de fácil implementación una vez que se conoce su funcionamiento.

La clase genérica "AsyncTask", espera 3 tipos en su implementación:

- *Params*: los tipos de los parámetros a enviar en la tarea
- *Progress*: el tipo de la unidad de progreso si el progreso de la tarea necesita ser publicado de algún modo
- *Result*: el tipo del resultado de la tarea

En este caso, los parámetros que necesita la consulta son de tipo *string*, ya que será únicamente la URL de la capa y la clausula *where* de tipo SQL para realizar la consulta. Por lo tanto, fuera del método "onCreate" pero dentro de la actividad, puede anidar una clase privada que extienda "AsyncTask" con estas condiciones:

```
private class AsyncQueryTask extends AsyncTask<String, Void, FeatureSet>
{

}
```

En este punto, el compilador le indicará que es necesario que implemente el método abstracto de la clase "AsyncTask" llamado "doInBackground(String…)". Incluso le aparecerá una opción, si pone el ratón encima de la clase recientemente creada "AsyncQueryTask", dándole la opción de implementarlo ("Add unimplemented methods"). Pinche en esa opción y obtendrá lo siguiente:

```
private class AsyncQueryTask extends AsyncTask<String, Void, FeatureSet>
{
	@Override
	protected FeatureSet doInBackground(String... params) {
		// TODO Auto-generated method stub
		return null;
	}
}
```

Eclipse ha implementado por usted el método pero fíjese que el tipo de parámetros del método es "String…" y el tipo de retorno del mismo es "FeatureSet" tal y como había indicado en la declaración de la clase. Es posible suponer en este momento que dicho "FeatureSet" será el conjunto de elementos que devolverá el servidor tras procesar la consulta, y "Strings…" serán los parámetros necesarios para realizar la consulta, en este caso, la URL y la clausula *where*.

La clase “AsyncTask” permite pasar del hilo central en la UI a un hilo en *background* que permite la ejecución de tareas asíncronas sin la necesidad de realizar laboriosas tareas de *threading*.

Realmente, cuando se realiza una tarea de tipo “AsyncTask” de este modo, se llevan a cabo 4 pasos aunque, como ha comprobado, sólo uno de ellos es de obligatoria implementación:

1. **onPreExecute():** este método es llamado inmediatamente después de ejecutar la tarea y suele utilizarse para darle los valores necesario y para establecer una barra de progreso.
2. **doInBackground():** es el método central en el cual se debe procesar la tarea. Además, de recibir los parámetros de entrada, debe desarrollarse de tal modo que el retorno del método sea del tipo indicado para la clase, en este caso, de tipo “FeatureSet”.
3. **onProgressUpdate():** este método se utiliza mientras la tarea está en progreso para actualiza su estado, por ejemplo, actualizando una barra de progreso o escribiendo información en un fichero *log*.
4. **onPostExecute():** tras terminar el método central, se ejecuta este método, recibiendo como parámetro el resultado de la tarea, es decir, el valor que devuelve el método central. En este caso, recibe el “FeatureSet” y puede utilizarse para añadir los elementos gráficos que contiene a la capa gráfica.

Teniendo esto en cuenta, puede comenzar a desarrollar el método **doInBackground()** comprobando que los parámetros son correctos y almacenándolos:

```
protected FeatureSet doInBackground(String... params)
      {
            // Comprobamos que la llamada es adecuada y tiene al menos los
parametros URL y clausula where
            if (params == null || params.length <= 1)
                  return null;

            // Los recogemos en strings
            String url = params[0];
            String whereClause = params[1];
```

El siguiente paso, es crear el objeto “Query” y darle la propiedades necesarias:

```
// Creamos un objeto Query
Query query = new Query();
// Le damos los parametros necesarios aplicando la clausula where
//    y solicitando que el servidor devuelva geometrias en el sistema de
referencia que deseamos
SpatialReference sr = SpatialReference.create(102100);
query.setOutSpatialReference(sr);
query.setReturnGeometry(true);
query.setWhere(whereClause);
```

En el caso actual, es necesario que el servidor devuelva las geometrías de los elementos ya que es lo que se desea mostrar en el mapa y se le aplica la clausula *where* que ha llegado como parámetro.

El siguiente paso es crear el objeto “QueryTask” utilizando la URL recibida y ejecutarlo, obteniendo así el “FeatureSet” con los elementos que cumplan con la consulta realizada.

```
// Creamos el objeto QueryTask utilizando la URL proporcionada
QueryTask qTask = new QueryTask(url);
FeatureSet fs = null;
try
{
	// Tratamos de ejecutar la tarea recogiendo el FeatureSet con los
elementos que concuerden con la consulta realizada
	fs = qTask.execute(query);
}
catch (Exception e)
{
	// Si hay algun error los enviamos a la consola
	System.out.println(e.getMessage());
}

// Devolvemos el FeatureSet recibido de la ejecucion de la consulta
return fs;
```

Con esto el método central de la ejecución quedaría resuelto. Puede aprovechar el método “onPostExecute” para obtener los elementos gráficos contenidos en el “FeatureSet” y añadirlos a la capa gráfica creada anteriormente:

```
protected void onPostExecute(FeatureSet result)
{
	if (result != null)
	{
		// Si el FeatureSet devuelto por el servidor no es nulo
comprobamos si contiene elementos y los añadimos a la capa grafica
		Graphic[] grs = result.getGraphics();
		if (grs.length > 0)
		{
			gl.addGraphics(grs);
		}
	}
}
```

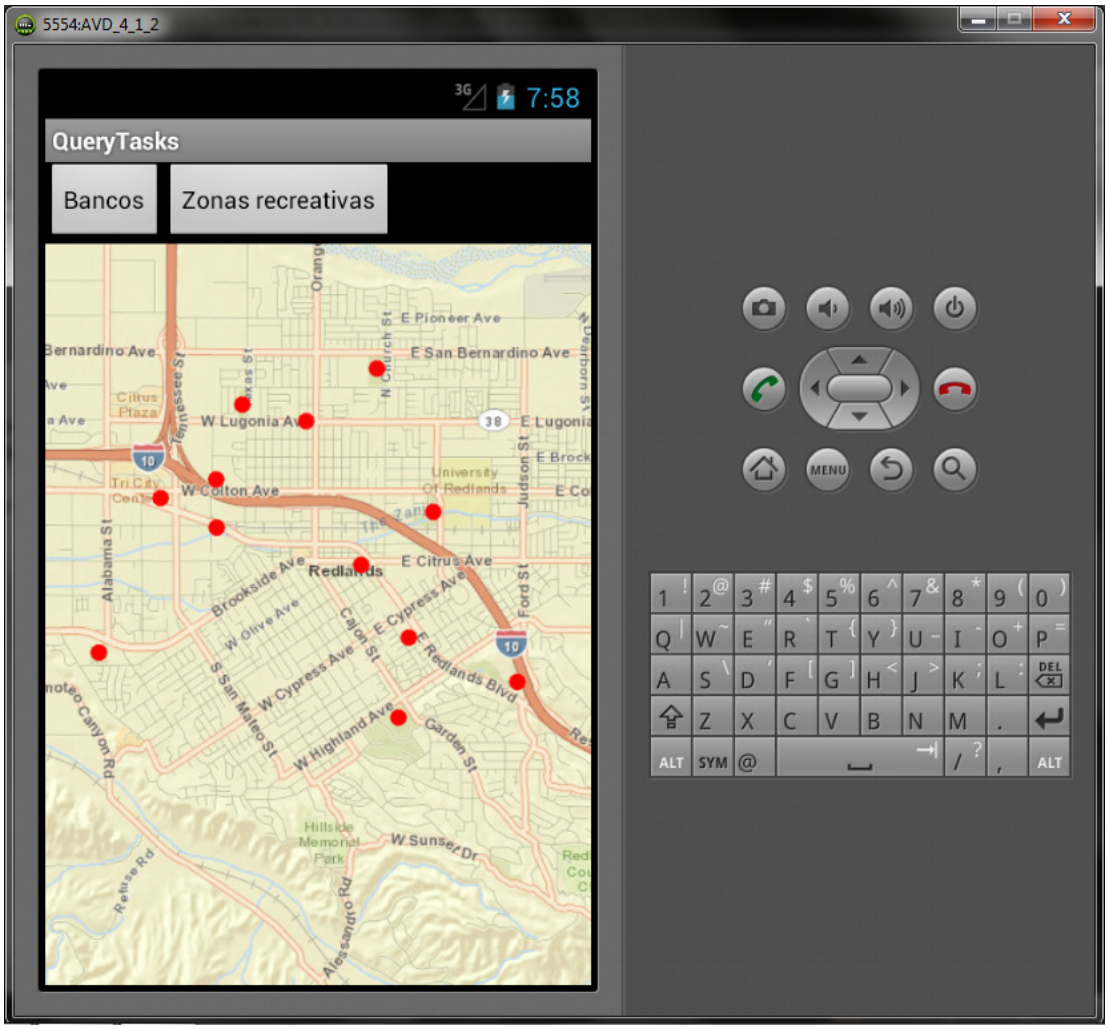
Una vez implementada la clase que le permite realizar tareas asíncronas de consulta, localice el *listener* del botón que ya tenía creado y trate de completarlo.

Primero, utilice el método “removeAll()” de la capa gráfica para que cada vez que haga clic en alguno de los botones se borren los resultados de anteriores ejecuciones. Después, cree un *array* de tipo *string* con los parámetros que necesita la consulta, la URL y cláusula. Y, por último, instancie un nuevo objeto de su clase “AsyncQueryTask” y ejecútelo con el *array* de parámetros.

```
// Obtenemos el botón de bancos
Button banksButton = (Button) findViewById(R.id.button1);
// Establecemos un listener para capturar cuando el usuario haga clic
banksButton.setOnClickListener(new View.OnClickListener() {

	public void onClick(View v) {
		// Limpiamos los elementos de la capa grafica
		gl.removeAll();
		// Establecemos como parametros la URL y la clausula where
		String[] queryParams = { targetServerURL, "TYPE='Bank'" };
		// Creamos una tarea asincrona y la lanzamos con los parametros
		AsyncQueryTask ayncQuery = new AsyncQueryTask();
		ayncQuery.execute(queryParams);
	}
});
```

Realice el mismo procedimiento para el botón de las zonas recreativas cambiando el parámetro “TYPE=’Bank’” por “TYPE=’Recreation’”.

Ejecute el programa y compruebe su funcionamiento.



5.6.- Ejercicio: Aplicar un *renderer* a una FeatureLayer

Paso 1

En este ejercicio va a realizar un uso más intenso de la simbología y utilizará el tipo de capa gráfica "FeatureLayer" que le permitirá recibir información del servidor sin necesidad de realizar consultas asíncronas.

El objetivo del ejercicio es cargar sobre un mapa base una capa de puntos de interés a la cual se aplicará un *renderer* de tal modo que, en base al valor que tenga cada elemento en un determinado campo, se simbolizará de determinado modo.

Cree un proyecto ArcGIS en Eclipse con un "MapView" en la *Activity* principal.

Después, añada un mapa base de ArcGIS Online a su mapa.

Al igual que ha creado anteriormente mapas cacheados y dinámicos mediante código Java, cree ahora una "FeatureLayer". El constructor de la clase "ArcGISFeatureLayer" necesita 2 parámetros:

- La URL donde se aloja la capa. Utilice la capa de puntos de interés del ejercicio anterior.
- El modo de la "FeatureLayer". De los 3 estudiados anteriormente, utilice el modo "SNAPSHOT".

Puede investigar en el API Reference como hacer esto. En este ejercicio contará con el tiempo suficiente para revisar la documentación ya vista y los nuevos conceptos.

No añada esta capa al mapa, lo hará al final del ejercicio.

Paso 2

Ahora creará el *renderer*, en este caso de tipo "UniqueValueRenderer". Después de instanciar un objeto de este tipo, utilice sus siguientes métodos:

- setField1(): con este método, se establece el campo que utilizará para discriminar a los elementos con diferentes simbologías. En este caso, discriminará por el campo "TYPE".
- setDefaultSymbol(): necesitará crear una simbología por defecto por si algún elemento no cumple con las especificaciones que hará más adelante en base al campo anterior. En el ejercicio anterior, ya creó una simbología para un *renderer*, utilícela.

