UNIDAD 2: PROGRAMACIÓN DE APLICACIONES PARA DISPOSITIVOS MÓVILES

Módulo Profesional:
Programación Multimedia y Dispositivos Móviles

Índice

RESUMEN INTRODUCTORIO

En este tema, se pretende mostrar al alumno los conceptos básicos y necesarios para el desarrollo de aplicaciones móviles, centrándose en varios aspectos importantes como interfaces de usuario, componentes gráficos, eventos y sus métodos correspondientes, comunicación entre dispositivos, almacenamiento de información y conexión inalámbrica. Al finalizar este tema, el usuario estará capacitado para realizar la implementación de aplicaciones sencillas en Android Studio y, gracias a su desarrollo, dispondrá de los conocimientos y destrezas necesarias para seguir mejorando y poder participar en un mercado cada vez más competente.

Además, todo lo aprendido a través de este tema servirá para reforzar en el alumnado aún más sus conocimientos en el lenguaje de programación Java, adquiriendo una mayor experiencia en la implementación de software orientado a objetos. De forma intrínseca, también reforzará los conocimientos y el desarrollo de aplicaciones a través del lenguaje XML, utilizado para la parte visual de los programas en los sistemas Android.

INTRODUCCIÓN

Los dispositivos móviles están conviviendo con todos nosotros desde hace ya varios años. Hoy es imposible pensar en vivir sin la presencia de un móvil en nuestro entorno hasta el punto de crear en la sociedad actual una necesidad imperiosa de estar constantemente conectado. Ante esta situación, conocida como la era de la comunicación, se hace inevitable que los profesionales del sector tecnológico e informático conozcan los recursos y las técnicas que permiten hacer frente a las necesidades para este tipo de dispositivos.

En este sentido, el sistema operativo Android ha tomado una posición privilegiada y ya forma parte de los móviles y las tablets de un elevadísimo número de usuarios. Android ofrece multitud de aplicaciones de diversa índole y, como tal, la competencia en el mercado del desarrollo de software se hace cada vez mayor, no solo a nivel empresarial sino a nivel de usuario con conocimientos suficientes para la implementación de aplicaciones.

1. HERRAMIENTAS Y FASES DE CONSTRUCCIÓN

El avance de las tecnologías de la información y de la comunicación en la actualidad ha hecho posible que, dentro del mundo de los *smartphones*, exista un gran número de herramientas dedicadas a la programación de aplicaciones para dispositivos móviles. Estas herramientas son el presente y el futuro, puesto que hoy en día el uso de aplicaciones móviles son una realidad en constante crecimiento.

Se podría listar un elevado número de herramientas o de *IDEs* (software para la programación de aplicaciones, también conocido como Entornos de Desarrollo; en inglés, *Integrated Development Environment*), pero de todas ellas se consideran dos: Xamarin y Android Studio, descritas más adelante.

Las fases de construcción de una aplicación se resumen en las 4 siguientes:

1. Definición y requisitos: En esta primera fase se trata de definir los conceptos, la funcionalidad y usabilidad de la aplicación, obteniendo como resultado el concepto y las expectativas de diseño. Para llevar a cabo esta fase basta con utilizar un papel y un bolígrafo.

2. Experiencia del usuario y aplicación de diseño del flujo de trabajo: En esta segunda etapa se definen los flujos de trabajo, el contenido y las interacciones de la aplicación, obteniendo maquetas y prototipos. En este punto se puede hacer uso de un software de prototipado.

3. Diseño gráfico: A partir de las fases 1 y 2 se diseña el aspecto gráfico de la aplicación (interfaces de usuario, guías de estilo, etc.) a través de software específico.

4. Desarrollo: En esta última etapa se implementa la aplicación a través del IDE apropiado. Durante esta etapa es fundamental la realización de pruebas que permitan detectar aquellos fallos tanto en el código de la aplicación como en la propia funcionalidad de la misma.

1.1 Xamarin

Es una herramienta implementada por Microsoft con gran versatilidad en el desarrollo multiplataforma para la creación de aplicaciones nativas, es decir, las aplicaciones se pueden desarrollar tanto para el sistema Android como para los sistemas iOS y Microsoft. La implementación de *Apps* en Xamarin se realiza a través del lenguaje C# (C Sharp) y de una biblioteca clases incorporada para ser utilizada por los programadores.

Xamarin, en su versión original, es un IDE comercial incorporado en Visual Studio Professional. Sin embargo, se puede encontrar una versión gratuita llamada Xamarin Studio Community para sistemas Windows y Mac, abierta para estudiantes, desarrollo Open Source y pequeños equipos.

Aunque Xamarin no será la herramienta utilizada en el desarrollo de *Apps* de este módulo, es importante conocer su existencia y el gran potencial que puede llegar a ofrecer.

1.2 Android Studio

Esta herramienta fue desarrollada por Google y lanzada inicialmente en diciembre de 2014. Está programada en Java y es multiplataforma, es decir, puede ser utilizada bajo cualquier sistema operativo. A través de este IDE, se desarrollan aplicaciones para sistemas Android, el cual se encuentra presente en un gran número de *smartphones* y *tablets*.

Con Android Studio vienen incorporados los siguientes módulos:

- El propio IDE Android Studio.
- Android SDK Tools, que proporciona una serie de herramientas para desarrollar y depurar programas.
- Android Platform Tools, que permite realizar la compilación del código implementado y la generación de paquetes con extensión apk para ser ejecutados en Android.
- La imagen (logotipo) del sistema operativo Android.

Durante este módulo será Android Studio la herramienta que se va a utilizar en la implementación de aplicaciones para móviles y realización de prácticas. Se hace necesario, por tanto, proceder a la instalación del IDE Android Studio.

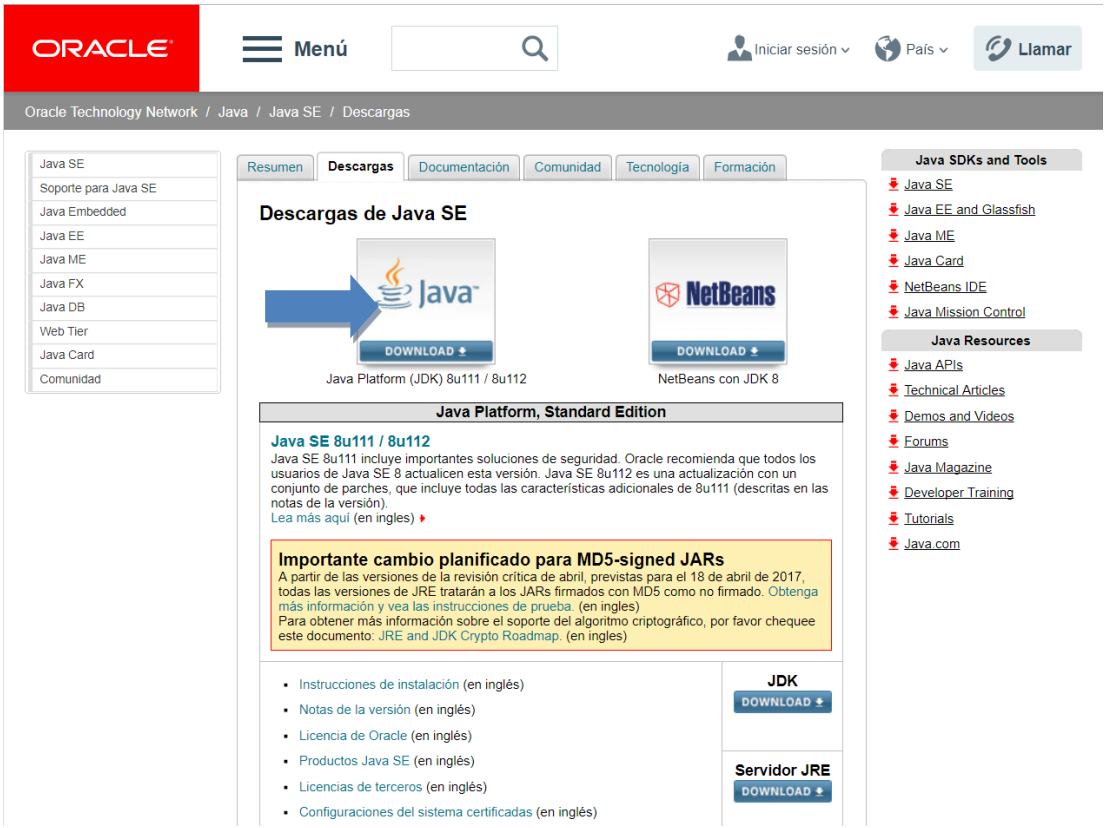
ENLACE DE INTERÉS

En el siguiente enlace encontrarás la web de Introducción a Android Studio:

https://developer.android.com/studio/intro?hl=es-419

1.2.1 Descargar JDK y Android Studio

El primer paso para la instalación de Android Studio es obtener el JDK (Java Development Kit) para realizar la compilación del código. Oracle compró la empresa Sun y, consecuentemente, Java pasó a ser propiedad de Oracle:

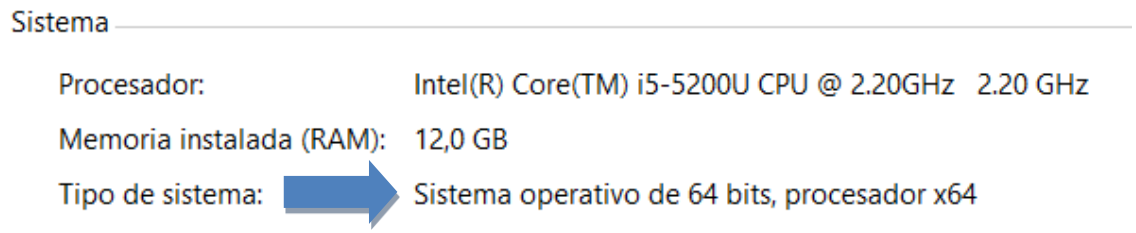


ENLACE DE INTERÉS

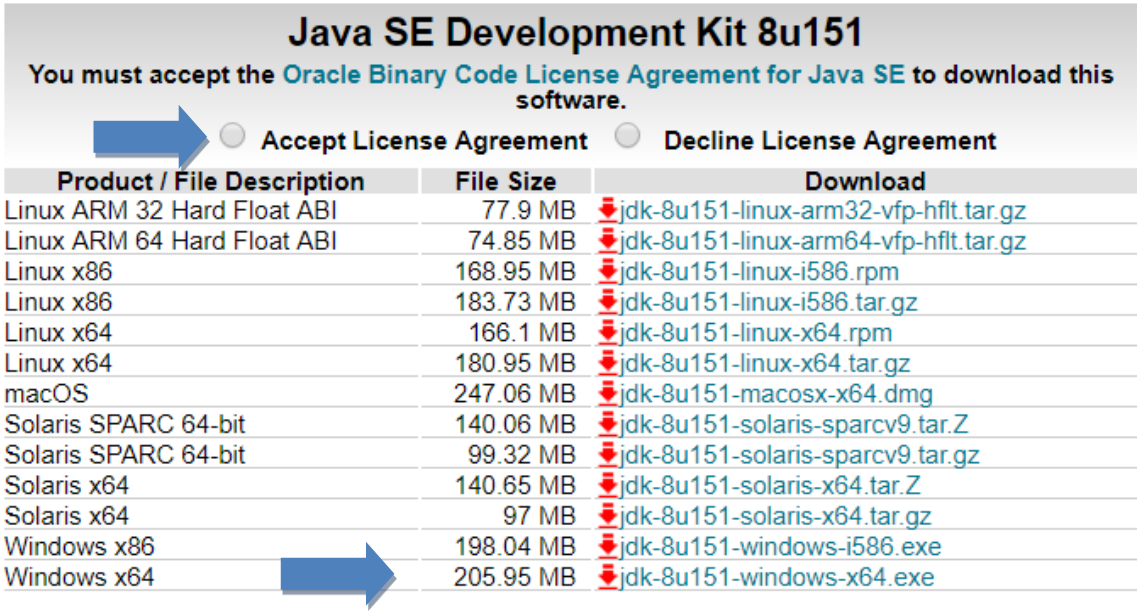
A continuación, encontrarás el enlace de descarga de JDK:

https://www.oracle.com/technetwork/es/java/javase/downloads/index.html

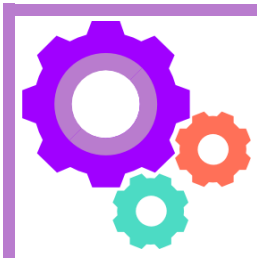
Tras pulsar sobre Java Platform (JDK) aparece una nueva página web con los diferentes sistemas operativos donde se desea instalar, su arquitectura, la ocupación en MB y el fichero JDK asociado. Es importante, antes de llevar a cabo la descarga, conocer estas características del sistema operativo en donde se desee instalar el JDK. Por ejemplo, en el caso Windows, dentro de la opción "Sistema", indica la arquitectura del microprocesador.