Un "UniqueValueRenderer" debe almacenar una lista de objetos de tipo "UniqueValue" de tal forma que cada uno de estos asocie un valor del campo a determinada simbología. Puede basarse en la siguiente relación:

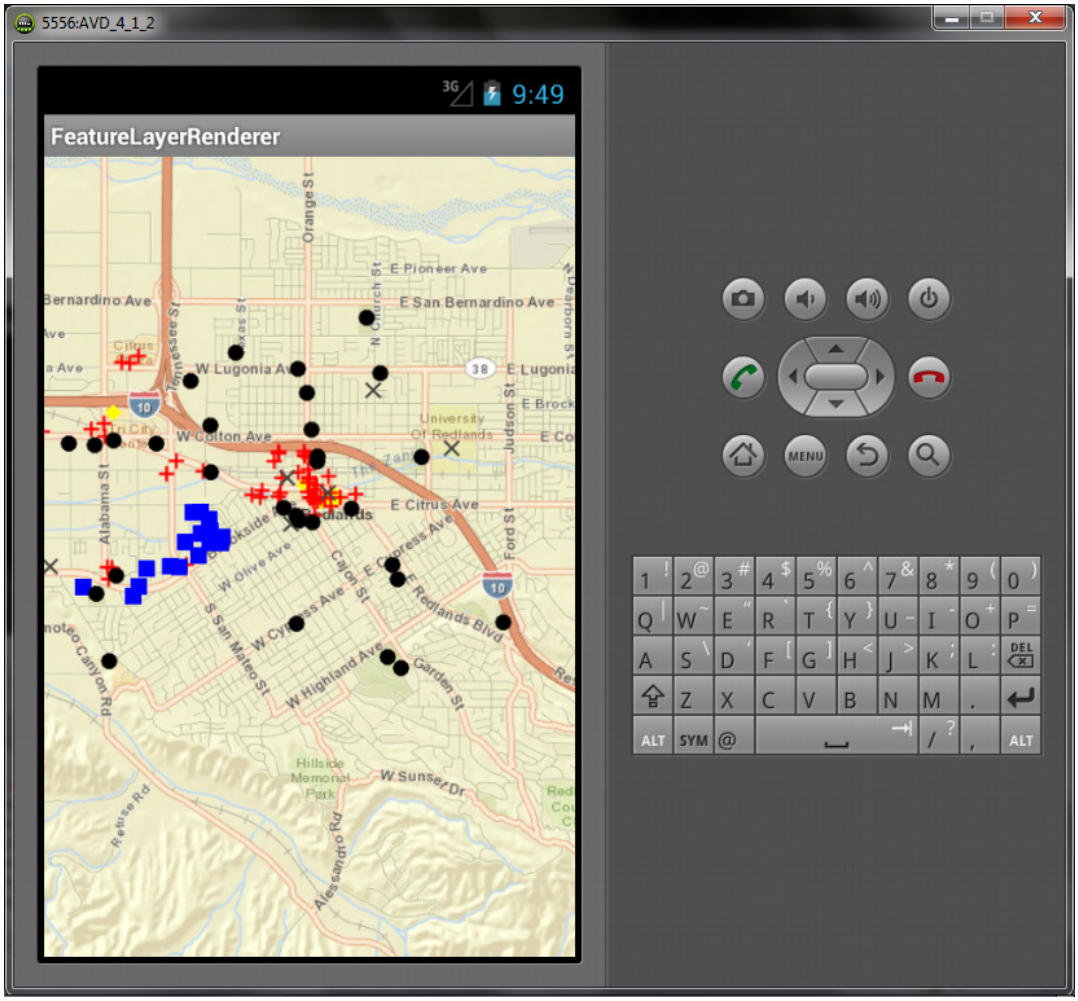
UniqueValue	Valor del campo TYPE	Simbología
uniqueValue1	Bank	Color amarillo, 10p, Diamond
uniqueValue2	Recreative	Color verde, 10p, Circle
uniqueValue3	Restaurant	Color rojo, 10p, Cross
uniqueValue4	Apartment	Color azul, 10p, Square
uniqueValue5	Arts	Color gris oscuro, 10p, X

Siguiendo esta tabla, deberá crear 5 objetos de tipo “UniqueValue”, establecer su valor y su simbología. Después, deberá añadirlos todos ellos al *renderer* utilizando el método “addUniqueValue()” o el método “setUniqueValueInfos”. Trate de hacerlo con ambos métodos.

Paso 3

Por último, aplique a la “FeatureLayer” el *renderer* y añádala al “MapView”.

El resultado debería ser el siguiente:

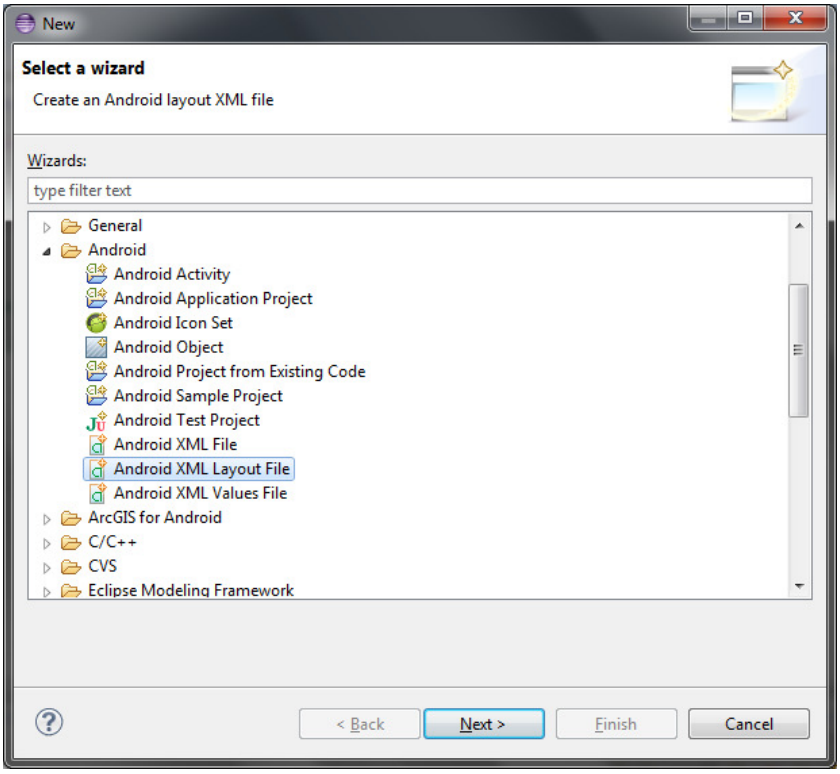


5.7.- Ejercicio: *Callouts*

Paso 1

Basándose en el ejercicio anterior, va a añadir a los elementos que ya se están mostrando, un callout que muestre información de sus atributos. En concreto, mostrará información de los campos "NAME" y "TYPE" de la capa de puntos de interés.

Como primer paso, puede crear el *layout* para la vista del *callout*, es decir, debe definir los elementos que se mostrar en el *callout* y su disposición. Haga clic con el botón derecho sobre el proyecto en el *Package Explorer* y pinche sobre "**New > Other**". En la ventana que le aparecerá, indique que desea crear un nuevo *layout*:

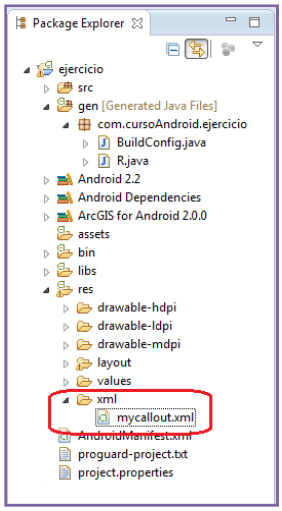


En la siguiente ventana, seleccione que desea crear un *layout* con forma de "LinearLayout" y dele un nombre. El *layout* que va a definir es muy sencillo ya que sólo se desea mostrar los valores de dos atributos de cada elemento. Por lo tanto, añada dos "TextView", uno debajo del otro y salve los cambios.

Paso 2

Además de un *layout*, un *callout* necesita un estilo visual. Para ello es necesario crear un fichero XML de recursos.

Cree una carpeta nueva dentro de la carpeta "res" de su proyecto y llámela "xml":



Edite fichero y cree un estilo como este:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <calloutViewStyle
   titleTextColor="#000000"
   titleTextSize="10"
   titleTextStyle="0"
   titleTextTypeFace="0"
   backgroundColor="#ffffff"
   backgroundAlpha="255"
   frameColor="#000000"
   flat="true"
   anchor="5" />
</resources>
```

Cuando termine el ejercicio, pruebe a modificar los valores del estilo y consultar la documentación para ver cómo cada parámetro afecta al estilo final de la interfaz.

Paso 3

Una vez definido completamente cómo será el *callout*, es necesario definir cómo y cuándo mostrarlo al usuario.

Al final del método "onCreate()" de su actividad principal, añada un *listener* para el evento "OnSingleTap" que indica que el usuario ha tocado una única vez la pantalla. Además, el método proporciona la coordenada donde ha tocado la pantalla. Deje que Eclipse le ayude a crear como hizo anteriormente:

```
mMapView.setOnSingleTapListener(new OnSingleTapListener()
{
	public void onSingleTap(float x, float y)
	{

	}
});
```

Como buena práctica, puede añadir una línea al comienzo del método “onSingleTap” para comprobar si el mapa ha sido cargado en el momento en el cual el usuario toca la pantalla para evitar así algún problema:

```
if (!mMapView.isLoaded())return;
```

Utilice ahora el método de la capa “getGraphicIDs()” pasando como parámetros la coordenada XY recibida y una tolerancia de dos pixeles para obtener un *array* con los elementos más cercanos al punto donde sucede el clic del usuario.

Si el *array* contiene elementos entonces deberá llamar más adelante al *callout*, pero si no los tiene, entonces querrá que se cierre, si está abierto:

```
int[] uids = featureLayer.getGraphicIDs(x, y, 2);
if (uids != null && uids.length > 0)
{

}
else
{
	if (callout != null && callout.isShowing())
	{
		callout.hide();
	}
}
```

Como el *array* está ordenado, el primer elemento del *array* será el que el usuario ha tratado de marcar.

Del objeto “MapView” es posible obtener el *callout* actual mediante el método “getCallout()”. Obténgalo y aplíquele el estilo definido:

```
// Obtenemos el primer elemento gráfico
Graphic gr = featureLayer.getGraphic(uids[0]);

// Obtenemos el callout del mapview
callout = mMapView.getCallout();

// Ahora le aplicamos el estilo definido en el XML
callout.setStyle(R.xml.mycallout);
```

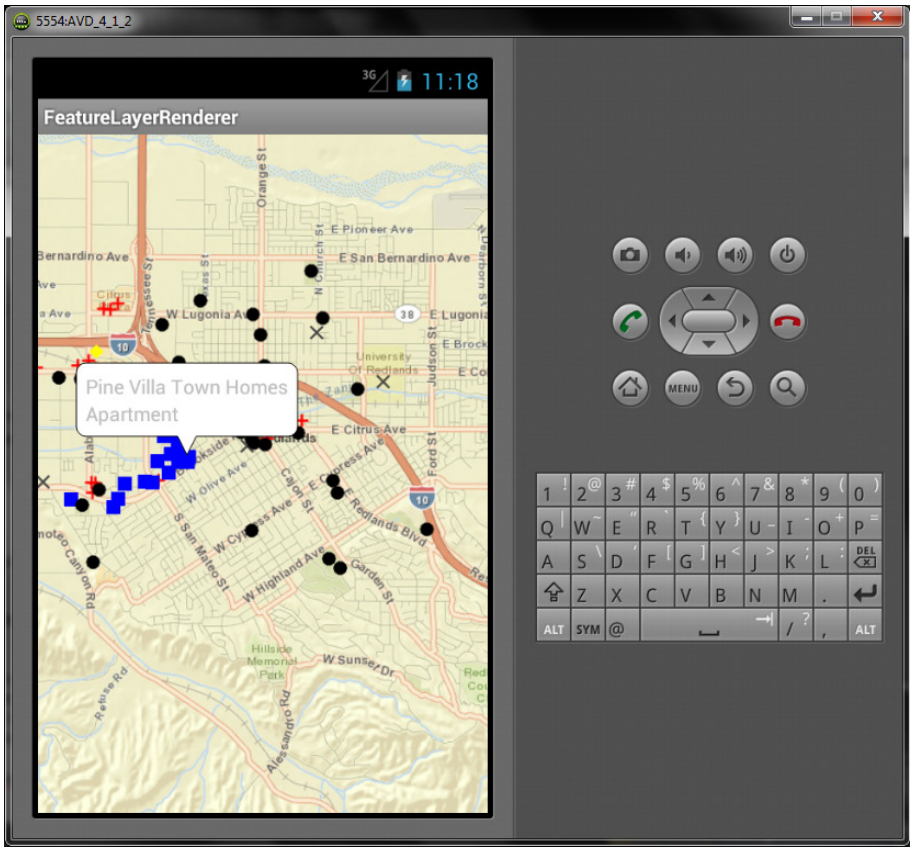
En este momento, tiene el *callout* con la forma del *layout* definido pero, sin embargo, este *callout* está vacío, es decir, aunque está preparado para albergar los elementos y la disposición correcta, aún no puede mostrarse al usuario debido a que no hay ningún objeto “View” creado para este propósito. Para crear una nueva vista con la forma de un determinado *layout*, puede utilizar la clase “LayoutInflater” que crea una vista en base al contexto del dispositivo. Es importante hacerlo de este modo ya que la vista se creará con las propiedades oportunas en cada dispositivo.

Una vez creada la vista necesaria para el *callout*, habrá que recoger los objetos de tipo "TextView" de la misma, en el *layout* definión que hubiera dos, y establecer los valores de los atributos.

Por último, necesitará establecer esa vista como el contenido del *callout* y mostrarlo en el mapa en el punto donde hizo clic el usuario:

```
// Una vez que tenemos el callout, necesitamos añadir el contenido
// Obtenemos los atributos del elemento gráfico
String poi_name = (String) gr.getAttributeValue("NAME");
String poi_type = gr.getAttributeValue("TYPE").toString();

// Utilizamos el inflater para crear una vista adecuada
View view =
LayoutInflater.from(FeatureLayerRendererActivity.this).inflate(R.layout.mylay
out, null);

// Obtenemos los 2 textviews que definimos en el layout
// Y les damos los valores correspondientes
TextView name = (TextView) view.findViewById(R.id.textView1);
name.setText(poi_name);
TextView number = (TextView) view.findViewById(R.id.textView2);
number.setText(poi_type);

// Aplicamos la vista
callout.setContent(view);
callout.show(mMapView.toMapPoint(new Point(x, y)));
```

Salve los cambios y ejecute el programa:

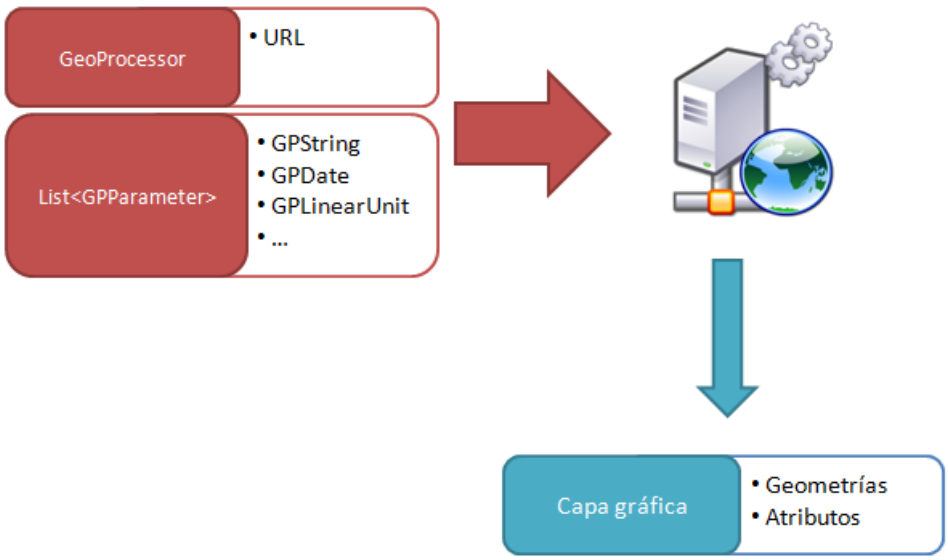


6.- Geoprocesamiento y geolocalización

6.1.- Geoprocesamiento

La ejecución de geoprocesamientos es un pilar básico en el desarrollo web. Sin embargo, en el desarrollo de aplicaciones para dispositivos móviles, debido a las limitaciones de los mismos, no es un concepto muy extendido aunque el Runtime SDK de ArcGIS para Android permite hacerlo tanto de forma síncrona como asíncrona.

Como ya ha estudiado en temas anteriores, el resultado de un geoprocesamiento así como de una consulta, lo renderizará en una capa gráfica en local. El funcionamiento es muy similar al visto anteriormente al realizar consultas. De nuevo, utilizará un objeto de tipo "Task", es decir, capaz de comunicarse con el servidor donde se aloja el geoprocesamiento y utilizará un objeto de tipo "Parameter", es decir, quien realmente almacena los parámetros del geoprocesamiento. En el caso de las consultas, el objeto de tipo "Task" era "QueryTask" y el objeto de tipo "Parameter" era "Query".

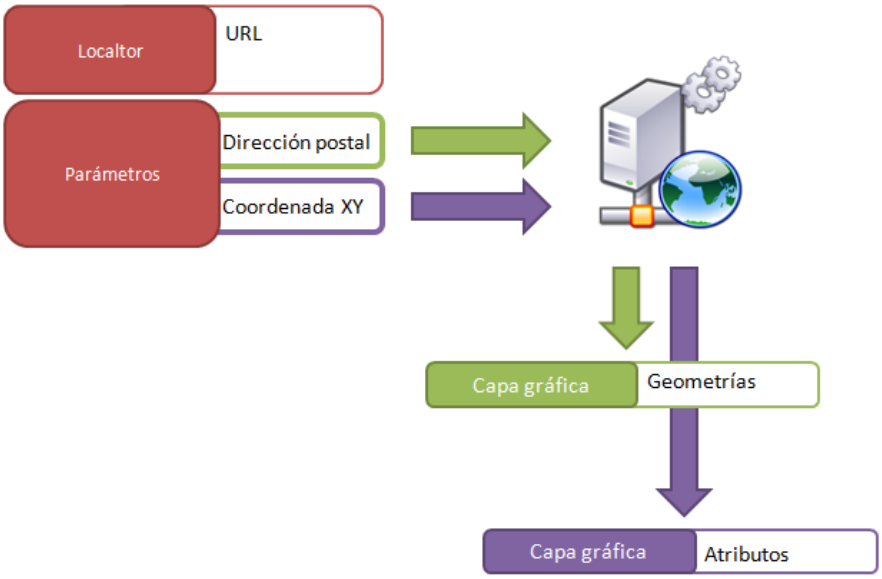


6.2.- Geolocalización

La geolocalización es un aspecto básico en dispositivos móviles, siendo una de las funcionalidades más apreciadas por el usuario final.

La geolocalización, al contrario que el resto de tareas, puede ejecutarse de dos modos diferentes: directa e inversa. Normalmente, la primera es conocida simplemente por geolocalización y consiste en localizar un punto geográfico de un mapa partiendo de una dirección postal dada por el usuario, mientras que la segunda, consiste en obtener la dirección postal en base a un punto señalado por el usuario o el GPS.

En ambos casos el funcionamiento es semejante, sólo cambiando los parámetros que se envían y se reciben en función de si se desea un geolocalización o una geolocalización inversa.



6.3.- Ejercicio: Geolocalización inversa

Paso 1

La geolocalización inversa es una de las funcionalidades más apreciadas por el usuario final. En base a un punto del mapa que el usuario solicite, creará una aplicación capaz de indicarle la dirección postal de dicho punto.

Comience creando un nuevo proyecto de ArcGIS en Eclipse y añada al *layout* un mapa con el *extent* inicial de Redlands, utilizado en ejercicios anteriores.

Además, en el código Java, añada un mapa cacheado de ArcGIS Online y una capa gráfica con un *renderer* simple. Puede consultar los ejercicios anteriores para realizar estas tareas.

Paso 2

Una vez que tenga el mapa preparado, debe construir la forma en la que el usuario se comunicará con la aplicación. En este caso, el usuario hará un "SingleTap" en algún punto del mapa para obtener

Para ello, cree un *listener* del mismo:

```
mMapView.setOnSingleTapListener(new OnSingleTapListener() {

	public void onSingleTap(float x, float y) {
		// TODO Auto-generated method stub

	}
});
```

Como buena práctica, primero elimine de la capa gráfica todos los elementos de resultados anteriores.

Como parámetros está llegando la coordenada XY del punto de la pantalla que tocó el usuario, utilice un método de la clase "MapView" visto anteriormente para convertir una coordenada de la pantalla a una del mapa. Necesitará esta coordenada del mapa como parámetro de entrada para el localizador.

Instancie ahora un nuevo objeto *locator*, puede crearlo al igual que el mapa y la capa gráfica como un miembro de la clase. Necesitará la URL de un localizador de ArcGIS Server o de ArcGIS Online:

```
locator = new
Locator("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Locators/
ESRI_Geocode_USA/GeocodeServer");
```

Para lanzar el localizador, además del punto del mapa indicado por el usuario y que ya tiene, necesitará el sistema de referencia del mapa y una tolerancia para realizar la búsqueda:

```
// Necesitaos el sistema de referencia del mapa para el localizador
SpatialReference sr = mMapView.getSpatialReference();

// Obtenemos el resultado de la consulta
LocatorReverseGeocodeResult result = locator.reverseGeocode(loc, 1000.00,
sr, sr);
```

De este modo, logrará ejecutar la geolocalización inversa enviando como parámetros el punto, una tolerancia de 1000 metros y el sistema de referencia adecuado para su mapa.

Trate de comprobar si el resultado de la geolocalización es correcto o no. Si lo es, mostrará en el *callout* del mapa, los atributos recibidos del localizador. Para mostrar información en el *callout*, necesita algún elemento de interfaz gráfica, por ejemplo, un "TextView". Puede crear un método que utilizará más adelante que convierta una cadena de texto en un "TextView":

```
private TextView createTextView(String text)
{
	final TextView msg = new TextView(this);
	msg.setText(text);
	msg.setTextSize(12);
	msg.setTextColor(Color.BLACK);
	return msg;
}
```

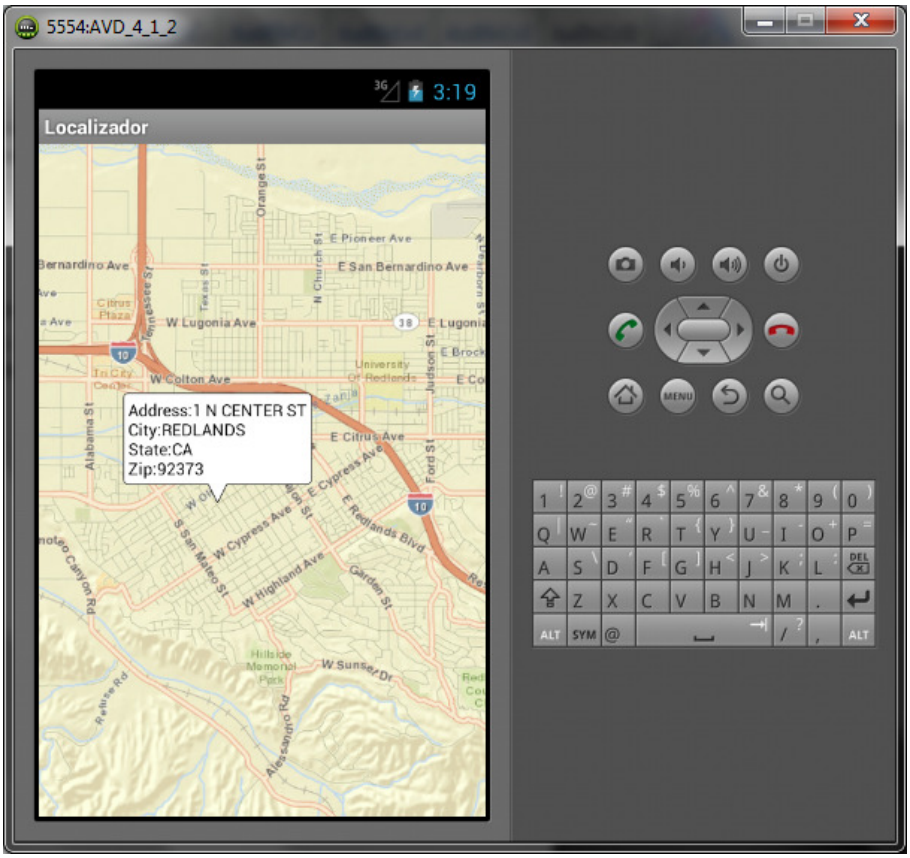
Puede guardar este método como un *snippet* de Eclipse.

Por último, obtenga del resultado de la geolocalización los atributos, que para este localizador son: "Address, City, State, Zip". Puede comprobarlo accediendo a la URL del mismo.

Una vez los tenga, puede pasarlos al *callout* del mapa utilizando el método anterior:

```
// Si todo va bien, creamos un string con los cuatro campos
String msg = "Address:"+ result.getAddressFields().get("Address") + "\n"
                    + "City:" + result.getAddressFields().get("City") + "\n"
                    + "State:"+ result.getAddressFields().get("State") + "\n"
                    + "Zip:" + result.getAddressFields().get("Zip");

mMapView.getCallout().show(loc, createTextView(msg));
```

Ejecute el programa y compruebe los resultados:



Paso Opcional

Cree un estilo personalizado para el *callout* en lugar de utilizar el *callout* por defecto.

Paso Opcional

Modifique el valor de la tolerancia de la geolocalización hasta llegar a un valor que le parezca equilibrado.

6.4.- Ejercicio: Cuencas visuales

Paso 1

Un geoprocesamiento es muy semejante en su ejecución a una consulta. En este ejercicio realizará una aplicación que ejecuta un geoprocesamiento de cuencas visuales alojado en un servidor de ArcGIS Online. Dicho geoprocesamiento requiere como parámetro de entrada un punto, en realidad un "FeatureSet" de puntos aunque sólo tenga un elemento, y una distancia. Como resultado, obtendremos una capa con la visibilidad que hay desde ese punto en el radio indicado por la distancia.

Al igual que en el ejercicio anterior cree un proyecto y siga los siguientes pasos:

1. Añada el mapa al *layout*.
2. En su *Activity*, obtenga el mapa y añada un mapa base y una capa gráfica aunque ahora no le aplique un *renderer*.
3. Suscríbase al evento "SingleTap" utilizando un *listener*.
4. En dicho evento, elimine los elementos del resultado anterior de la capa gráfica.

Paso 2

Para facilitar al usuario la comprensión de la tarea, añadirá al mapa el punto que él mismo ha tocado:

```
// Obtenemos el punto y creamos un elemento grafico con el
mappoint = mMapView.toMapPoint(x, y);
Graphic g = new Graphic(mappoint, new SimpleMarkerSymbol(Color.RED, 10,
STYLE.CIRCLE));

gLayer.addGraphic(g);
```

Paso 3

Ahora es necesario crear los parámetros necesarios para ejecutar el geoprocesamiento. Este geoprocesamiento requiere dos parámetros de entrada:

Número	Nombre	Tipo
1	Input_Observation_Point	GPFeatureRecordSetLayer
2	Viewshed_Distance	GPLinearUnit

El primer parámetro es un "GPFeatureRecordSetLayer", es decir, un conjunto de elemento. Este tipo de parámetro es genérico y se utiliza muy habitualmente para utilizar varios elementos al mismo tiempo aunque en este caso sólo es necesario el punto del mapa que toca el usuario.

Por lo tanto, debe instanciar el tipo de parámetro utilizando su nombre y cargar en él el elemento gráfico que es el punto creado:

```
// Primer parametro para el GP, un GPFeatureRecordSetLayer
GPFeatureRecordSetLayer gpf = new
GPFeatureRecordSetLayer("Input_Observation_Point");
gpf.setSpatialReference(mMapView.getSpatialReference());
gpf.setGeometryType(Geometry.Type.POINT);
// Añadimos el punto del usuario al featurerecordset
Graphic f = new Graphic(mappoint, new
SimpleMarkerSymbol(Color.RED,25,STYLE.DIAMOND));
gpf.addGraphic(f);
```

El segundo parámetro, es un "GPLinearUnit", es decir, un parámetro que indica una distancia pero además, la unidad de medida en la cual se debe tomar esa distancia:

```
// Segundo parametro, un GPLinearUnit
GPLinearUnit gpl = new GPLinearUnit("Viewshed_Distance");
gpl.setUnits("esriMeters");
gpl.setDistance(1000);
```

Por último, sería necesario crear una lista con todos los parámetros del geoprocesamiento antes de lanzarlo:

```
// Creamos una lista de parametros
params = new ArrayList<GPParameter>();
params.add(gpf);
params.add(gpl);
```

Paso 4

Antes de poder lanzar el geoprocesamiento, es necesario crear una nueva clase que extienda "AsyncTask" al igual que sucedía con las consultas de tal modo que pueda ser ejecutado de forma asíncrona.