Con esta información, y en el caso del ejemplo anterior, se debe seleccionar el JDK para el sistema Windows x64. Es necesario aceptar los términos de la licencia para realizar la descarga y su posterior instalación.



Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.9 MB	jdk-8u151-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.85 MB	jdk-8u151-linux-arm64-vfp-hflt.tar.gz
Linux x86	168.95 MB	jdk-8u151-linux-i586.rpm
Linux x86	183.73 MB	jdk-8u151-linux-i586.tar.gz
Linux x64	166.1 MB	jdk-8u151-linux-x64.rpm
Linux x64	180.95 MB	jdk-8u151-linux-x64.tar.gz
macOS	247.06 MB	jdk-8u151-macosx-x64.dmg
Solaris SPARC 64-bit	140.06 MB	jdk-8u151-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.32 MB	jdk-8u151-solaris-sparcv9.tar.gz
Solaris x64	140.65 MB	jdk-8u151-solaris-x64.tar.Z
Solaris x64	97 MB	jdk-8u151-solaris-x64.tar.gz
Windows x86	198.04 MB	jdk-8u151-windows-i586.exe
Windows x64	205.95 MB	jdk-8u151-windows-x64.exe



PARA SABER MÁS

En Java se destacan las herramientas JDK, necesaria para la compilación del código, y la Máquina Virtual Java (conocida como JVM), necesaria para la ejecución de los programas, garantizando la portabilidad de las aplicaciones Java.

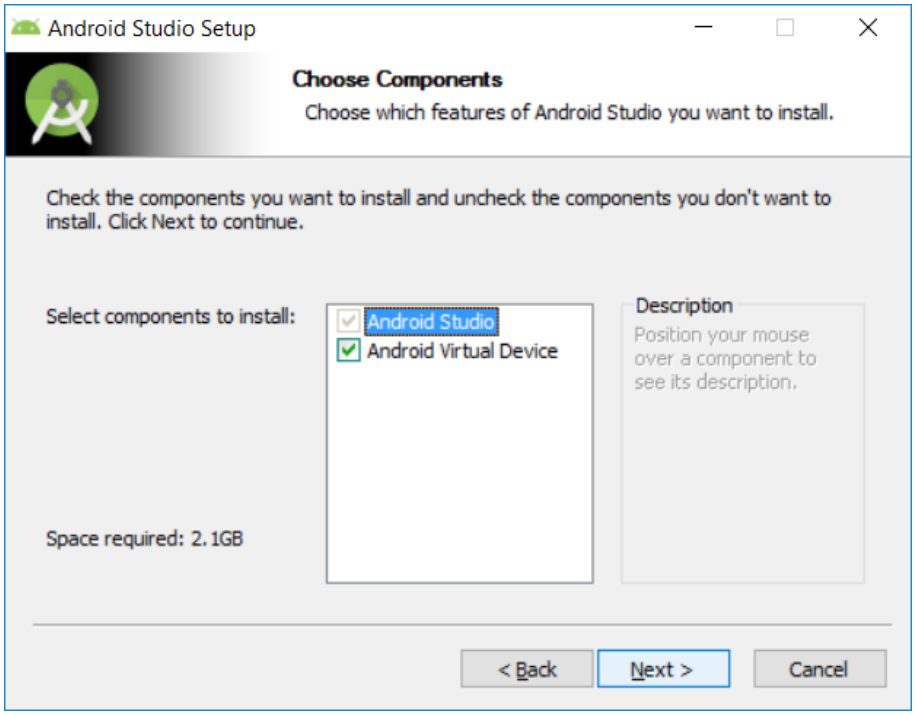
Una vez se ha descargado e instalado el JDK correspondiente, se procede a descargar Android Studio. En la actualidad, la última versión es la 3.0 y se encuentra en el siguiente enlace:
https://developer.android.com/sdk/installing/studio.html

Esta página web detecta el sistema operativo por el que se está accediendo y descarga la versión para dicho sistema. Es necesario aceptar los términos y las condiciones antes de realizar la descarga.

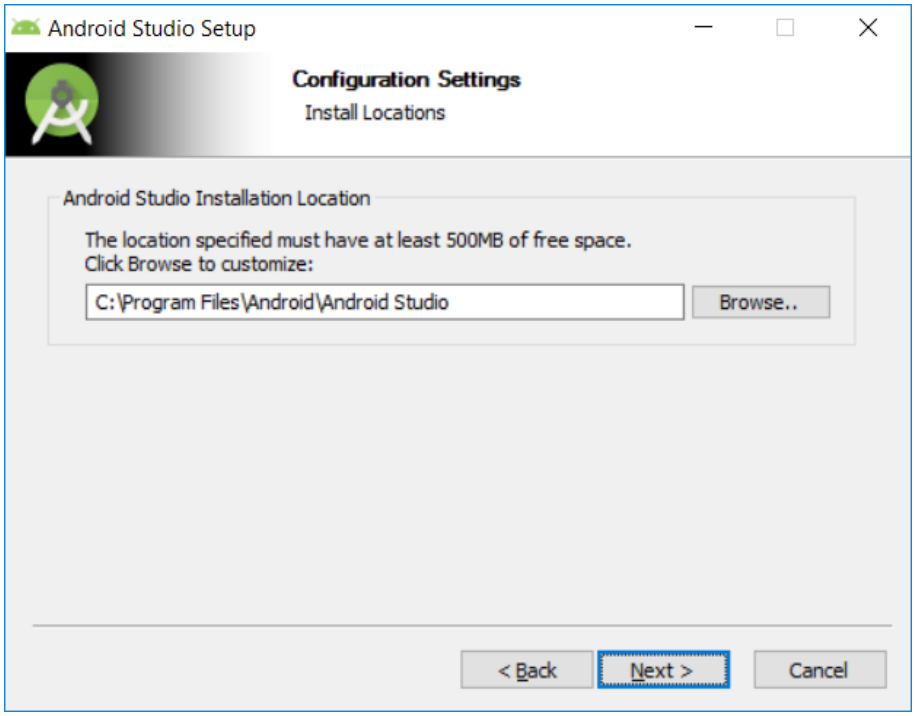
1.2.2 Instalar Android Studio

Tras la descarga del fichero de instalación de Android Studio, se procede a su instalación mediante la ejecución del fichero .exe descargado. La primera interfaz que aparece muestra la bienvenida al proceso de instalación:

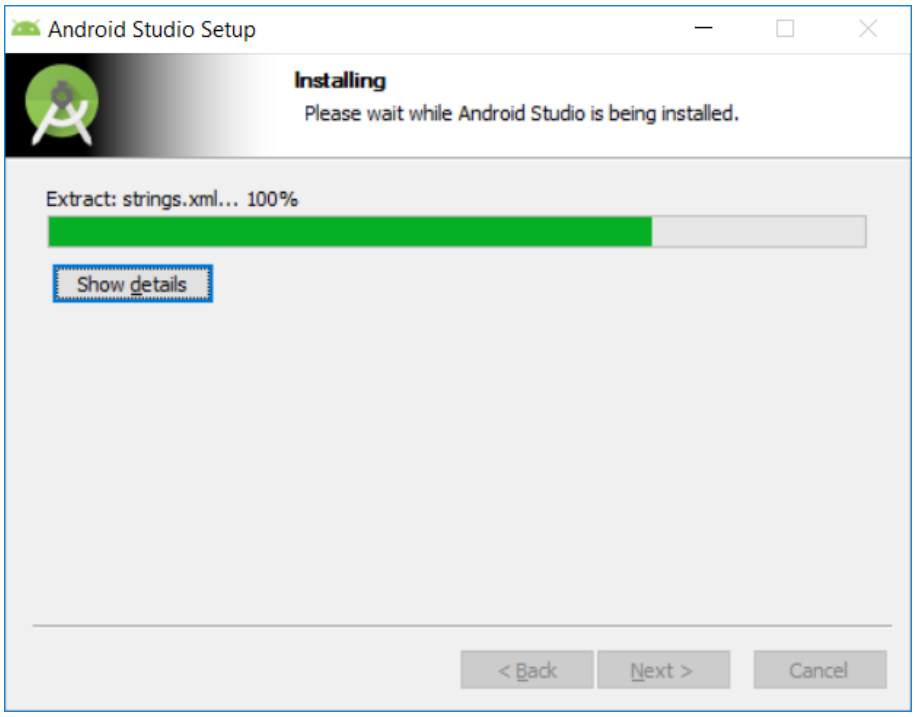
Tras pulsar sobre "Next>" aparece la siguiente interfaz de la instalación:



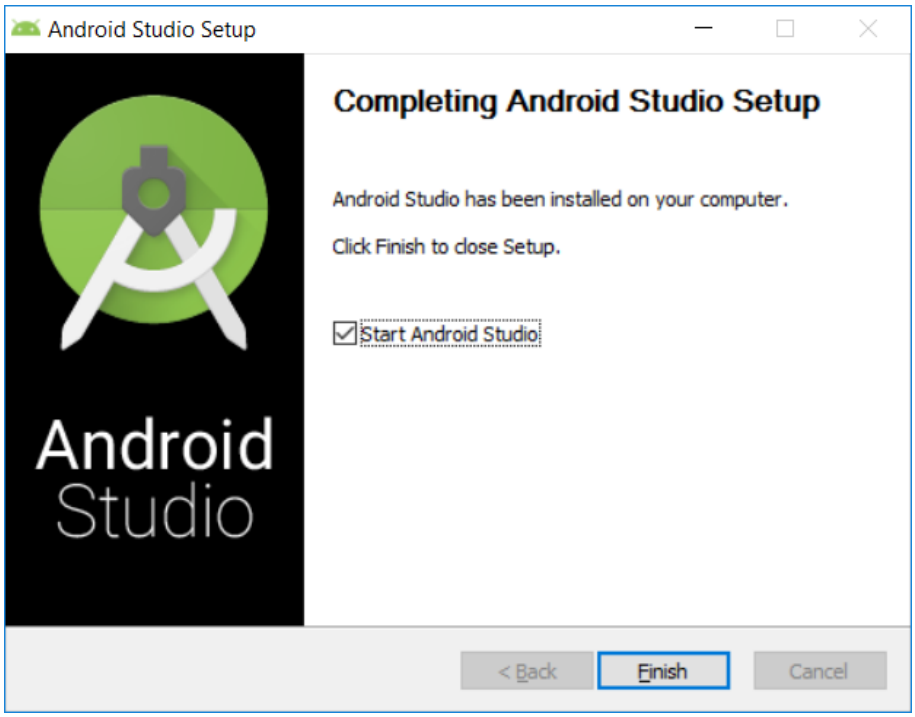
Se dejan seleccionados todos los elementos por defecto. Se pulsa "Next" tras el cual aparece la interfaz para seleccionar el directorio destino donde se va a instalar Android Studio. Se recomienda dejar por defecto y volver a pulsar "Next>".