```
class ViewShedQuery extends AsyncTask<ArrayList<GPParameter>, Void,
GPParameter[]>
{
            GPParameter[] outParams = null;
}
```

Revise la documentación y el ejercicio de consultas para revisar los siguientes puntos:

- Los 3 tipos de parámetro que se indican a "AsyncTask" en la declaración de la clase. Trate de justificar por qué se ha definido la clase de este modo.
- Los 4 métodos que pueden ser sobrescritos al heredar de la clase "AsyncTask".

Consulte con su profesor si tiene alguna duda al respecto.

Paso 5

Va a sobrescribir el método "doInBackground". En este método, recibe como parámetro de entrada una lista con los parámetros, es decir, con el *recordset* que contiene el punto del mapa y con la distancia a utilizar.

En este método necesita crear el geoprocesamiento y ejecutarlo con los parámetros.

```
// Instanciamos un nuevo geoprocessor y le damos el sistema de referencia
gp = new
Geoprocessor("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Elev
ation/ESRI_Elevation_World/GPServer/Viewshed");
gp.setOutSR(mMapView.getSpatialReference());

// Ejecutamos el GP y obtenemos una lista de resultados, nos quedamos el
primero
GPResultResource rr = gp.execute(params1[0]);
outParams = rr.getOutputParameters();
```

Compare este método con el método que implemento en el ejercicio de consultas y verifique el funcionamiento es el realmente el mismo.

Asegúrese de que el método devuelve el resultado, es decir, la variable "outParams".

Paso 6

Una vez ejecutada la tarea, utilizará el método "onPostExecute" para mostrar en el mapa el resultado obtenido.

Al implementar este método, recibirá como parámetro el resultado del método anterior. Compruebe en primera instancia que no es nulo y de ser así, devuelva un valor nulo.

El resultado del geoprocesamiento es de tipo "GPFeatureRecordSetLayer" al igual que uno de los parámetros de entrada. Es necesario por lo tanto, recorrer todos los posibles elementos que haya en dicho *recordset* y añadirlos a la capa gráfica:

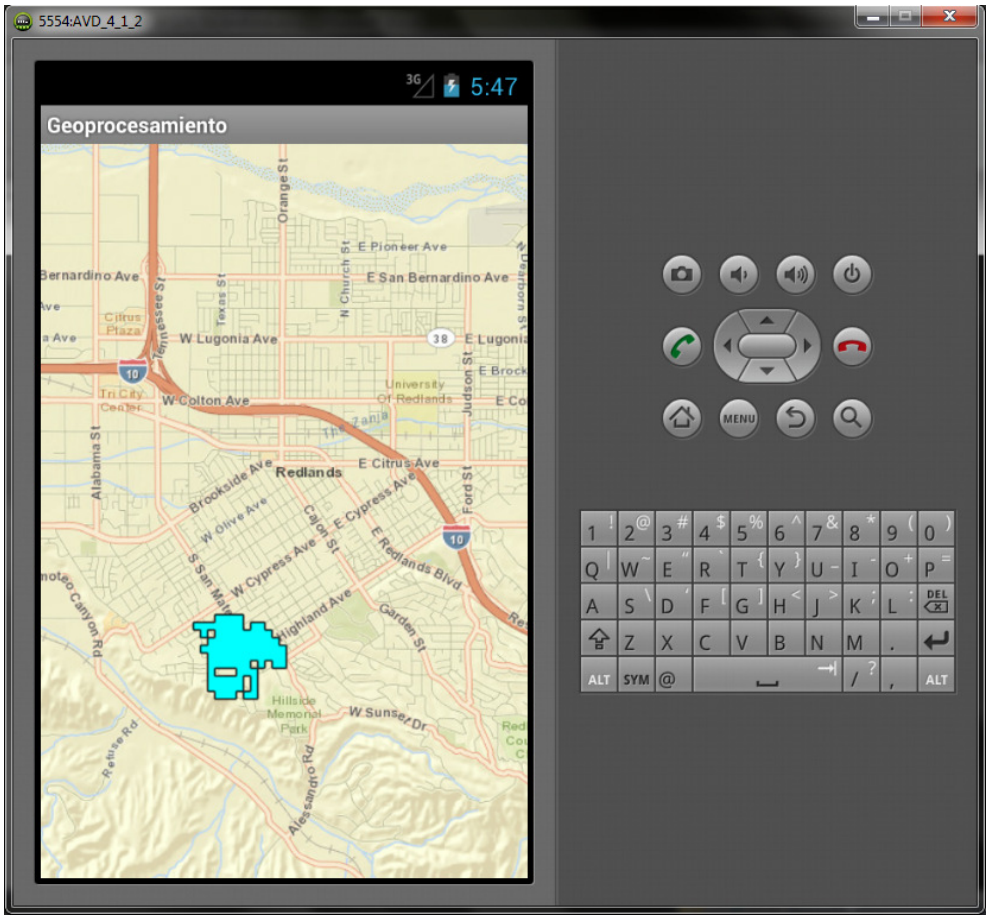
```
// Una vez obtenida la capa de salida, obtenemos todos los elementos
graficos y los añadimos a la capa
GPFeatureRecordSetLayer fsl = (GPFeatureRecordSetLayer) result[0];
for (Graphic feature : fsl.getGraphics())
{
      Graphic g = new Graphic(feature.getGeometry(),new
SimpleFillSymbol(Color.CYAN));
      gLayer.addGraphic(g);
}
```

Paso 7

Terminada la clase que permite la ejecución asíncrona, sólo le falta, ejecutar un nuevo objeto de dicha clase con los parámetros creados:

```
new ViewShedQuery().execute(params);
```

Ejecute la aplicación y compruebe el resultado.



Modifique el valor del parámetro de distancia y compruebe los cambios que suceden.

7.- Edición

7.1. – Edición

La edición mediante el Runtime SDK de ArcGIS para Android no es tan avanzada como puede ser una edición web y mucho menos completa que una edición en ArcGIS Desktop. Esto se debe a las limitaciones que tiene un dispositivo móvil.

Sin embargo, es posible editar datos siempre que las ediciones sean relativamente sencillas. Se puede distinguir tres casos de edición aceptados por el Runtime SDK: creación de nuevas geometrías, modificación de geometrías existentes y edición de atributos. Es posible además, unir todas ellas en una sola aplicación.

Creación de nuevas geometrías

Antes de desarrollar una aplicación que cree nuevas geometrías es necesario pensar primero en la forma en la que el usuario se comunicará con la aplicación. Es posible que el usuario necesite editar varias capas y por lo tanto necesite escoger primero cuál de ellas es la que necesita. O bien es posible que sea el GPS del dispositivo el que indique a la aplicación la localización de la nueva geometría. En cualquier caso, es necesario establecer unos pasos previos a la edición.

Además, si la edición se da a través de un servicio de edición de ArcGIS Server (*feature service*), podemos desarrollar la aplicación de tal modo que utilice las plantillas (*templates*) que el encargado de diseñar el mapa decidió establecer, esto incluye no sólo las simbologías o *renderer* sino los valores por defecto a establecer en los atributos de los nuevos elementos que se van creando.

Una vez definida la forma en la que se va a proceder con la edición, en el desarrollo, habrá que localizar los eventos del mapa necesarios y establecer un *workflow* lo más sencillo posible para que el usuario se sienta cómodo con la edición. Puede consultar todos los eventos del mapa disponibles en el API Reference del Runtime SDK.

Por último, es necesario enviar las ediciones realizadas en la aplicación al servidor. Ya que la edición se hace en local, a efectos de desarrollo esto significa que se dará en una capa gráfica, en particular, en una *feature layer*. Y el modo que se debe utilizar para que la comunicación cliente-servidor sea correcta será el modo por defecto, *on demand*. Y el envío de las ediciones, se realizará utilizando el método “applyEdits()” de la capa gráfica y esperar la respuesta del servidor.

Edición de geometrías existentes

En el caso de la modificación de geometrías existentes es todavía más crítico el hecho de establecer un *workflow* de trabajo correcto ya que un usuario inexperto puede encontrar dificultades en su trabajo. Los tres pasos a establecer son:

1. Seleccionar la geometría que se desea modificar
2. Establecer las herramientas necesarias para realizar el tipo de edición que se desea (añadir vértices, dividir parcelas, cortar parcelas,…)
3. Enviar la edición al servidor

De nuevo, para establecer este tipo de edición utilizará una *feature layer* pero en este caso con el modo "selection". Por si sólo, este modo no muestra ningún elemento hasta que no se selecciona; para mostrar al usuario dónde se encuentran las geometrías para poder seleccionarlas y recibirlas en su dispositivo, puede utilizar un mapa dinámico debajo de su capa gráfica. La selección se hare mediante un objeto "Query" que realiza un filtro geográfico, por ejemplo, a partir del punto que marca el usuario.

La edición de geometrías, sobre todo de polígonos, puede ser relativamente complicada pero puede llevarse a cabo utilizando la clase "GeometryEngine" que permite realizar este tipo de operación en el mismo dispositivo, sin ser necesario realizar constantes peticiones a un servicio de geometría alojado en el servidor como suele ser habitual en aplicaciones web.

Debido a las limitaciones de conexión que puede tener un dispositivo móvil, se ha desarrollado el Runtime SDK de este modo, facilitando en gran medida el trabajo de campo. De este modo, el "GeometryEngine", provee operaciones geométricas basadas en grandes *arrays* de geometrías como *clips*, *buffers*, uniones,… La unión de este tipo de operaciones y las operaciones de creación de nuevas geometrías forman una completa edición geométrica en la aplicación.

Edición de atributos

Es importante a la hora de permitir a los usuarios la edición de atributos en un dispositivo móvil, pensar en el pequeño tamaño de las pantallas y en la dificultar de introducir valores.

La edición de atributos, en ocasiones, no tiene por qué ser completa, es decir, podemos encontrar casos en los cuales el usuario no debe ser capaz de editar todos los datos. Detectados los campos editables, puede determinar el tipo de valor que almacena para presentar al usuario la interfaz adecuada (campos numéricos con decimales, fechas, desplegables,…).

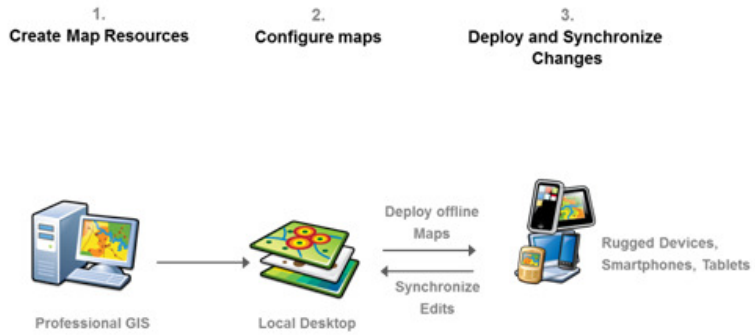
Por lo tanto, durante la edición de atributos, debe realizar una inspección de los campos de la capa a editar de tal forma que se mantenga la integridad de la edición sin errores.

Recuerde antes de realizar un “applyEdits()”, comprobar la integridad de los datos y deteniendo el *commit* si los valores introducidos por el usuario no concuerdan con los valores aceptados por los campos.

7.2. – Opciones de edición

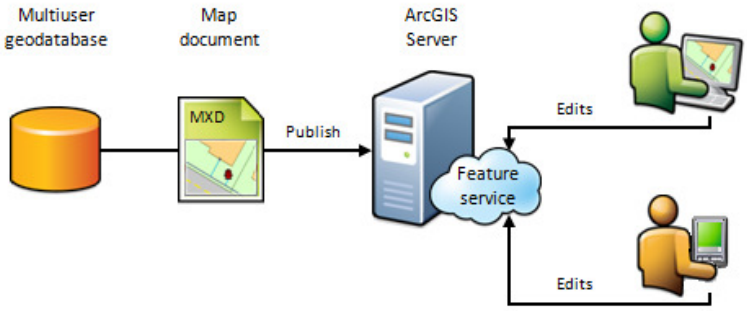
La edición en dispositivos móviles puede darse de dos modos básicos: conectado y desconectado.

El modo desconectado se basa en la idea de salir a trabajar a campo con un dispositivo móvil sin la necesidad de que este acceda en tiempo real a los datos alojados en el servidor. Este modo es muy útil cuando se trata de entornos en los cuales el dispositivo móvil no tiene cobertura o cuando no se dispone de una conexión a internet, por ejemplo, 3G.



Tras realizar las tareas de campo, al volver a la oficina el usuario del dispositivo móvil es capaz de sincronizar sus datos con la base de datos del servidor.

El otro modo, sería un modo conectado, es decir, el dispositivo realiza las ediciones directamente sobre los datos del servidor:

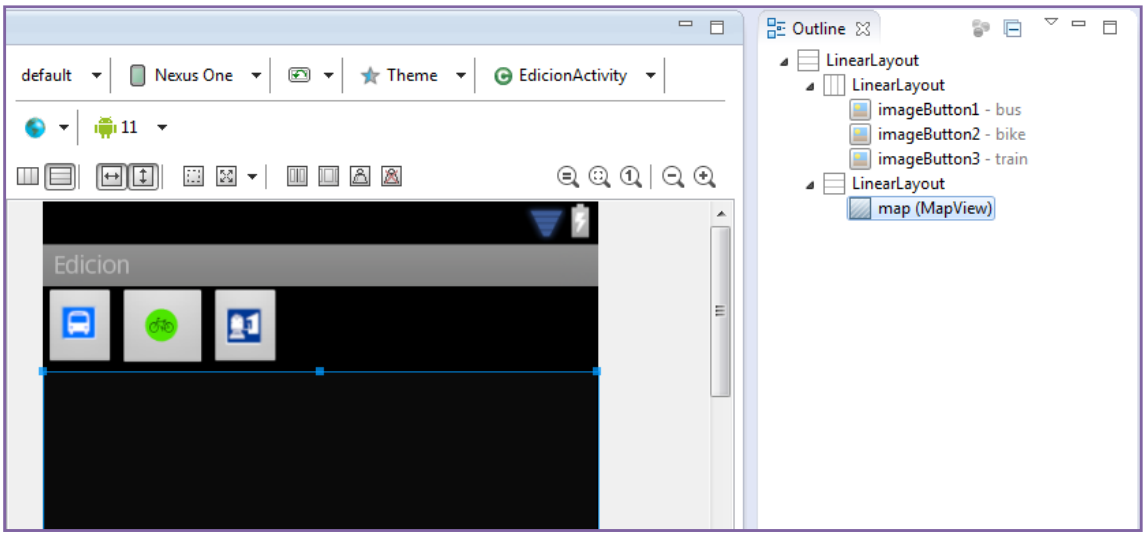


7.3.- Ejercicio: Edición

Paso 1

Cree un nuevo proyecto Android en Eclipse. Tal y como hizo en el ejercicio 5.5 sobre consultas, creará 2 *layouts* dentro del *layout* por defecto, el primero horizontal donde colocará 3 "ImageButton" y el segundo, vertical, donde colocará un "MapView".

En la documentación del curso, verá que hay tres imágenes. Cópielas y en Eclipse, péguelas en la carpeta "**res > drawable-ldpi**". En los "ImageButton" encontrará una propiedad "Src" que le permitirá seleccionar las imágenes que acaba de introducir. Colóquelas de forma que su *layout* quede del siguiente modo:



Paso 2

En su *Activity* añada al mapa un mapa base una *feature layer* como esta:

```
featureLayer = new
ArcGISFeatureLayer("http://trainingcloud.arcgis.com/ArcGIS/rest/services/Feat
ureService/Redlands_CommunityDesign/FeatureServer/0",
ArcGISFeatureLayer.MODE.ONDEMAND);

mMapView.addLayer(featureLayer);
```

La *feature layer* ha tenido que ser añadida a la clase como un miembro de la misma.

Paso 3

El objetivo del ejercicio es realizar una edición de una capa de puntos de estaciones de servicio en Redlands, en concreto, estaciones de autobús, de tren y parking para bicicletas. Esta capa, que ha cargado en el mapa, contiene un subtipo para cada uno de estos tipos de servicio.

La aplicación funcionará de tal modo que cuando el usuario haga clic sobre uno de los botones, se editará un elemento con el subtipo adecuado. Para gestionar estos subtipos cree las siguientes variables de clase:

```
Boolean bBus = false;
Boolean bTrain = false;
Boolean bBike = false;
static String BUS_STOP = "Bus stop";
static String BICYCLE_RACK = "Bicycle Rack";
static String TRAIN_STATION = "Train Station";
```

Las cadenas de texto creadas coinciden exactamente con los subtipos de la capa.

Paso 4

En este paso debe obtener una referencia a cada uno de los botones de su aplicación y establecer un *listener* de tal modo que se actualicen las variables de la clase de tal modo que sea cierta la última que ha sido pinchada:

```
ImageButton button_bus = (ImageButton) findViewById(R.id.imageButton1);
button_bus.setOnClickListener(new View.OnClickListener() {
	public void onClick(View v)
	{
		bBus = true;
		bTrain = false;
		bBike = false;
	}
});
```

Repita este paso para todos los botones.

Paso 5

A través de un *listener*, suscríbase al evento del mapa “OnSingleTap”.

Una vez dentro del método autogenerado por Eclipse, obtenga el punto marcado por el usuario y cree un elemento gráfico a partir de dicho punto. No necesita utilizar ninguna plantilla por el momento:

```
Point point = mMapView.toMapPoint(new Point(x, y));
Graphic g = featureLayer.createFeatureWithTemplate(null, point);
```

Cree una variable de tipo “String” y llámela “type”. Utilice las variables booleanas de la clase para comprobar cuál de ellas es verdadera y asigne a la variable el “static String” que le corresponde.

Paso 6

En esta aplicación va a asignar el atributo “Type” en base al tipo que ha seleccionado el usuario que quiere editar. Para ello debe primero obtener un mapeo de todos los atributos y luego establecer a cada uno de los atributos el valor que se desea. En este caso, siendo “g” el elemento gráfico creado y “type” la cadena de texto con el subtipo escogido por el usuario.

```
Map<String, Object> attributes = g.getAttributes();
attributes.put("Type", type);
```

Paso 7

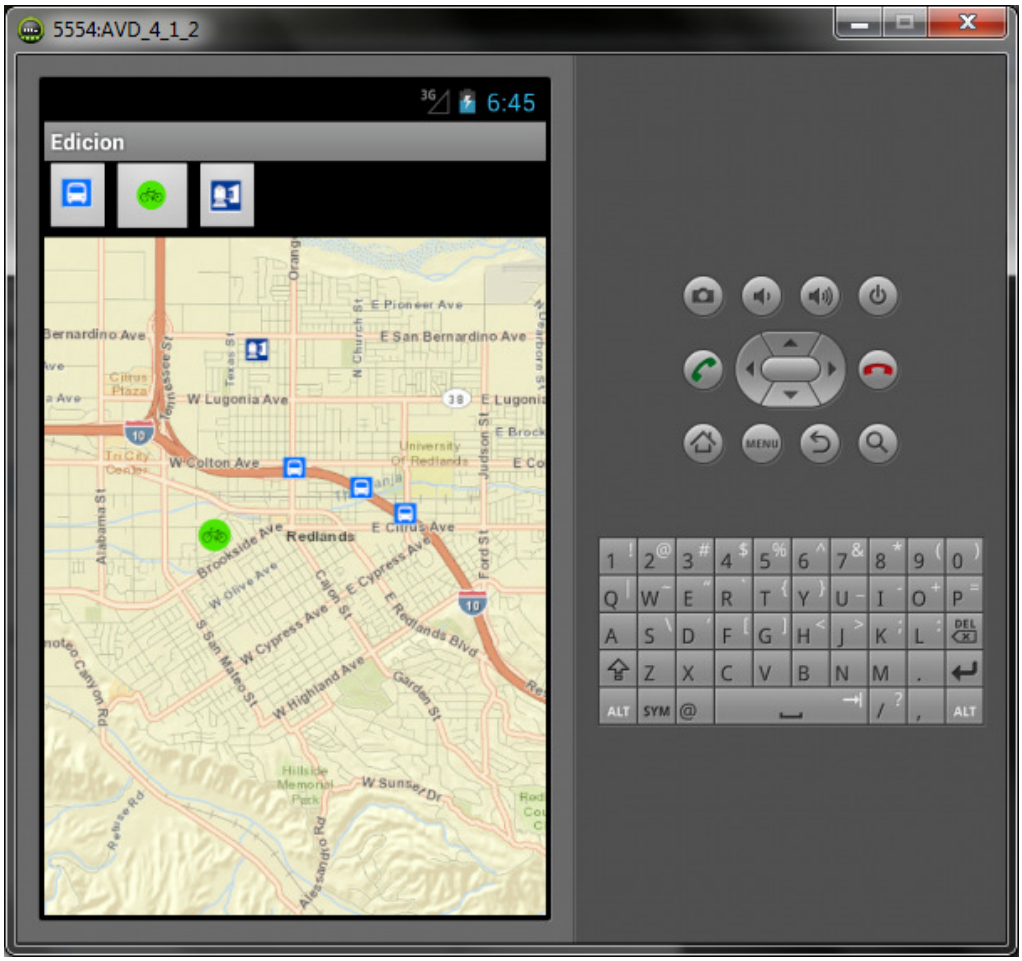
El último paso es muy importante debido a un concepto básico pero poco intuitivo de los elementos gráficos.

Anteriormente, ha creado un elemento gráfico “g”, ha obtenido todos sus atributos y ha modificado el valor de uno de ellos. Sin embargo, no es posible realizar un *commit (apply edits)* de los cambios debido a que un elemento gráfico es inmutable.

Por lo tanto, si quiere enviar un elemento gráfico con los cambios realizados, debe crear un elemento gráfico nuevo basado en el estado actual del primer elemento:

```
Graphic newGraphic = new Graphic(point, g.getSymbol(), attributes,
g.getInfoTemplate());
Graphic[] adds = new Graphic[] { newGraphic };