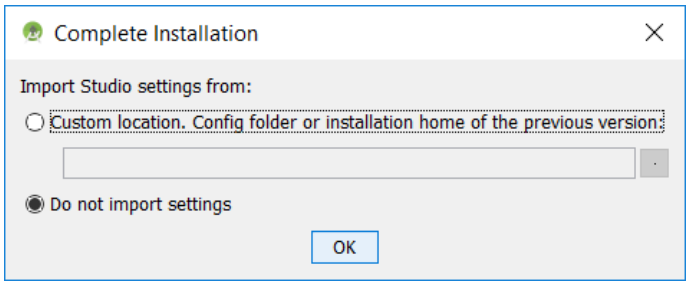
La siguiente interfaz muestra el directorio donde aparece instalado Android Studio en el menú de selección de aplicaciones de Windows. Luego, se pulsa "Install".



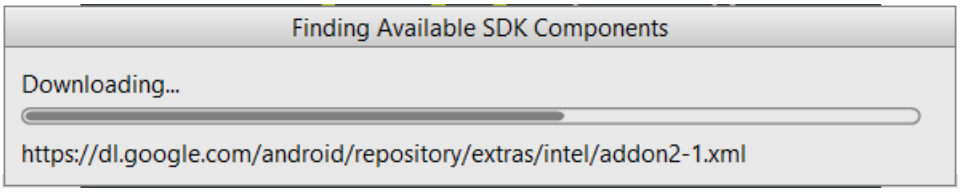
La última interfaz informa de la correcta instalación del IDE. En ella aparece un *check* seleccionado por defecto que permitirá abrir Android Studio tras pulsar sobre "Finish".



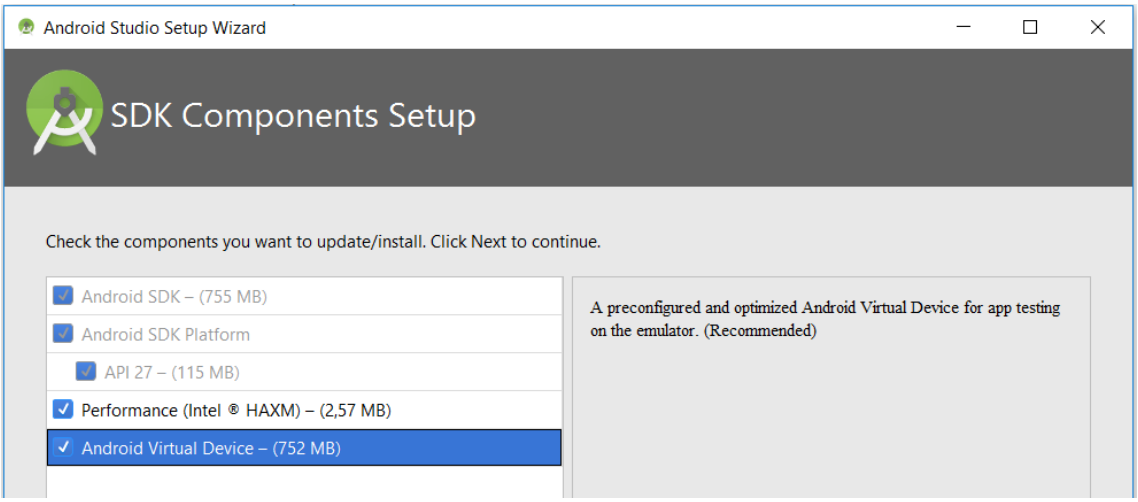
Si es la primera vez que se arranca Android Studio en el equipo, pregunta si se desea importar configuraciones previas. En el caso actual, se deja fija la opción de no importar nada y se pulsa "OK".



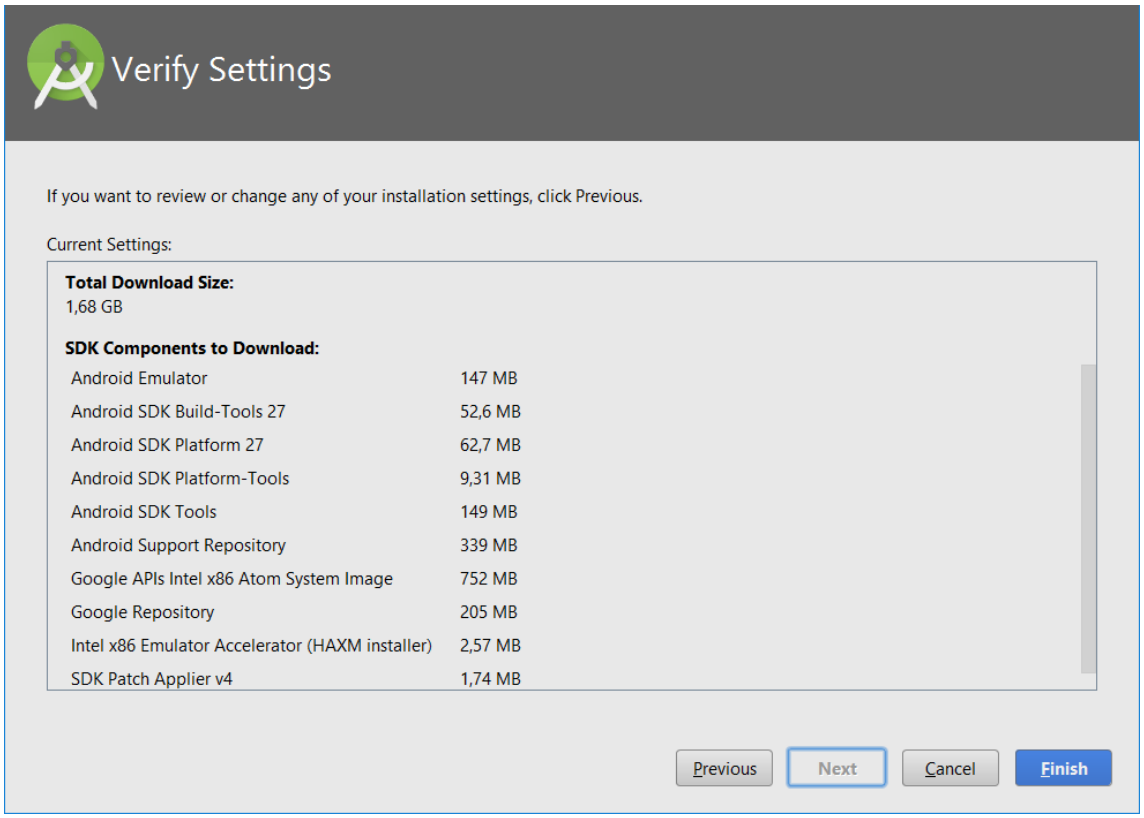
Se puede observar cómo, justo antes de que Android Studio muestre su interfaz principal, realiza una comprobación sobre los componentes SDK que están disponibles en los repositorios de Android en la web.



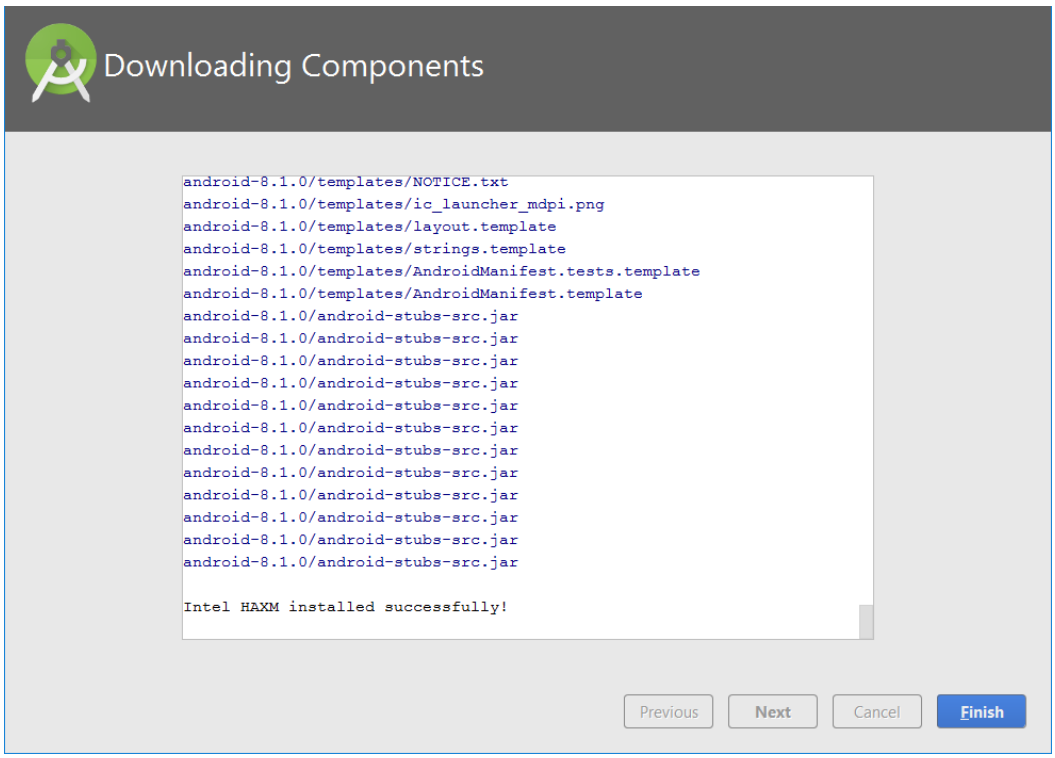
Una vez realizada esta comprobación, Android Studio muestra un *wizard* de bienvenida. En la interfaz Welcome se pulsa sobre "Next", en "Install Type" se deja por defecto "Standard" y se vuelve a pulsa sobre "Next". En "Select UI Theme" se elige el tipo de diseño preferido y se pulsa sobre "Next". Por último, sobre la interfaz SDK Components Setup se selecciona todos y se pulsa en "Next".

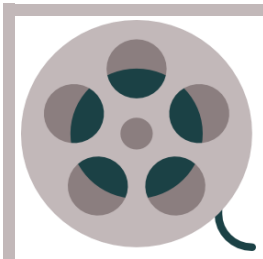


La siguiente interfaz hace un resumen de los componentes de SDK que van a ser instalados. Para ello, se pulsa en "Finish".



Los componentes mostrados en la figura comienzan a descargarse de forma automática y posteriormente serán instalados. En este punto se requiere un poco de paciencia ya que puede conllevar un tiempo considerable descargar e instalar todos los componentes SDK. Una vez finalizado, se pulsa en "Finish".

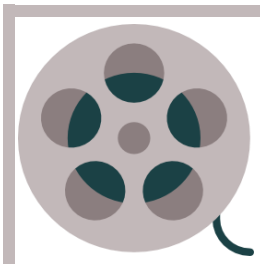




VIDEO DE INTERÉS

En el siguiente vídeo podrás visualizarás la instalación de Android Studio:

https://www.youtube.com/watch?v=BhkljH1krts



VIDEO DE INTERÉS

En el siguiente vídeo encontrarás la instalación de SDK Manager de Android Studio:

https://www.youtube.com/watch?v=wsdTJzfPbbo

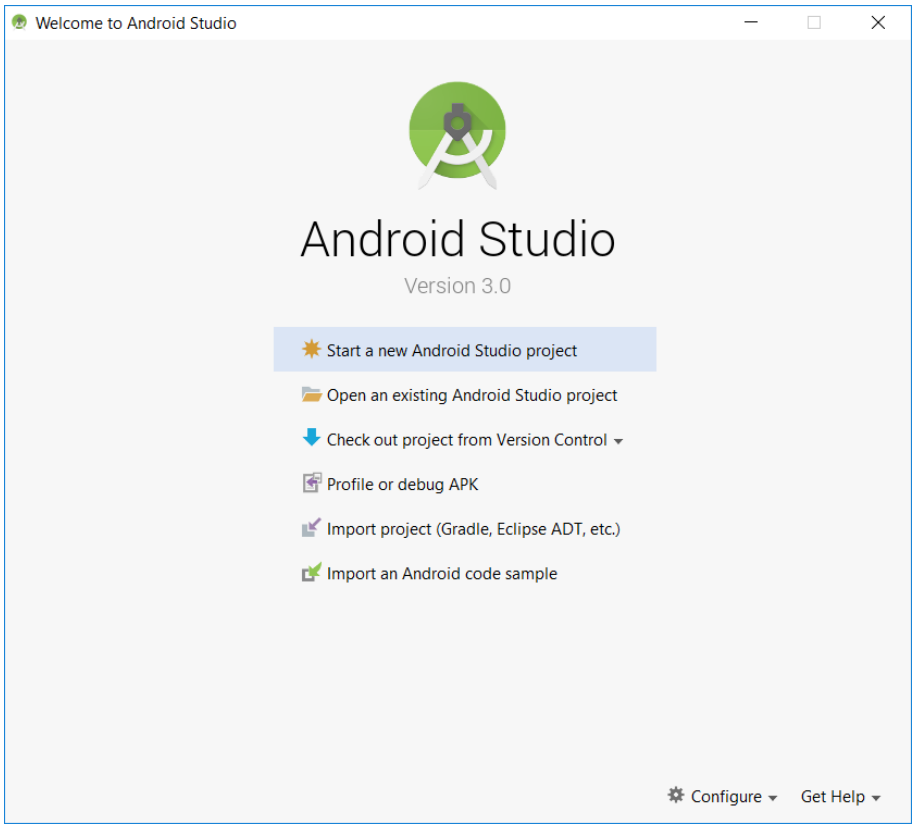
2. DESARROLLO DEL CÓDIGO

La programación en Android Studio se realiza a través de los lenguajes Java y XML, por tanto, la creación y el desarrollo del código debe cumplir los requisitos y la normativa que estos lenguajes aplican. La mejor manera de comprender la implementación paso a paso de una aplicación software es a través de ejemplos.

Por todo ello, llegado a este punto, es importante haber instalado Android Studio en el equipo de trabajo, ya que será la herramienta principal del módulo y a través de la cual se irán realizando ejemplos y proponiendo ejercicios y prácticas que van a permitir, por un lado, adaptarse a este IDE, y por otro, aprender a desarrollar aplicaciones móviles para Android tanto en el lenguaje XML como en Java.

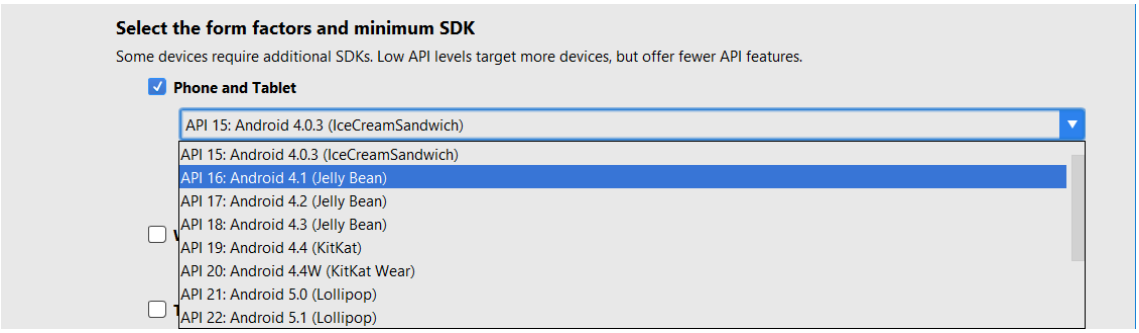
2.1 Creación de un proyecto en Android Studio

Tras la instalación de Android Studio se procede a crear el primer proyecto para una aplicación móvil. La interfaz que se muestra es la siguiente donde se pulsará sobre “Start” a new Android Studio Project:

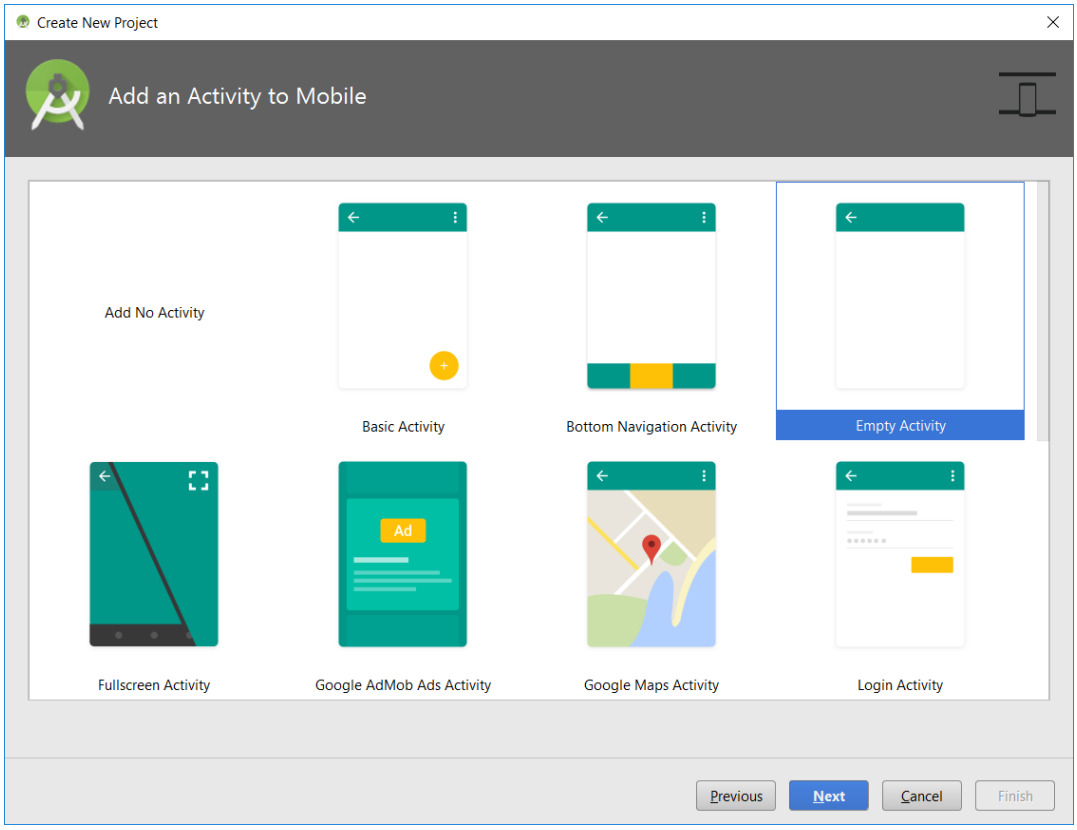


A continuación, se le pide al usuario que escriba un nombre para el proyecto (incluyendo el dominio de la compañía) y la localización del proyecto dentro del disco duro (se puede dejar todo por defecto), y se pulsa sobre "Next".

En la siguiente interfaz Target Android Devices, se pide al usuario que seleccione el mínimo SDK con el que funcionará la aplicación. Una versión compatible con los móviles y las *tablets* es la API 16 (Jelly Bean). Luego se pulsa "Next".



El siguiente paso consiste en seleccionar el tipo de actividad que contendrá la interfaz de usuario de la aplicación que se va a desarrollar. Puede observarse cómo existen diferentes actividades como, por ejemplo, "Actividad básica", "Actividad botón de navegación", "Actividad vacía", "Actividad de Google Maps", "Actividad de login", "Actividad de configuración", etc.



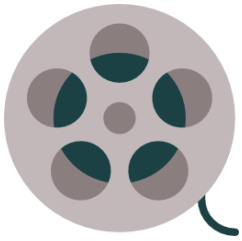
Se procede a seleccionar la actividad *Empty Activity*, se pulsa "Next", se escribe el nombre de la actividad y se pulsa nuevamente en "Next". En este momento comienza a llevarse a cabo la creación y configuración del proyecto cuyo fin es la preparación del IDE y comenzar a desarrollar la aplicación móvil. Es importante conocer las diferentes partes en las que se divide la interfaz de Android Studio. Los siguientes puntos resumen cada una de ellas:

- Explorador de proyectos: Situado a la izquierda, donde aparecen los directorios y ficheros del proyecto.
- El panel central: Dividido en las pestañas de diseño (*Design*) y de código (*Text*). En diseño se observan los componentes y un móvil con la interfaz que se está implementando. Los ficheros que componen una aplicación en Android Studio son de tipo XML (aspecto visual) y Java (comportamientos). A la derecha de este panel se encuentran las propiedades y los atributos de los componentes de la aplicación.

COMPRUEBA LO QUE SABES

¿Por qué crees que las aplicaciones en Android Studio se programan haciendo uso de dos lenguajes como son Java y XML?

Coméntalo en el foro de la unidad.



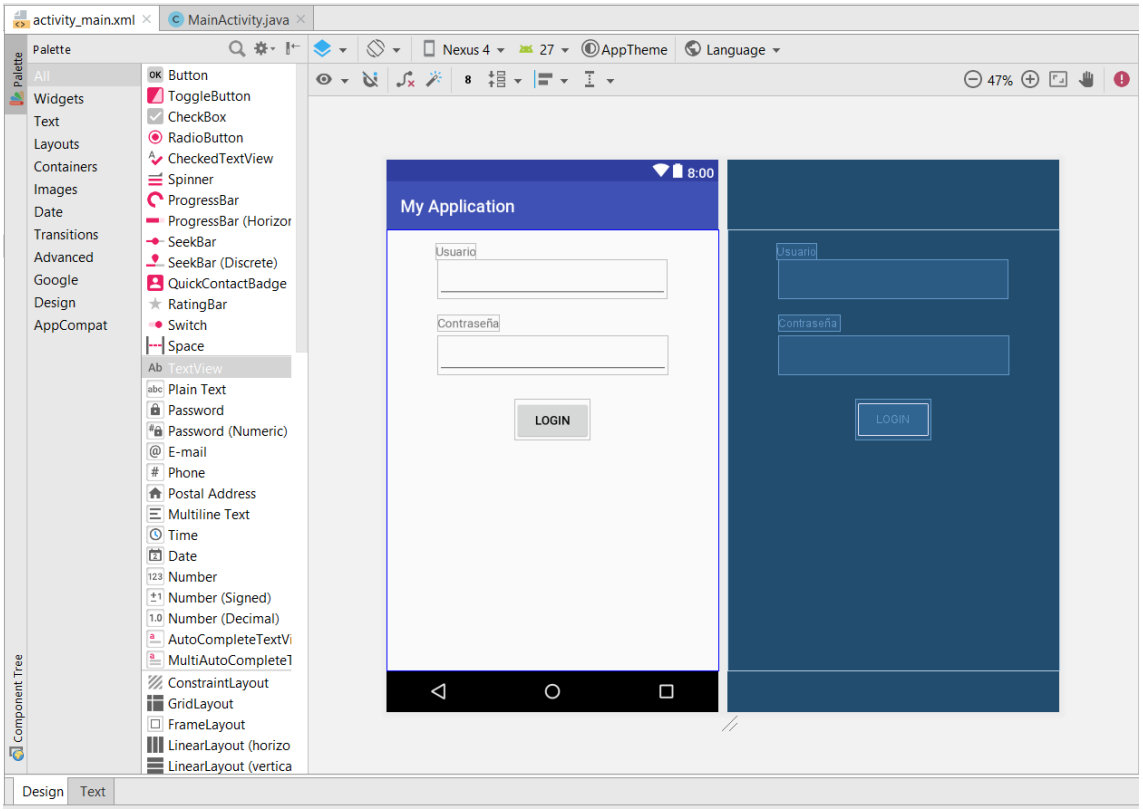
VIDEO DE INTERÉS

En el siguiente enlace encontrarás un vídeo sobre las partes de un proyecto en un dispositivo virtual Android:

https://www.youtube.com/watch?v=MwCjokz-sA4

2.2 Primer ejemplo en Android Studio

Se pretende realizar un pequeño diseño que represente una interfaz de login. Para ello, son necesarios 5 componentes: dos etiquetas (usuario y contraseña), dos campos de texto (para cada una de ellas) y un botón para realizar el login. Mediante Android Studio es sencillo diseñar interfaces de aplicaciones arrastrando los componentes necesarios a la pantalla del móvil. Estos componentes se ubican en el listado de la izquierda de la imagen.



A través de las propiedades de los componentes (lado derecho), es posible modificar sus valores. En el ejemplo mostrado, el textView para el usuario y la contraseña tienen en la propiedad text el valor "Usuario y Contraseña", respectivamente. El botón tiene como text el valor "LOGIN". Es importante observar cómo se va desarrollando código automáticamente (ver pestaña Text) en el fichero XML mientras se va diseñando la interfaz de usuario de la aplicación móvil. Este fichero puede ser modificado de forma manual para añadir o modificar propiedades, componentes, etc. Fundamentalmente, el código en Android Studio se divide en dos ficheros: XML activity_main.xml y MainActivity.java.

El siguiente fragmento de código se corresponde con parte del XML donde se observa la creación de 3 componentes: la etiqueta contraseña ("TextView") el campo para escribir la contraseña ("EditText") y el botón Login ("Button").

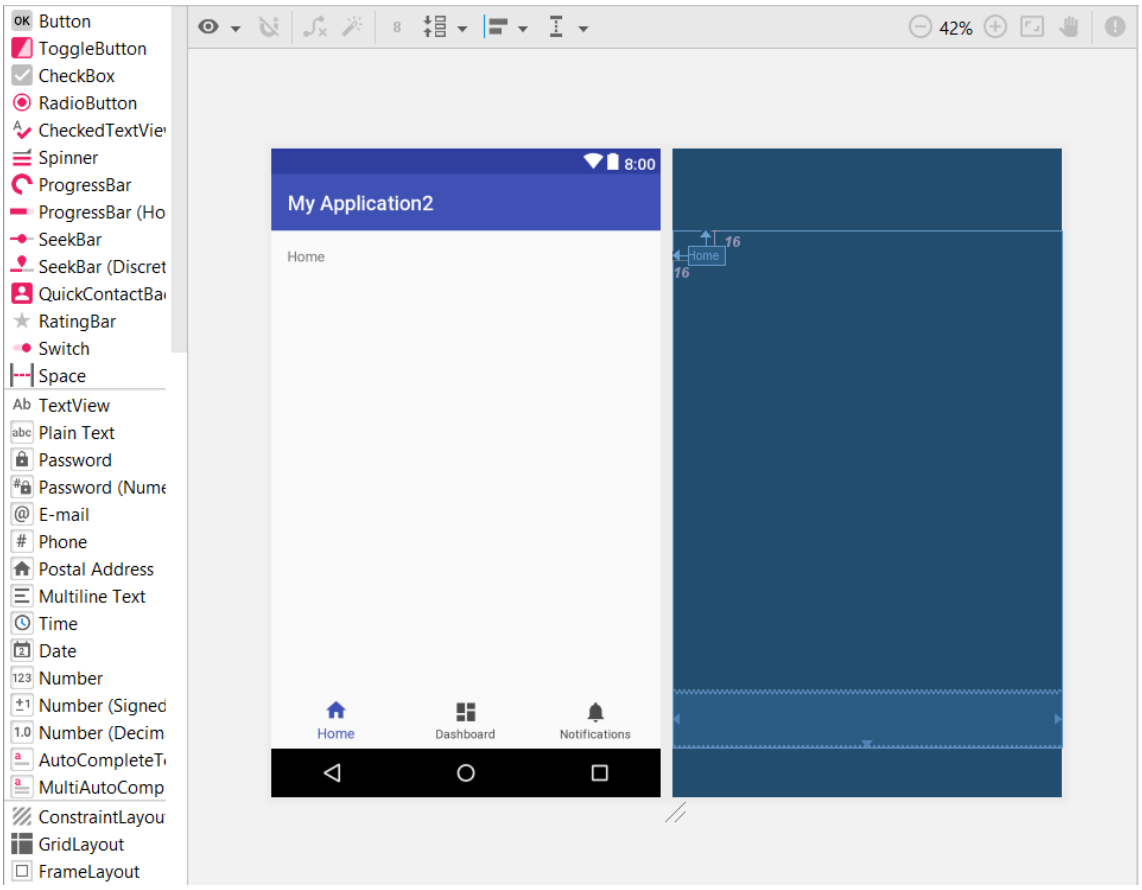
```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText2"
    android:layout_alignStart="@+id/editText2"
    android:layout_below="@+id/editText2"
    android:layout_marginTop="18dp"
    android:text="Contraseña" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_alignStart="@+id/textView2"
    android:layout_below="@+id/textView2"
    android:ems="13"
    android:inputType="textPersonName" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/editText3"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="26dp"
    android:text="LOGIN" />
```

El fichero Java no ha sufrido modificaciones ya que solo se han realizado cambios en el diseño de la interfaz, no en el comportamiento de la misma.

```
package com.example.josecarlos.myapplication;

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

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

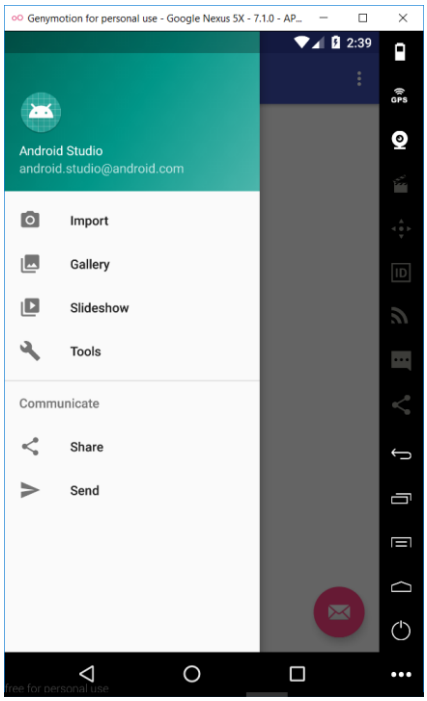
Al avanzar en los contenidos se irá profundizando en el código y en las diferentes alternativas que proporciona Android Studio para XML y Java.

A continuación, se muestra un ejemplo de aplicación básica con la plantilla Bottom Navigation Activity creada de forma automática:



Este tipo de plantilla es óptima para aplicaciones Android en las que se desee incorporar navegación del tipo Home, Dashboard, Notifications.

Otra de las plantillas que afortunadamente incluye Android Studio se muestra en el siguiente ejemplo (Navigation Drawer):



3. COMPILACIÓN, PREVERIFICACIÓN, EMPAQUETADO Y EJECUCIÓN

En esta sección se pretenden estudiar cuatro conceptos importantes relacionados con el desarrollo de aplicaciones en Android Studio. Sería interesante haber implementado el ejemplo anterior (interfaz Login) para poder ser aplicados en la práctica siguiendo las acciones descritas en cada uno de ellos.

3.1 Compilación

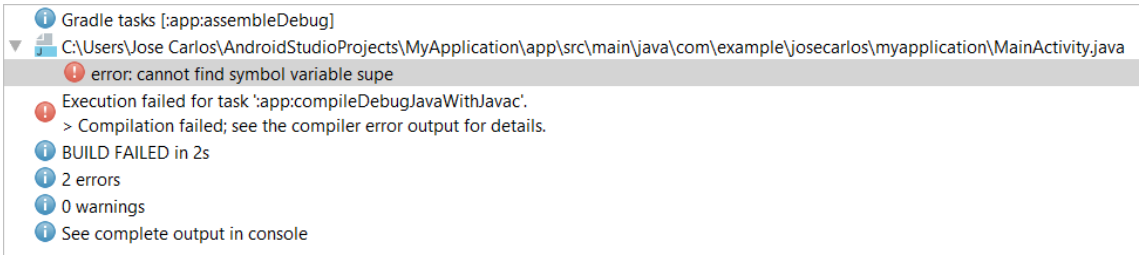
El lenguaje con el que se desarrolla una aplicación es entendible por el usuario, pero no por el equipo informático (PC, dispositivo móvil, tablet, etc.). Para ello, es necesario traducir ese lenguaje a otro que sí lo sea. Por tanto, el proceso de compilación se puede definir como la traducción del código fuente de la aplicación (lenguaje Java, por ejemplo) en código ejecutable (lenguaje máquina) a través de un compilador.

En Android Studio el proceso de compilación se realiza a través de Gradle, que es un plugin para compilar, construir y depurar programas. La acción para compilar una aplicación se origina al pulsar sobre Run (símbolo ▸). En ese preciso momento, el código fuente (lenguaje entendible por el usuario) es analizado por el compilador obteniendo dos posibles resultados: el código es correcto y se procede a su ejecución o el código no es correcto y muestra los errores detectados. No obstante, si existiesen errores sintácticos en el código, Android Studio lo indica identificando la palabra o palabras erróneas o bien a través de iconos representativos en la línea afectada sin necesidad de pasar por el compilador.

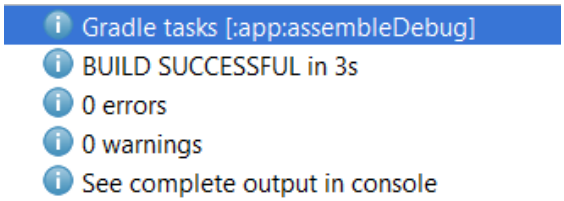
```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        supe.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

En este ejemplo, se observa cómo en la palabra *supe* existe un error, ya que la palabra debería ser *super*. Como tal, se identifica en este caso de color rojo y, además, un icono alertando la línea correspondiente. Además, este error conlleva que exista otro error en onCreate, que espera *super* y no *supe*.