featureLayer.applyEdits(adds, null, null, null);
```

Y finalmente, debe enviar al servidor la edición realizada mediante el método “applyEdits()” de la *feature layer*. Ejecute y compruebe que es capaz de crear nuevos elementos de los tres subtipos disponibles:



Paso Opcional

Revise la documentación del método "applyEdits()" y comprobará que los tres primeros parámetros hacen referencia a tres tipos de ediciones. Consulte con su profesor si no entiende alguna de los mismos.

Además, tiene un cuarto parámetros que se trata de un *listener* que recoge la respuesta de ArcGIS Server al proceso de edición. Cree un *listener* de este tipo con su correspondiente método y si el servidor envía algún tipo de error, trate de avisar al usuario mediante un *toast* de Android. Puede encontrar información sobre los *toasts* aquí:

http://developer.android.com/intl/es/guide/topics/ui/notifiers/toasts.html

Paso Opcional

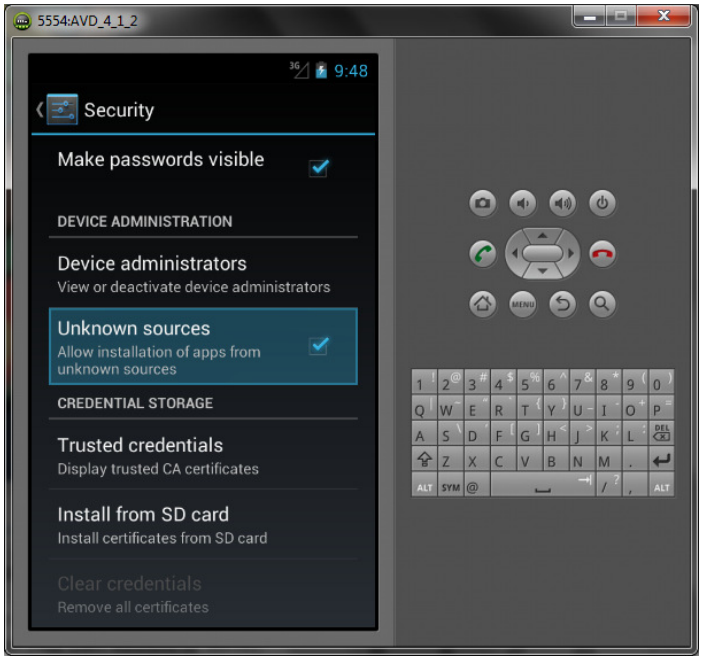
Es posible mediante programación acceder a los diferentes subtipos de la capa. Trate de revisar los ejemplos de ArcGIS para Android para localizar cómo hacerlo.

8.- Despliegue de aplicaciones

8.1. – Distribución

La distribución de aplicaciones comerciales suele producirse mediante el uso de mercados de aplicaciones o *App Marketplace*. El más extendido entre ellos, su aplicación está disponible en todas las versiones de Android, es Google Play. Este método es el más extendido entre los usuarios finales, por lo que si su aplicación está enfocada a un público genérico, es la opción más recomendada.

Es posible que quiera dejar disponible su aplicación, empaquetada en forma de fichero APK, en otro mercado alternativo o directamente en un sitio web, incluso un servidor empresarial o su intranet. Debe saber que en estos casos, el dispositivo donde será instalada su aplicación debe tener habilitada la opción de correspondiente. Puede encontrarla en "**Settings > Security**":



Como última opción, puede enviar mediante correo electrónico el paquete APK y su dispositivo móvil lo reconocerá como tal y comenzará la instalación del mismo aunque también necesita la opción de seguridad comentada anteriormente.

8.2. – Modelos de negocio

Desarrollar una aplicación Android puede estar motivado por dos causas: bajo demanda o por iniciativa propia. Si el proyecto es bajo demanda, el modelo de negocio ya lo tiene fijado. Sin embargo, si desarrolla aplicaciones Android por iniciativa propia conviene repasar los modelos de negocio más habituales aunque hay otros.

Venta directa de la aplicación

Es el ejemplo histórico más clásico. La ventaja de vender una aplicación de forma directa es que obtendrá ingresos por cada usuario, sin embargo, necesitará promocionar sus aplicaciones y el alcance de su producto será menor.

Versión lite vs. Versión premium

Este modelo obliga a desarrollar dos versiones de la aplicación, la primera de ellas tendrá una cantidad de contenidos o funcionalidad menor y será gratuita, mientras que la segunda será un desarrollo completo y dejará de ser gratuita.

Si la aplicación es realmente útil, es posible obtener beneficios con la versión Premium, sirviendo la aplicación Lite de demostración.

DLCs

Un *DLC* (*downloadable content*) es un contenido descargable para un determinado software. El origen de este modelo de negocio reside en los videojuegos y surgió para ampliar los contenidos de algunos de ellos haciendo pagar a los usuarios por ellos. Es aplicable a otros tipos de software pero la mayor extensión es en el mundo de los videojuegos.

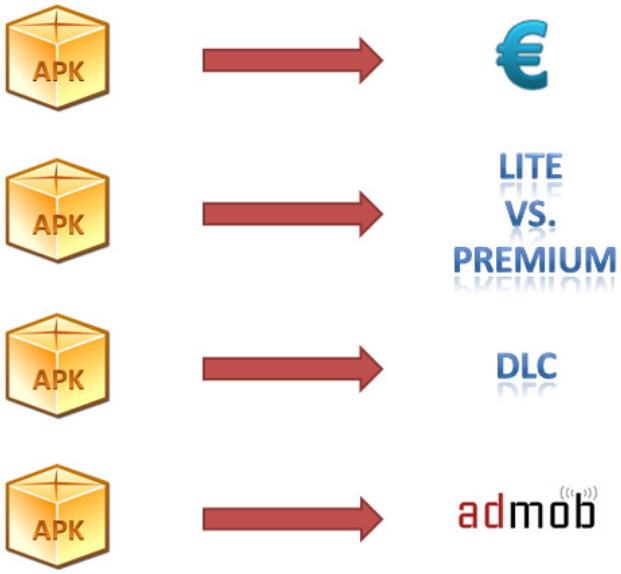
La creación de *DLCs* es indistinta del modelo de negocio utilizado para la versión original de la aplicación, es decir, indistintamente de si el usuario ha pagado por la aplicación o no, es posible la creación de *DLCs* para la misma.

Publicidad: Admob

Un modelo muy extendido hoy en día es el de la publicidad en aplicaciones gratuitas. Debido a la tremenda competencia en el desarrollo de aplicaciones existente, gran número de

desarrolladores se ven obligados a crear aplicaciones gratuitas, forzando su modelo de negocio a obtener ingresos por publicidad.

Hay muchos sistemas de publicidad pero tal vez el más extendido es el proporcionado por Google: *admob*.

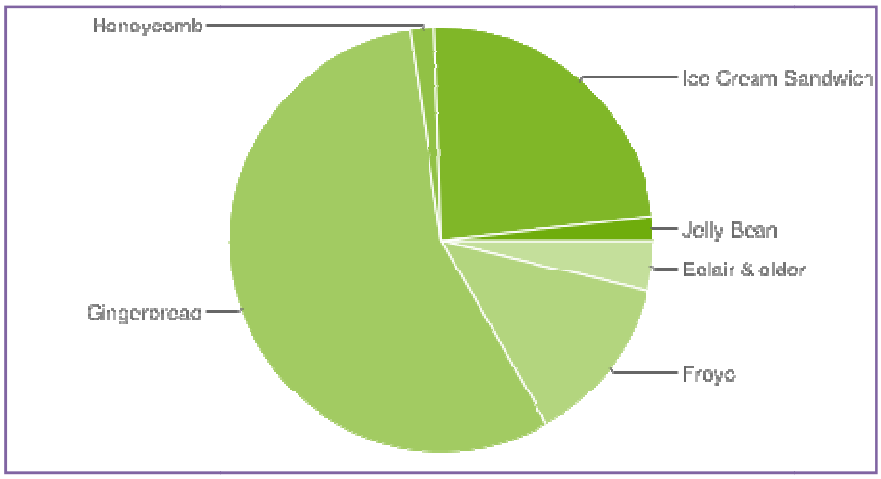


Anexo 1: Acerca de las versiones de Android

Estas son las diferentes versiones de Android en la actualidad:

Versión	Nombre	API	Distribución
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.4%
2.1	Eclair	7	3.4%
2.2	Froyo	8	12.9%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	55.5%
3.1	Honeycomb	12	0.4%
3.2		13	1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	23.7%
4.1	Jelly Bean	16	1.8%

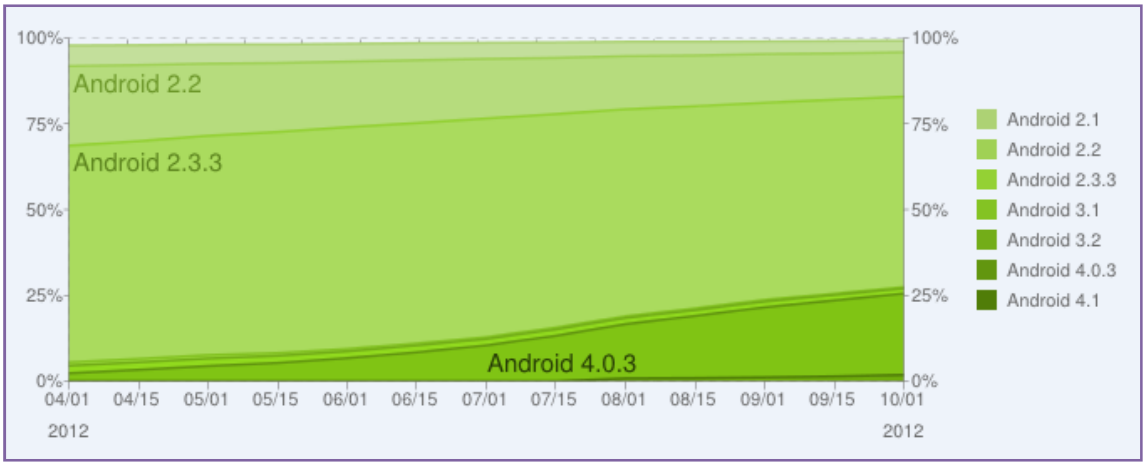
Si en el momento de desarrollar una aplicación no se conoce la versión target de Android o el desarrollo se hace para un público abierto, es interesante conocer las propias estadísticas de Google al respecto (http://developer.android.com/about/dashboards/index.html):



Estos datos fueron publicados a finales de Octubre de 2012.

Cabe tener en cuenta que en comparación con datos anteriores, Android 2.2 Froyo está en claro declive, Android 2.3 Gingerbread en moderado descenso y Android 4.0 Ice Cream Sandwich en ascenso.

La siguiente tabla representa una evolución histórica según los datos de Google:


100%
75%
50%
25%
0%
Android 2.2
Android 2.3.3
Android 4.0.3
04/01
2012
04/15
05/01
05/15
06/01
06/15
07/01
07/15
08/01
08/15
09/01
09/15
10/01
2012
Android 2.1
Android 2.2
Android 2.3.3
Android 3.1
Android 3.2
Android 4.0.3
Android 4.1

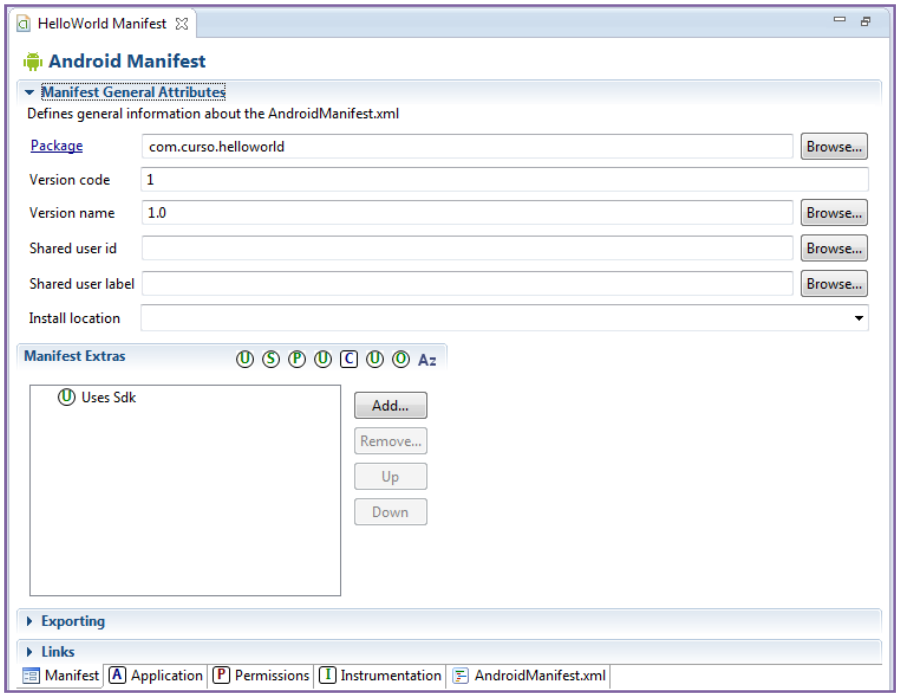
Anexo 2: AndroidManifest.xml

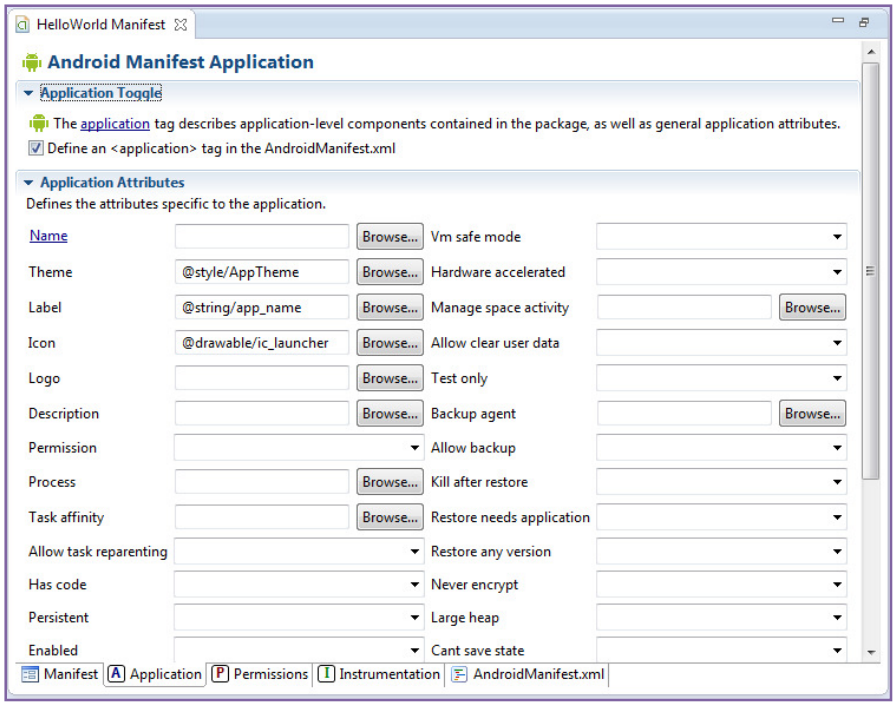
Además de carpetas, también se incluyen en el proyecto algunos archivos genéricos que, sin ser parte del código fuente o recursos, determinan la forma en que se generará la aplicación al compilar. De todos ellos, el archivo **AndroidManifest.xml** es el más importante de todos.

Este archivo, el *manifiesto* de la aplicación, contiene la información que el sistema operativo necesita conocer antes de poder ejecutar la aplicación. Aunque nos referiremos a este archivo en muchas ocasiones en futuros posts, valga por el momento la siguiente lista como resumen de su contenido:

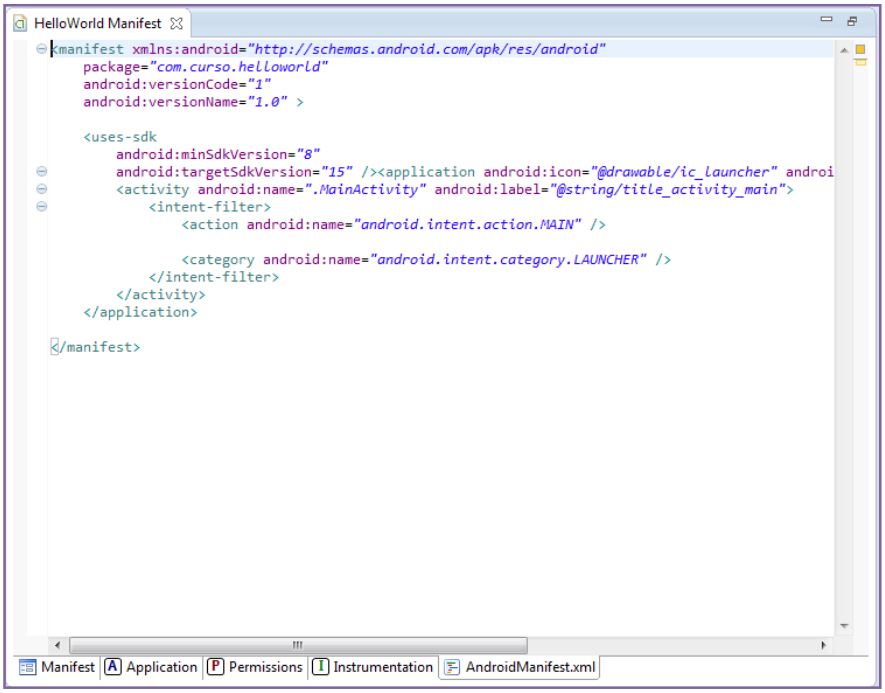
- **Identificación de la aplicación**: nombre del paquete, número de versión, etc. También si permitimos que la aplicación se pueda mover a la tarjeta SD.
- **Atributos de la aplicación**: el nombre que se le mostrará al usuario, icono y logotipo, una descripción de su funcionalidad e incluso permisos globales.
- **Permisos**: los que requiere la aplicación para funcionar. Son los que se le muestran al usuario antes de instalarla.
- **Componentes hardware**: los que la aplicación utiliza o puede utilizar. Se puede indicar si son indispensables o no y sirven para que la aplicación sólo aparezca en el Android Market de equipos compatibles.
- **Pantallas**: las que son compatibles con la aplicación. También se utiliza para filtrar aplicaciones en el Android Market.
- **Versiones de Android**: la mínima que la aplicación requiere para ejecutarse y aquella para la que se ha creado (que no tienen por qué coincidir). Principal elemento de filtrado en el Android Market.
- **Actividades**: datos de todas las actividades que incluye la aplicación, como nombre, icono, permisos de ejecución, etc.
- **Otros componentes de la aplicación**: datos de servicios, proveedores y receptores de contenidos, etc.

El ADT provee de un editor basado en formularios que facilitan la edición del XML:





A través de las pestañas localizadas en la parte inferior del editor, es posible modificar el manifiesto para adecuarlo a los requisitos de cada proyecto. En la última de ellas, es posible acceder al texto XML:



Anexo 3: Resultado de los ejercicios

Consultas