Como se ha descrito anteriormente, en este caso no ha sido necesario el proceso de compilación. No obstante, si se pulsa Run, se lleva a cabo la compilación y el Gradle, en este caso, mostraría lo siguiente:



Corrigiendo el error anterior y realizando la compilación nuevamente, el Gradle muestra información sin aparecer error alguno:



3.2 Preverificación

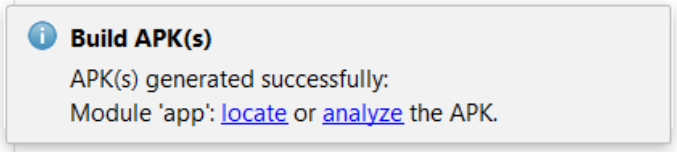
La preverificación es un proceso mediante el cual se examina el código Java de la aplicación para comprobar si no se viola ninguna restricción de seguridad de la plataforma en la cual va a ser ejecutada. Las máquinas virtuales Java verifican en el momento de la carga de una clase (fichero Java) por razones de seguridad. Estas verificaciones incluyen, entre otras, el acceso a la memoria fuera de su espacio de ejecución.

Esta operación es costosa en recursos y se ejecuta fuera de la máquina virtual. Para llevarse a cabo, es necesario usar la utilidad de preverificación, la cual modifica el fichero Java compilado para que, en caso de problemas, la carga de la clase se interrumpa. En este momento, y para no complicar más el aprendizaje del desarrollo de aplicaciones en Android Studio, solamente se hace referencia a la preverificación como el concepto teórico descrito.

3.3 Empaquetado

El empaquetado de una aplicación en dispositivos Android es un proceso que consiste en recopilar todos los ficheros que la componen y crear el archivo APK para ser ejecutado en el dispositivo final. Existen 2 formas de creación del APK: APK Debug y Signed APK.

En el primero de ellos (APK Debug), la aplicación es empaquetada en un APK que podrá ser utilizado en dispositivos virtuales o en un móvil personal para realizar pruebas y "jugar" con la aplicación. Este fichero no sería válido para subirlo a Google Play (plataforma de distribución digital de aplicaciones móviles para dispositivos Android y tienda virtual en línea desarrollada y operada por Google). Para general el APK en Android Studio se accede a la opción Build → Build APK(s) y la herramienta procede al empaquetado de la aplicación en un fichero APK ejecutable en un teléfono Android con nombre app-debug.apk.



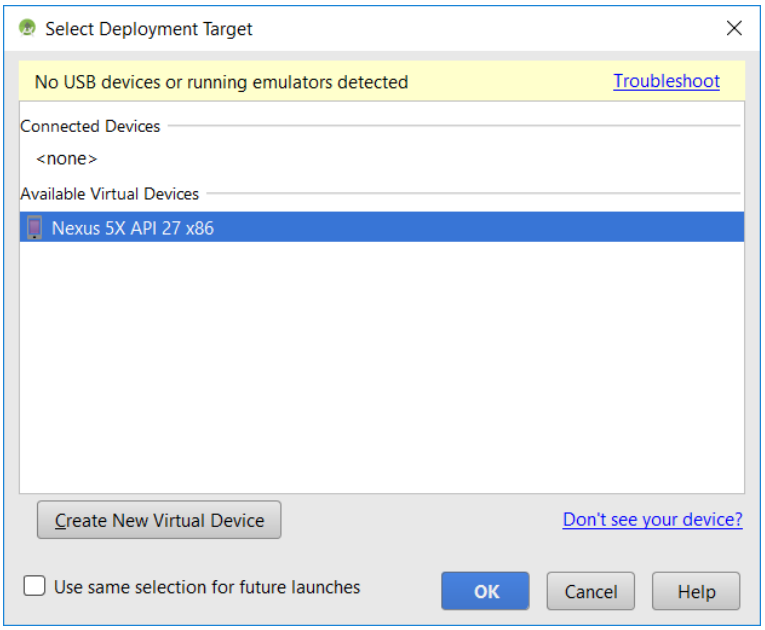
En el segundo de ellos (Signed APK) se firma digitalmente la aplicación, siendo posible subir la aplicación a Google Play. Para generar este fichero APK se debe acceder a la opción Build → Generate Signed APK y rellenar los campos necesarios para proceder a la firma digital. En este momento solo va a ser interesante generar el APK Debug para probar las aplicaciones en un teléfono móvil personal o tablet con sistema operativo Android.

3.4 Ejecución

Este proceso se encuentra íntimamente relacionado con la compilación ya que, tras pulsar sobre Run y compilar correctamente el código desarrollado, se lleva a cabo de forma automática la ejecución de la aplicación. Por tanto, la acción Run realiza 2 procesos: la compilación del código y la ejecución de la aplicación ante la ausencia de errores.

La ejecución de las aplicaciones en Android Studio se realiza mediante un emulador o bien a través del dispositivo móvil personal conectado al equipo por un USB. La primera de las ejecuciones suele ser bastante lenta y debido al gran número de ejecuciones que se realizan mientras se desarrolla código, se hace bastante pesado y desesperante. La segunda de las opciones suele ser más rápida, pero presenta cierta dificultad a la hora de configurar el móvil.

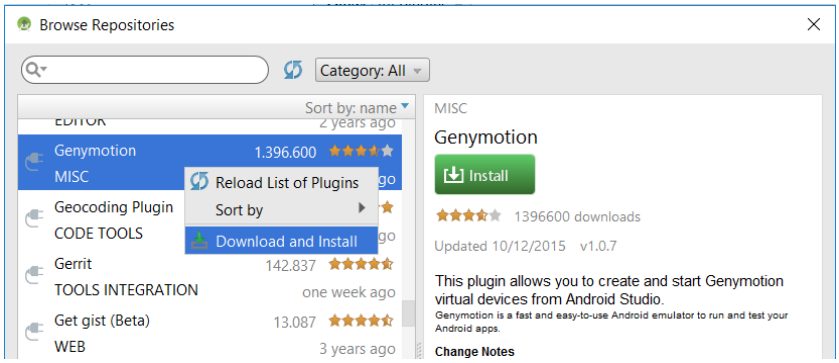
La siguiente imagen muestra el paso previo a la ejecución de la aplicación a través del simulador proporcionado por Android Studio.



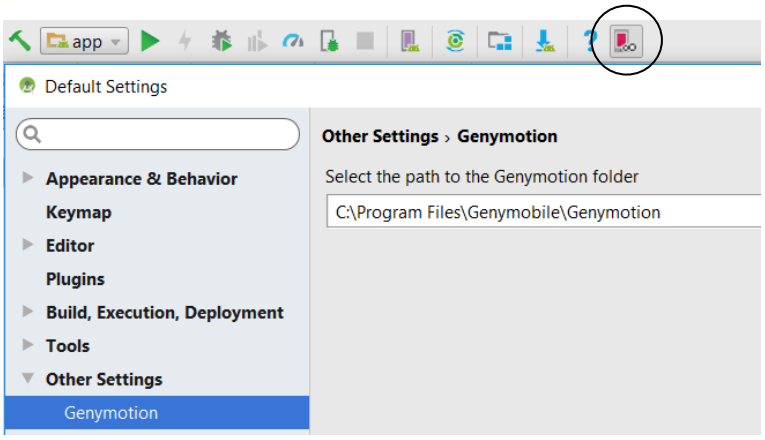
Estando configurado y conectado el dispositivo móvil por USB, aparecería en la imagen dentro de "Connected Devices".

Una solución bastante eficiente para paliar el problema de los recursos es instalar un emulador potente y rápido llamado Genymotion. Se trata de un emulador gratuito que funciona a través de máquinas virtuales. Se procede a la descarga e instalación de Genymotion en Android Studio.

Una vez descargado, se debe asociar Android Studio con este emulador. Para ello, se pulsa sobre la opción "File → Settings → Plugins → Browse Repositories". Luego se pulsa el botón derecho en Genymotion y sobre "Download and Install".

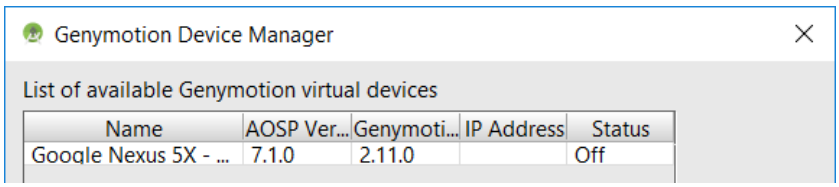


Tras ello será necesario reiniciar el IDE y ejecutar nuevamente la aplicación a través del icono de Genymotion.

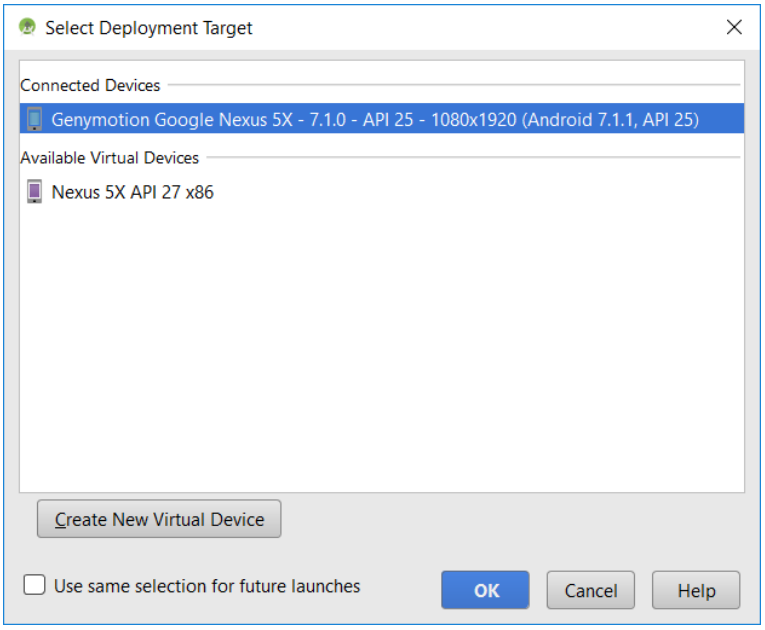


El IDE espera que el usuario indique la ruta donde se encuentra instalado el emulador. En este caso (para Windows) la que aparece en la figura. Una vez realizado el proceso, se ejecuta nuevamente sobre el icono de Genymotion y se accede a su web para registrarse. Tras el registro se realizará el login en el propio emulador, el cual pedirá la elección de un dispositivo móvil para emular. Se selecciona, por ejemplo, Google Nexus 5x. A partir de este momento, Genymotion comienza a instalar el emulador para dicho dispositivo.

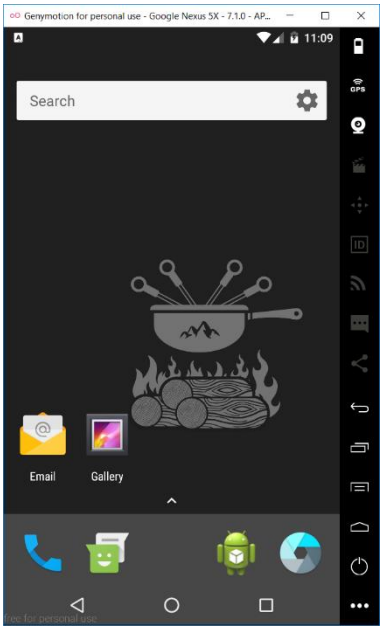
Se puede observar cómo pulsando sobre el icono de Genymotion ya aparece el dispositivo virtual instalado. Seleccionar y pulsar "Start".

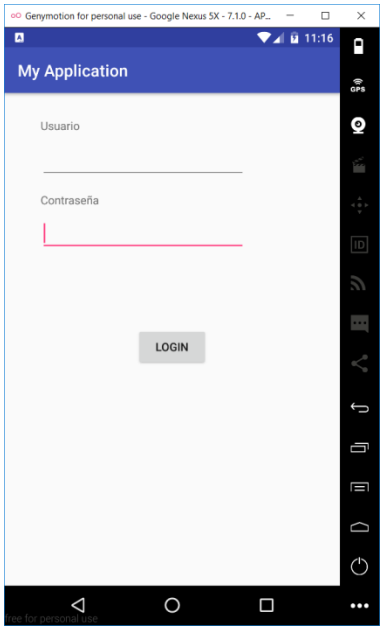


En este preciso instante aparece el emulador virtual de Nexus 5x listo para ser utilizado como emulador de las aplicaciones en Android Studio.



Tras la ejecución del emulador, aparece la aplicación Login en el dispositivo virtual Nexus 5x.





Con Genymotion la ejecución de las aplicaciones a través de Android Studio será mucho más rápida que a través de su propio emulador e incluso que en dispositivos móviles personales conectados al IDE.

BIBLIOGRAFÍA RECOMENDADA

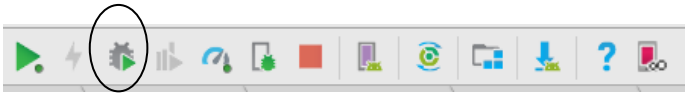
López Montalbán, I., Martínez Carbonell M. y Manrique Hernández, J. C. (2015). *Android. Programación Multimedia y de Dispositivos Móviles*. Garceta.

4. DEPURACIÓN

La depuración del código es un proceso que permite a los programadores detectar y solucionar errores en el código de la aplicación. Es lógico que al implementar una aplicación aparezcan fallos que deben ser descubiertos y solucionados. El proceso de compilación permite detectar errores sobre todo de tipo léxico. No obstante, es posible que al ser resueltos la aplicación compile sin problemas, pero el resultado de la ejecución no sea el esperado. Para ello, existen las técnicas de depuración, que permiten inspeccionar el código a partir de un cierto punto del código e ir observando mediante saltos de línea a línea el resultado de la ejecución en cada momento. El punto o los puntos a partir de los cuales se desea comenzar la depuración se les conoce como *breakpoints* o puntos de ruptura. Para crear un punto de ruptura basta con hacer clic sobre el margen de la línea o líneas involucradas (esfera roja).

```
7
8   public class MainActivity extends AppCompatActivity {
9
10      @Override
11      protected void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.activity_main);
14      }
15
16      public void pulsar(View view) {
17          EditText eT4 = (EditText)findViewById(R.id.editText4);
18          EditText eT5 = (EditText)findViewById(R.id.editText5);
19          if (eT4.getText().toString().equals(eT5.getText().toString())) {
20              ((EditText) findViewById(R.id.editText5)).setText("IGUALES");
21          }
22
23      }
24  }
```

Al proceso de depuración se le llama *debugger* y se ejecuta a través de un botón cercano a Run, que simula un bicho con un play.



Al iniciar el depurador de código, la interfaz de Android Studio muestra las diferentes opciones de depuración:

Las opciones más utilizadas son las 3 flechas azules, conocidas como "Step Over", "Step Into" y "Step Out", según el orden de aparición:

- Step Over: Se ejecuta la sentencia actual del código en la que el programa se encuentra detenidos en ese momento (*breakpoint*).
- Step Into: Se desplaza a la siguiente línea de código. Esta línea de código sería la que tendría la aplicación si estuviera escrita en un único bloque secuencial, sin llamadas a funciones u otros métodos de clase.

- Step Out: El nivel de depuración llega a la ejecución de métodos internos ya depurados por los creadores.

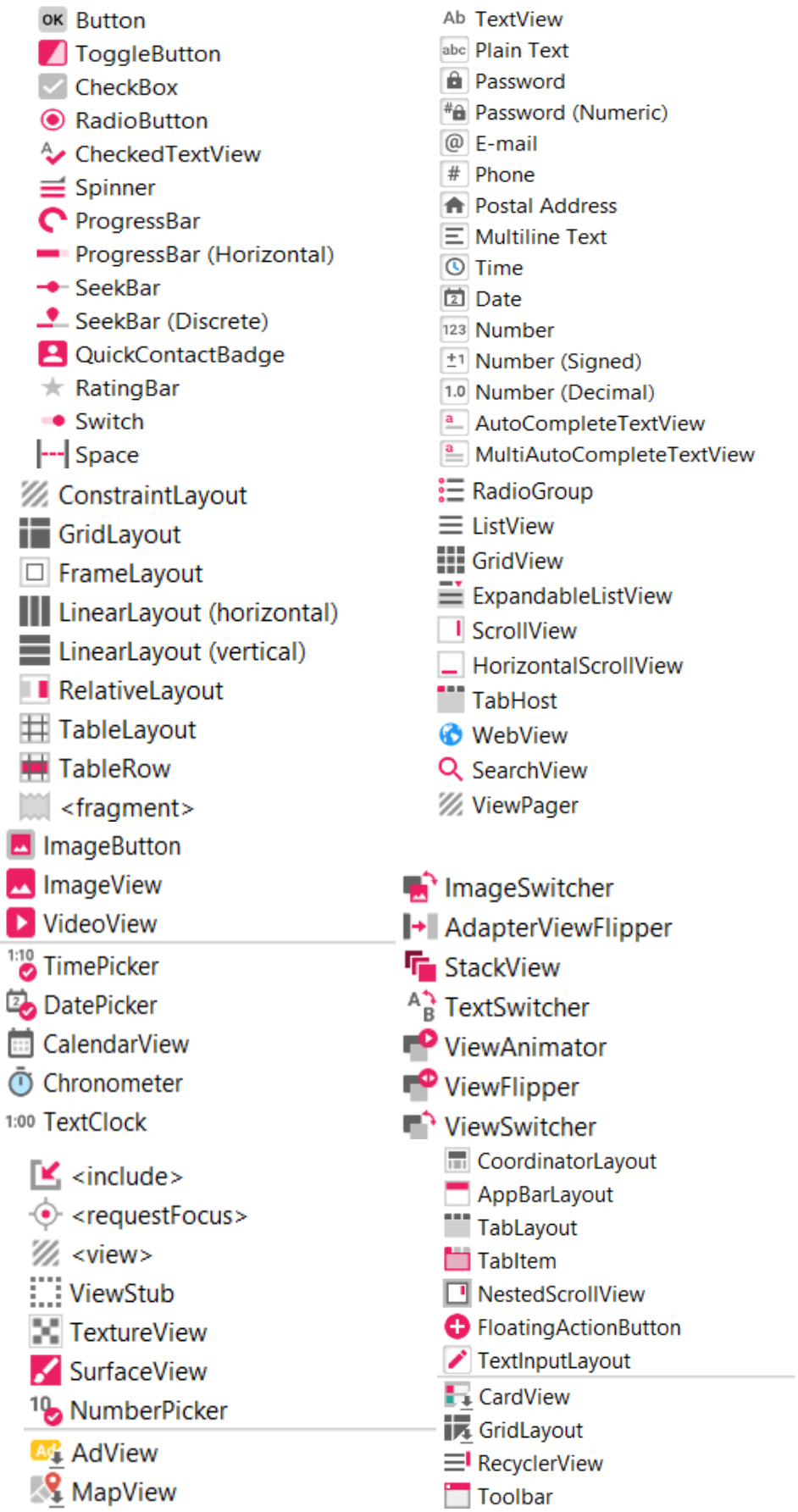
5. INTERFACES DE USUARIO. CLASES ASOCIADAS

A la hora de implementar software para dispositivos móviles es fundamental tener conocimiento sobre el diseño de interfaces de usuario y las clases van a ayudar a su desarrollo a través de componentes visuales y no visuales.

Dispositivo móvil
Fuente: https://pixabay.com/es/photos/iphone-plantilla-maqueta-iphone-6-500291/

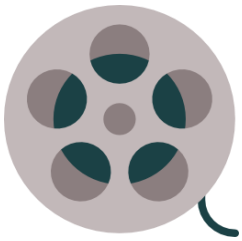
5.1 Interfaces de usuario

Una interfaz de usuario es la capa intermedia entre el usuario y el dispositivo o equipo donde se ejecuta la aplicación. En este sentido, la interfaz de usuario es la vía de comunicación para que los usuarios realicen un uso correcto de la aplicación. Por tanto, es fundamental que las interfaces sean sencillas, amigables, intuitivas y cómodas de usar, de modo que se facilite esta comunicación. Dependiendo del tipo de aplicación, las interfaces de usuario que la conforman tendrán unos componentes u otros. En Android Studio se presenta una lista bastante amplia de componentes que permiten al programador desarrollar interfaces de toda índole. Estas imágenes muestran los componentes proporcionados por Android Studio.



Componentes de Android Studio

Cada uno de estos componentes hace referencia a una clase llamada con igual nombre que el componente. En este sentido, existe una clase Button, otra clase CheckBox, otra clase RadioButton, etc., que serán utilizada por el desarrollador para crear las interfaces de usuario correspondientes. Cada una de estas clases posee una serie de atributos o propiedades que permiten, a la hora de desarrollar interfaces de usuario, darle forma y contenido a las mismas a través de los componentes.



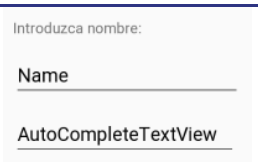
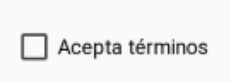
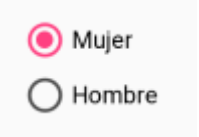
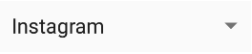
VIDEO DE INTERÉS

En el vídeo que encontrarás en el siguiente enlace podrás visualizar un detalle del editor gráfico de interfaces:

https://www.youtube.com/watch?v=61LHPqMNPsc

5.2 Clases asociadas

Para razonar en detalle la idea de clase y componente, se analizan varios de ellos considerados de los más habituales en las interfaces de usuario convencionales:

Componente	Clase/s	Descripción	Imagen
Botón	Button	Un botón para pulsar sobre él y generar una acción.	
Texto	TextView, PlainText, AutoCompleteTextView	Etiquetas de texto para dar información (editable o no).	
Casilla de verificación	CheckBox	Una opción que puede estar de forma activa o desactiva sin importar el número de elementos activos.	
Botón de opción	RadioButton, RadioGroup	Una opción que puede estar de forma activa o desactiva, siendo estos mutuamente excluyentes	
Switches	ToggleButton	Opción encendido o apagado, on/off, …	
Combo de selección	Spinner	Lista desplegable para seleccionar una opción de entre varias.	

Además de estos componentes, en Android Studio existen otros muchos bastante utilizados como DatePicker y TimePicker para la selección de una fecha y una hora, respectivamente; las imágenes y los botones imágenes que se corresponden con las clases ImageView e ImageButton; los Layouts, que permiten ubicar los elementos en la interfaz de una forma u otra, etc.

En los siguientes apartados se va a mostrar, a modo de ejemplo, el desarrollo de una interfaz de usuario para un dispositivo móvil a partir de Android Studio añadiendo cada uno de los componentes de la tabla y analizando algunas de las propiedades más habituales.