```
package com.cursoAndroid.consultas;

import android.app.Activity;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import com.esri.android.map.GraphicsLayer;
import com.esri.android.map.MapView;
import com.esri.android.map.ags.ArcGISTiledMapServiceLayer;
import com.esri.core.geometry.SpatialReference;
import com.esri.core.map.FeatureSet;
import com.esri.core.map.Graphic;
import com.esri.core.renderer.SimpleRenderer;
import com.esri.core.symbol.SimpleMarkerSymbol;
import com.esri.core.tasks.ags.query.Query;
import com.esri.core.tasks.ags.query.QueryTask;

public class ConsultasActivity extends Activity
{
	MapView mMapView ;
	GraphicsLayer gl;
	String targetServerURL =
"http://trainingcloud.arcgis.com/ArcGIS/rest/services/Redlands_PointsOfInterest/MapServer/0";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Obtenemos el MapView
        mMapView = (MapView) findViewById(R.id.map);

        // Añadimos un mapa base al mapa
        ArcGISTiledMapServiceLayer tiled = new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/MapServer");
             mMapView.addLayer(tiled);

             // Creamos una capa grafica
             gl = new GraphicsLayer();
             // Para la capa grafica necesitamos un renderer.
             // En este caso necesitamos un renderer de puntos ya que en esta capa añadiremos puntos de interes de redlands puntuales
             SimpleRenderer sr = new SimpleRenderer(new
SimpleMarkerSymbol(Color.RED, 10, SimpleMarkerSymbol.STYLE.CIRCLE));
             gl.setRenderer(sr);
```

```
            // La añadimos al mapa
            mMapView.addLayer(gl);

            // Obtenemos el botón de bancos
            Button banksButton = (Button) findViewById(R.id.button1);
            // Establecemos un listener para capturar cuando el usuario haga
clic
            banksButton.setOnClickListener(new View.OnClickListener() {
                  public void onClick(View v) {
                        // Limpiamos los elementos que haya actualmente en
la capa grafica
                        gl.removeAll();
                        // Establecemos como parametros la URL de la capa y
la clausula where que le queremos aplicar
                        String[] queryParams = { targetServerURL,
"TYPE='Bank'" };
                        // Creamos una nueva tarea asincrona y la lanzamos
con los parametros
                        AsyncQueryTask ayncQuery = new AsyncQueryTask();
                        ayncQuery.execute(queryParams);
                  }
            });

            // Obtenemos el botón de zonas recreativas
            Button recreativesButton = (Button) findViewById(R.id.button2);
            // Establecemos un listener para capturar cuando el usuario haga
clic
            recreativesButton.setOnClickListener(new View.OnClickListener()
{
                  public void onClick(View v) {

                        // Limpiamos los elementos que haya actualmente en
la capa grafica
                        gl.removeAll();
                        // Establecemos como parametros la URL de la capa y
la clausula where que le queremos aplicar
                        String[] queryParams = { targetServerURL,
"TYPE='Recreation'" };
                        // Creamos una nueva tarea asincrona y la lanzamos
con los parametros
                        AsyncQueryTask ayncQuery = new AsyncQueryTask();
                        ayncQuery.execute(queryParams);
                  }
            });
    }

    // Creamos una clase que ejecute tareas asincronas extendiendo la clase
AsyncTask
    private class AsyncQueryTask extends AsyncTask<String, Void, FeatureSet>
    {
      // Este metodo es obligatorio de implementar y realmente representa la
ejecucion asincrona que se quiere realizar
      protected FeatureSet doInBackground(String... queryParams)
            {
            // Comprobamos que la llamada es adecuada y tiene al menos los
parametros URL y clausula where
                  if (queryParams == null || queryParams.length <= 1)
                        return null;
```

```
                    // Los recogemos en strings
                    String url = queryParams[0];
                    String whereClause = queryParams[1];

                    // Creamos un objeto Query
                    Query query = new Query();
                    // Le damos los parametros necesarios aplicando la
clausula where
                    //           y solicitando que el servidor devuelva
geometrias en el sistema de referencia que deseamos
                    SpatialReference sr = SpatialReference.create(102100);
                    query.setOutSpatialReference(sr);
                    query.setReturnGeometry(true);
                    query.setWhere(whereClause);

                    // Creamos el objeto QueryTask utilizando la URL
proporcionada
                    QueryTask qTask = new QueryTask(url);
                    FeatureSet fs = null;
                    try
                    {
                          // Tratamos de ejecutar la tarea recogiendo el
FeatureSet con los elementos que concuerden con la consulta realizada
                          fs = qTask.execute(query);
                    }
                    catch (Exception e)
                    {
                          // Si hay algun error los enviamos a la consola
                          System.out.println(e.getMessage());
                    }

                    return fs;
              }

      // Uno de los metodos que mas habitualmente se implementan es
onPostExecute, que recibe como parametro el resultado de la ejecucion
              protected void onPostExecute(FeatureSet result)
              {
                    if (result != null)
                    {
                          // Si el FeatureSet devuelto por el servidor no es
nulo comprobamos si contiene elementos y los añadimos a la capa grafica
                          Graphic[] grs = result.getGraphics();
                          if (grs.length > 0)
                          {
                                gl.addGraphics(grs);

                          }
                    }
              }
      }


      @Override
      protected void onDestroy() {
            super.onDestroy();
 }
      @Override
      protected void onPause() {
```

```
            super.onPause();
            mMapView.pause();
 }
      @Override
      protected void onResume() {
            super.onResume();
            mMapView.unpause();
      }

}
```

Renderer y callout

```
package com.cursoAndroid.featureLayerRenderer;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;

import com.esri.android.map.Callout;
import com.esri.android.map.MapView;
import com.esri.android.map.ags.ArcGISFeatureLayer;
import com.esri.android.map.ags.ArcGISTiledMapServiceLayer;
import com.esri.android.map.event.OnSingleTapListener;
import com.esri.core.geometry.Point;
import com.esri.core.map.Graphic;
import com.esri.core.renderer.UniqueValue;
import com.esri.core.renderer.UniqueValueRenderer;
import com.esri.core.symbol.SimpleMarkerSymbol;

public class FeatureLayerRendererActivity extends Activity
{
       MapView mMapView;
       Callout callout;
       ArcGISFeatureLayer featureLayer;

    @Override
       public void onCreate(Bundle savedInstanceState)
       {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.main);

              // Añadimos un mapa base al mapa
              mMapView = (MapView) findViewById(R.id.map);
              mMapView.addLayer(new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/serv
ices/World_Street_Map/MapServer"));

              // Creamos una featurelayer
              featureLayer = new
ArcGISFeatureLayer("http://trainingcloud.arcgis.com/ArcGIS/rest/services/Redl
ands_PointsOfInterest/MapServer/0", ArcGISFeatureLayer.MODE.SNAPSHOT);

              // Creamos el uniquevaluerenderer y los uniquevalues necesarios
              UniqueValueRenderer renderer = new UniqueValueRenderer();
              renderer.setField1("TYPE");
              renderer.setDefaultSymbol(new SimpleMarkerSymbol(Color.BLACK,
10, SimpleMarkerSymbol.STYLE.CIRCLE));

              UniqueValue unique1 = new UniqueValue();
              unique1.setLabel("Bank");
              unique1.setSymbol(new SimpleMarkerSymbol(Color.YELLOW, 10,
SimpleMarkerSymbol.STYLE.DIAMOND));
              unique1.setValue(new Object[]{"Bank"});
              renderer.addUniqueValue(unique1);
```

```
            UniqueValue unique2 = new UniqueValue();
            unique2.setLabel("Recreative");
            unique2.setSymbol(new SimpleMarkerSymbol(Color.GREEN, 10,
SimpleMarkerSymbol.STYLE.CIRCLE));
            unique2.setValue(new Object[]{"Recreative"});
            renderer.addUniqueValue(unique2);

            UniqueValue unique3 = new UniqueValue();
            unique3.setLabel("Restaurant");
            unique3.setSymbol(new SimpleMarkerSymbol(Color.RED, 10,
SimpleMarkerSymbol.STYLE.CROSS));
            unique3.setValue(new Object[]{"Restaurant"});
            renderer.addUniqueValue(unique3);

            UniqueValue unique4 = new UniqueValue();
            unique4.setLabel("Apartment");
            unique4.setSymbol(new SimpleMarkerSymbol(Color.BLUE, 10,
SimpleMarkerSymbol.STYLE.SQUARE));
            unique4.setValue(new Object[]{"Apartment"});
            renderer.addUniqueValue(unique4);

            UniqueValue unique5 = new UniqueValue();
            unique5.setLabel("Arts");
            unique5.setSymbol(new SimpleMarkerSymbol(Color.DKGRAY, 10,
SimpleMarkerSymbol.STYLE.X));
            unique5.setValue(new Object[]{"Arts"});
            renderer.addUniqueValue(unique5);

            // Aplicamos el renderer y la añadimos al mapa
            featureLayer.setRenderer(renderer);
            mMapView.addLayer(featureLayer);

            // Suscribimos el vento singletap
            mMapView.setOnSingleTapListener(new OnSingleTapListener()
            {
                  public void onSingleTap(float x, float y)
                  {
                        if (!mMapView.isLoaded())return;

                        // Recogemos los elementos graficos mas cercanos al
punto pulsado
                        int[] uids = featureLayer.getGraphicIDs(x, y, 2);
                        if (uids != null && uids.length > 0)
                        {
                              // El elementos a mostrar es el primero del
array, el mas proximo
                              Graphic gr =
featureLayer.getGraphic(uids[0]);

                              // Recogemos el callout del mapview y le
aplicamos el estilo1
                              callout = mMapView.getCallout();
                              callout.setStyle(R.xml.mycallout);

                              // Obtenemos los valores de los atributos
name y type
                              String poi_name = (String)
gr.getAttributeValue("NAME");
```

```
                                    String poi_type =
gr.getAttributeValue("TYPE").toString();

                                    // Creamos una vista para el layout
utilizando el inflater
                                    View view =
LayoutInflater.from(FeatureLayerRendererActivity.this).inflate(R.layout.mylin
earlayout, null);

                                    // Obtenemos los textviews de la vista creada
y modificamos los textos
                                    TextView name = (TextView)
view.findViewById(R.id.textView1);
                                    name.setText(poi_name);
                                    TextView number = (TextView)
view.findViewById(R.id.textView2);
                                    number.setText(poi_type);

                                    // Finalmente se aplica el contenido y se
muestra el callout en el mapa
                                    callout.setContent(loadView(poi_name,
poi_type));
                                    callout.show(mMapView.toMapPoint(new Point(x,
y)));
                            }
                            else
                            {
                                    // Si el callout esta visible y el usuario
hace clic, se oculta
                                    if (callout != null && callout.isShowing())
                                    {
                                          callout.hide();
                                    }
                            }
                      }
                });
        }


        @Override
        protected void onDestroy() {
                super.onDestroy();
 }
        @Override
        protected void onPause() {
                super.onPause();
                mMapView.pause();
 }
        @Override    protected void onResume() {
                super.onResume();
                mMapView.unpause();
        }

}
```

Localización

```
package com.cursoAndroid.localizador;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.widget.TextView;

import com.esri.android.map.GraphicsLayer;
import com.esri.android.map.MapView;
import com.esri.android.map.ags.ArcGISTiledMapServiceLayer;
import com.esri.android.map.event.OnSingleTapListener;
import com.esri.core.geometry.Point;
import com.esri.core.geometry.SpatialReference;
import com.esri.core.renderer.SimpleRenderer;
import com.esri.core.symbol.SimpleMarkerSymbol;
import com.esri.core.tasks.ags.geocode.Locator;
import com.esri.core.tasks.ags.geocode.LocatorReverseGeocodeResult;


public class LocalizadorActivity extends Activity {

      MapView mMapView;
      GraphicsLayer gl;
      Locator locator;


    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);


        // Añadimos un mapa base al mapa
        mMapView = (MapView)findViewById(R.id.map);
        ArcGISTiledMapServiceLayer tiled = new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/serv
ices/World_Street_Map/MapServer");
            mMapView.addLayer(tiled);

            // Creamos una capa grafica
            gl = new GraphicsLayer();
            // Para la capa grafica necesitamos un renderer.
            // En este caso necesitamos un renderer de puntos ya que en esta
capa añadiremos puntos de interes de redlands puntuales
            SimpleRenderer sr = new SimpleRenderer(new
SimpleMarkerSymbol(Color.RED, 10, SimpleMarkerSymbol.STYLE.CIRCLE));
            gl.setRenderer(sr);
            // La añadimos al mapa
            mMapView.addLayer(gl);

            // Obtenemos el evento singleTap
            mMapView.setOnSingleTapListener(new OnSingleTapListener()
            {
                  public void onSingleTap(final float x, final float y)
                  {
```

```
                        // Limpiamos la capa grafica de resultados
anteriores
                        gl.removeAll();

                        // Obtenemos el punto tocado por el usuario
                        final Point loc = mMapView.toMapPoint(x, y);

                        // Instanciamos el localizador
                        locator = new
Locator("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Locators/
ESRI_Geocode_USA/GeocodeServer");

                        try
                        {
                              // Necesitaos el sistema de referencia del
mapa para el localizador
                              SpatialReference sr =
mMapView.getSpatialReference();

                              // Obtenemos el resultado de la consulta
                              LocatorReverseGeocodeResult result =
locator.reverseGeocode(loc,     1000.00, sr, sr);

                              // El estado tiene 2 caracteres y el zip 5 o
no esta bien la direccion
                              if
(result.getAddressFields().get("State").length() != 2
                                          &&
result.getAddressFields().get("Zip").length() != 5)
                              {
                                    mMapView.getCallout().show(loc,
createTextView("No se ha localizado una direccion."));
                              }
                              else
                              {
                                    // Si todo va bien, creamos un string
con los cuatro campos
                                    String msg = "Address:"   +
result.getAddressFields().get("Address") + "\n"
                                                + "City:" +
result.getAddressFields().get("City") + "\n"
                                                + "State:"   +
result.getAddressFields().get("State") + "\n"
                                                + "Zip:" +
result.getAddressFields().get("Zip");

                                    mMapView.getCallout().show(loc,
createTextView(msg));
                              }
                        }
                        catch (Exception e)
                        {
                              e.printStackTrace();
                              mMapView.getCallout().show(loc,
createTextView("No se ha localizado una direccion."));
                        }
                  }
            });
    }
```

```
    // Utilizamos este metodo para crear un textview a partir de un texto
dado
       private TextView createTextView(String text)
       {
              final TextView msg = new TextView(this);
              msg.setText(text);
              msg.setTextSize(12);
              msg.setTextColor(Color.BLACK);
              return msg;
       }


       @Override
       protected void onDestroy() {
              super.onDestroy();
 }
       @Override
       protected void onPause() {
              super.onPause();
              mMapView.pause();
 }
       @Override    protected void onResume() {
              super.onResume();
              mMapView.unpause();
       }

}
```

Geoprocesamiento

```
package com.cursoAndroid.Geoprocesamiento;

import java.util.ArrayList;

import android.app.Activity;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;

import com.esri.android.map.GraphicsLayer;
import com.esri.android.map.MapView;
import com.esri.android.map.ags.ArcGISTiledMapServiceLayer;
import com.esri.android.map.event.OnSingleTapListener;
import com.esri.core.geometry.Geometry;
import com.esri.core.geometry.Point;
import com.esri.core.map.Graphic;
import com.esri.core.renderer.SimpleRenderer;
import com.esri.core.symbol.SimpleFillSymbol;
import com.esri.core.symbol.SimpleMarkerSymbol;
import com.esri.core.symbol.SimpleMarkerSymbol.STYLE;
import com.esri.core.tasks.ags.geoprocessing.GPFeatureRecordSetLayer;
import com.esri.core.tasks.ags.geoprocessing.GPLinearUnit;
import com.esri.core.tasks.ags.geoprocessing.GPParameter;
import com.esri.core.tasks.ags.geoprocessing.GPResultResource;
import com.esri.core.tasks.ags.geoprocessing.Geoprocessor;


public class GeoprocesamientoActivity extends Activity {

      MapView mMapView;
      ArrayList<GPParameter> params;
      Geoprocessor gp;
      GraphicsLayer gLayer;
      Point mappoint;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Añadimos un mapa base al mapa
        mMapView = (MapView)findViewById(R.id.map);
        ArcGISTiledMapServiceLayer tiled = new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/serv
ices/World_Street_Map/MapServer");
             mMapView.addLayer(tiled);

             // Creamos una capa grafica
             gLayer = new GraphicsLayer();
             // La añadimos al mapa
             mMapView.addLayer(gLayer);

             // Obtenemos el evento singleTap
             mMapView.setOnSingleTapListener(new OnSingleTapListener()
```

```
            {
                  public void onSingleTap(final float x, final float y)
                  {
                        // Limpiamos la capa grafica de resultados
anteriores
                        gLayer.removeAll();

                        // Obtenemos el punto y creamos un elemento grafico
con el
                        mappoint = mMapView.toMapPoint(x, y);
                        Graphic g = new Graphic(mappoint, new
SimpleMarkerSymbol(Color.RED, 10, STYLE.CIRCLE));
                        gLayer.addGraphic(g);

                        // Primer parametro para el GP, un
GPFeatureRecordSetLayer
                        GPFeatureRecordSetLayer gpf = new
GPFeatureRecordSetLayer("Input_Observation_Point");

      gpf.setSpatialReference(mMapView.getSpatialReference());
                        gpf.setGeometryType(Geometry.Type.POINT);
                        // Añadimos el punto del usuario al
featurerecordset
                        Graphic f = new Graphic(mappoint, new
SimpleMarkerSymbol(Color.RED,25,STYLE.DIAMOND));
                        gpf.addGraphic(f);

                        // Segundo parametro, un GPLinearUnit
                        GPLinearUnit gpl = new
GPLinearUnit("Viewshed_Distance");
                        gpl.setUnits("esriMeters");
                        gpl.setDistance(1000);

                        // Creamos una lista de parametros
                        params = new ArrayList<GPParameter>();
                        params.add(gpf);
                        params.add(gpl);

                        // Ejecutamos el GP con la clase AsyncTask
                        new ViewShedQuery().execute(params);
                  }
            });
    }

    class ViewShedQuery extends AsyncTask<ArrayList<GPParameter>, Void,
GPParameter[]>
    {
            GPParameter[] outParams = null;

            @Override
            protected GPParameter[] doInBackground(ArrayList<GPParameter>...
params1)
            {
                  // Instanciamos un nuevo geoprocessor y le damos el
sistema de referencia
                  gp = new
Geoprocessor("http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/Elev
ation/ESRI_Elevation_World/GPServer/Viewshed");
                  gp.setOutSR(mMapView.getSpatialReference());
```

```
                    try
                    {
                          // Ejecutamos el GP y obtenemos una lista de
resultados, nos quedamos el primero
                          GPResultResource rr = gp.execute(params1[0]);
                          outParams = rr.getOutputParameters();
                    }
                    catch (Exception e)
                    {
                          e.printStackTrace();
                    }
                    return outParams;
              }

              @Override
              protected void onPostExecute(GPParameter[] result)
              {
                    if (result == null)
                          return;

                    // Una vez obtenida la capa de salida, obtenemos todos los
elementos graficos y los añadimos a la capa
                    GPFeatureRecordSetLayer fsl = (GPFeatureRecordSetLayer)
result[0];
                    for (Graphic feature : fsl.getGraphics())
                    {
                          Graphic g = new Graphic(feature.getGeometry(),new
SimpleFillSymbol(Color.CYAN));
                          gLayer.addGraphic(g);
                    }
              }
    }


       @Override
       protected void onDestroy() {
             super.onDestroy();
 }
       @Override
       protected void onPause() {
             super.onPause();
             mMapView.pause();
 }
       @Override    protected void onResume() {
             super.onResume();
             mMapView.unpause();
       }

}
```

Edición

```
package com.cursoAndroid.edicion;

import java.util.Map;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;

import com.esri.android.map.MapView;
import com.esri.android.map.ags.ArcGISFeatureLayer;
import com.esri.android.map.ags.ArcGISTiledMapServiceLayer;
import com.esri.android.map.event.OnSingleTapListener;
import com.esri.core.geometry.Point;
import com.esri.core.map.Graphic;

public class EdicionActivity extends Activity {

	MapView mMapView;
	ArcGISFeatureLayer featureLayer;

	Boolean bBus = false;
	Boolean bTrain = false;
	Boolean bBike = false;
	static String BUS_STOP = "Bus stop";
	static String BICYCLE_RACK = "Bicycle Rack";
	static String TRAIN_STATION = "Train Station";

	/** Called when the activity is first created. */
	@Override
	public void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);

		// Añadimos un mapa base al mapa
		mMapView = (MapView) findViewById(R.id.map);
		mMapView.addLayer(new
ArcGISTiledMapServiceLayer("http://services.arcgisonline.com/ArcGIS/rest/serv
ices/World_Street_Map/MapServer"));

		// Creamos una featurelayer
		featureLayer = new
ArcGISFeatureLayer("http://trainingcloud.arcgis.com/ArcGIS/rest/services/Feat
ureService/Redlands_CommunityDesign/FeatureServer/0",
ArcGISFeatureLayer.MODE.ONDEMAND);
		mMapView.addLayer(featureLayer);

		ImageButton button_bus = (ImageButton)
findViewById(R.id.imageButton1);
		button_bus.setOnClickListener(new View.OnClickListener() {
			public void onClick(View v)
			{
				bBus = true;
				bTrain = false;
				bBike = false;
```

```
                }
            });

            ImageButton button_bike = (ImageButton)
findViewById(R.id.imageButton2);
            button_bike.setOnClickListener(new View.OnClickListener() {
                  public void onClick(View v)
                  {
                        bBus = false;
                        bTrain = false;
                        bBike = true;
                  }
            });

            ImageButton button_train = (ImageButton)
findViewById(R.id.imageButton3);
            button_train.setOnClickListener(new View.OnClickListener() {
                  public void onClick(View v)
                  {
                        bBus = false;
                        bTrain = true;
                        bBike = false;
                  }
            });

            mMapView.setOnSingleTapListener(new OnSingleTapListener() {
                  public void onSingleTap(float x, float y)
                  {
                        Point point = mMapView.toMapPoint(new Point(x, y));
                        Graphic g =
featureLayer.createFeatureWithTemplate(null, point);

                        String type;
                        if(bBus)
                              type = BUS_STOP;
                        else if(bBike)
                              type = BICYCLE_RACK;
                        else if(bTrain)
                              type = TRAIN_STATION;
                        else
                              return;

                        Map<String, Object> attributes = g.getAttributes();
                        attributes.put("Type", type);

                        // Los elementos graficos son inmutables por eso no
es posible
                        // modificar sus attributos una vez creados asi que
creamos uno
                        // nuevo basado en el primero
                        Graphic newGraphic = new Graphic(point,
g.getSymbol(), attributes, g.getInfoTemplate());
                        Graphic[] adds = new Graphic[] { newGraphic };

                        featureLayer.applyEdits(adds, null, null, null);
                  }
            });
      }
```

```
	@Override
	protected void onDestroy() {
		super.onDestroy();
	}

	@Override
	protected void onPause() {
		super.onPause();
		mMapView.pause();
	}

	@Override
	protected void onResume() {
		super.onResume();
		mMapView.unpause();
	}
}
```

Antonio Remírez Remírez