ENLACE DE INTERÉS

En el siguiente enlace encontrarás información de todos los componentes de Android Studio para el desarrollo de interfaces de usuario:

https://developer.android.com/reference/android/widget/package-summary.html

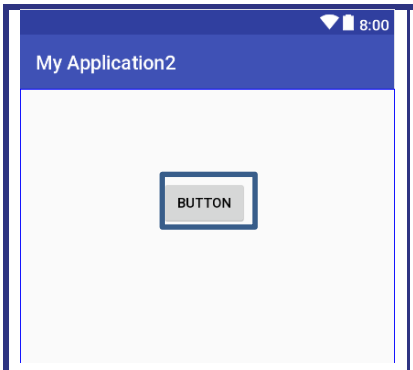
VIDEO DE INTERÉS

En el vídeo que encontrarás en el siguiente enlace podrás visualizar un detalle de la clase y container ConstraitLayout:

https://www.youtube.com/watch?v=8J1-Y1Fv-vI

5.2.1 La clase Button

Esta clase hace referencia al componente "botón". En el diseño de la interfaz de usuario de ejemplo, se añade un simple botón. A partir de ahí, se van a analizar las propiedades más utilizadas. Al crear un botón en Android Studio, aparece lo siguiente:

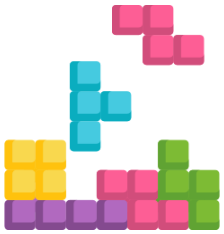



```
<Button
 android:id="@+id/button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="148dp"
 android:layout_marginTop="112dp"
 android:text="Button"

app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent"
/>
```

El componente "Button" queda definido por las siguientes propiedades:

- id: Será su identificador para poder acceder a él a través del código.
- Text: El texto del botón. Inicialmente, aparece Button, pero puede ser cambiado por el texto deseado.
- layout_marginLeft y layout_marginTop: Distancia entre el margen izquierdo y el botón y el margen superior y el botón, respectivamente.

- layout_constraintLeft_toLeftOf y layout_constraintTop_toTopOf: A través del valor "parent" se consigue que el botón quede ubicado en el lugar donde aparece dentro del diseño. Deben añadirse si no aparecen.



EJEMPLO PRÁCTICO

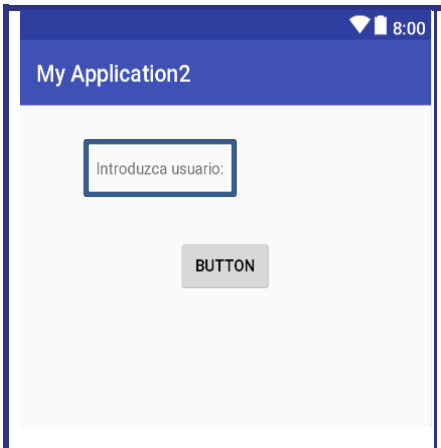
Se propone crear un botón con texto "Salir" con la ubicación 200dp sobre el margen izquierdo y 300dp sobre el margen superior. Mantener las demás propiedades con sus valores correspondientes. ¿Qué ocurriría con el botón si se eliminan las propiedades layout_constraintLeft_toLeftOf y layout_constraintTop_toTopOf con sus respectivos valores "parent"?

PARA SABER MÁS

Los layouts en Android Studio son muchos y muy variados. La importancia de definir layouts es crucial para el desarrollo de la interfaz de usuario y la ubicación de los diferentes componentes con respecto a los márgenes y puntos clave.

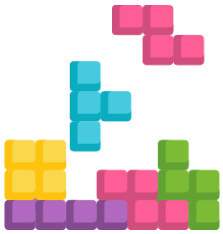
5.2.2 La clase TextView

Esta clase hace referencia al componente etiqueta. En el diseño de la interfaz de usuario de ejemplo, se añade el texto "Introduzca usuario". Tras ello, se analizan las propiedades más utilizadas. Al crear un TextView en Android Studio, aparece lo que se muestra en la imagen:



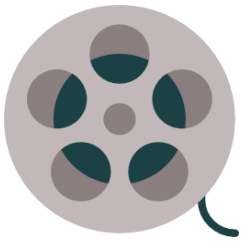

```
<TextView
 android:id="@+id/textView2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="72dp"
 android:layout_marginTop="44dp"
 android:text="Introduzca usuario:"

app:layout_constraintLeft_toLeftOf="paren
t"

app:layout_constraintTop_toTopOf="parent"
/>
```

El componente TextView queda definido por las mismas propiedades que el botón (id para identificarse en el código, text para el texto mostrado y los layouts correspondientes).



EJEMPLO PRÁCTICO

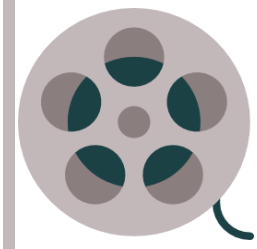
Se propone crear una etiqueta con texto "Nombre y apellidos" con la ubicación 100dp sobre el margen izquierdo y 100dp sobre el margen superior. Mantener las demás propiedades con sus valores correspondientes.



VIDEO DE INTERÉS

En el siguiente vídeo podrás visualizar la clase TextView:

https://www.youtube.com/watch?v=PPgcyIVE9q0



VIDEO DE INTERÉS

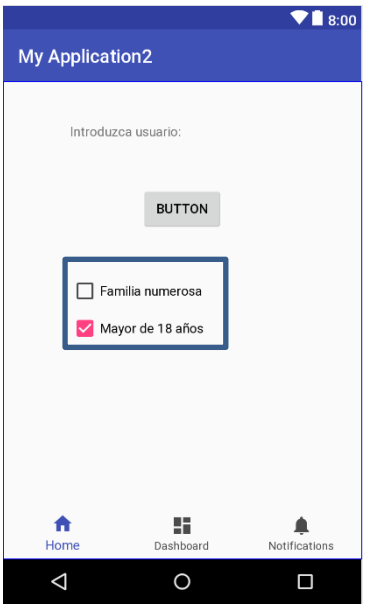
En el siguiente vídeo podrás encontrar la clase PlaintTextView:

https://www.youtube.com/watch?v=RbndSGh73-4

5.2.3 La clase CheckBox

Esta clase hace referencia al componente casilla de verificación. En el diseño de la interfaz de usuario de ejemplo, se añaden los CheckBox "Familia numerosa" y "Mayor de 18 años", dejando seleccionado este último. Tras ello, se analizan las propiedades más utilizadas.

Al crear dos CheckBox en Android Studio, aparece lo que se muestra en la imagen:



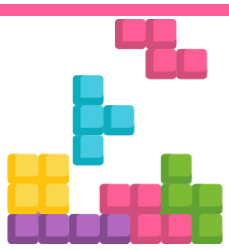

```
<CheckBox
 android:id="@+id/checkBox1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="72dp"
 android:layout_marginTop="208dp"
 android:text="Familia numerosa"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent"
/>

<CheckBox
 android:id="@+id/checkBox2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="72dp"
 android:layout_marginTop="248dp"
 android:checked="true"
 android:text="Mayor de 18 años"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent"
/>
```

Los componentes CheckBox quedan definidos por las mismas propiedades (id para identificarse en el código, text para el texto mostrado y los layouts correspondientes). Los ids son únicos, por tanto, cada CheckBox tendrá uno diferente. Además, aparece una nueva propiedad llamada checked con valor true, es decir, ese CheckBox se encuentra seleccionado ☑

ENLACE DE INTERÉS

A continuación, encontrarás un enlace sobre la usabilidad en las interfaces de usuario, más concretamente para el uso de CheckBox:

http://www.javiergosende.com/usabilidad-formularios-web/25/11/2014/4652



EJEMPLO PRÁCTICO

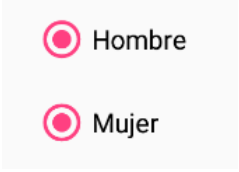
Se propone crear una casilla de verificación con texto "Acepto los términos", chequeada por defecto, con la ubicación izquierda y superior deseada. Mantener las demás propiedades con sus valores correspondientes

5.2.4 Las clases RadioButton y RadioGroup

La clase RadioButton hace referencia a los botones de opción. En el diseño de la interfaz de usuario de ejemplo, se añaden los RadioButton "Hombre" y "Mujer". Tras ello, se analizan las propiedades más utilizadas. Al crear dos RadioButton en Android Studio, aparece lo siguiente:

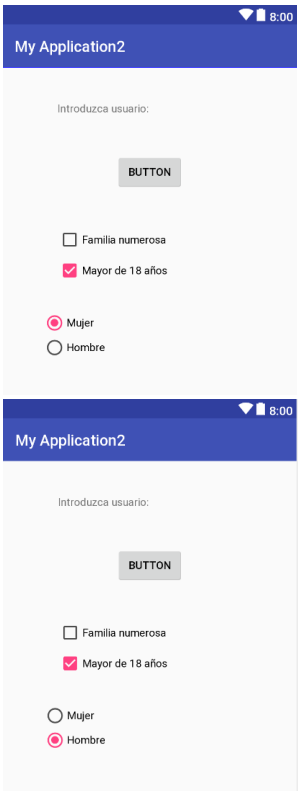

```
<RadioButton
 android:id="@+id/radioButton1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="48dp"
 android:layout_marginTop="372dp"
 android:text="Mujer"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent" />

<RadioButton
 android:id="@+id/radioButton2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="48dp"
 android:layout_marginTop="328dp"
 android:text="Hombre"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent" />
```

Los componentes RadioButton quedan definidos por las mismas propiedades (id para identificarse en el código, text para el texto mostrado y los layouts correspondientes). Los ids son únicos, por tanto, cada RadioButton tendrá uno diferente. Si se hubiese decidido activar una opción por defecto, aparecería nuevamente la propiedad llamada checked con valor true. En este momento, se han creado dos componentes RadioButton. Si se ejecuta esta aplicación, se puede observar cómo ambos RadioButtons pueden activarse.



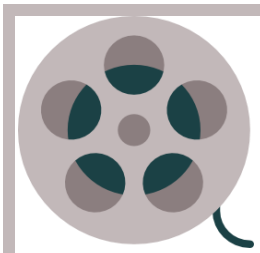
Por definición del componente RadioButton, esto no se puede realizar, ya que sólo permite seleccionar una opción sobre múltiples opciones (en este caso 2 opciones). Para ello se utiliza el componente llamado RadioGroup.

La clase RadioGroup hace referencia a poder agrupar los RadioButtons que comparten opciones (por ejemplo: Hombre/Mujer, Si/No, etc.) Por tanto, van a servir de contenedor de RadioButtons. En el diseño de la interfaz de usuario de ejemplo, se añade un RadioGroup para añadir en él los RadioButton "Hombre" y "Mujer". Una vez se ha creado el RadioGroup se debe introducir en él el código de los radioButtons anteriormente creados:




```
<RadioGroup

 android:id="@+id/radioGroup"
 android:layout_width="90dp"
 android:layout_height="65dp"
 android:layout_marginLeft="52dp"
 android:layout_marginTop="316dp"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent">

 <RadioButton
 android:id="@+id/radioButton1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Mujer"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent" />

 <RadioButton
 android:id="@+id/radioButton2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Hombre"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent" />

</RadioGroup>
```

Puede observarse cómo los RadioButtons no incluyen las propiedades layout_marginLeft y layout_marginTop, puesto que vendrán definidas por el contenedor RadioGroup.



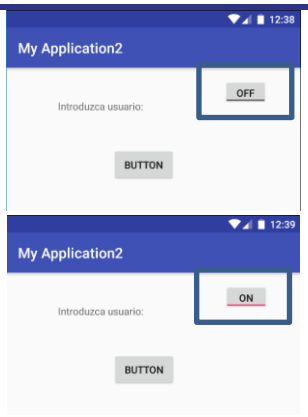
VIDEO DE INTERÉS

En el siguiente enlace podrás visualizar un detalle de la clase RadioGroup y RadioButton:

https://www.youtube.com/watch?v=UtIqbVXt9r4

5.2.5 La clase ToggleButton

Esta clase hace referencia al componente switch (on/off). En el diseño de la interfaz de usuario de ejemplo se añade este componente. Tras ello, se analizan las propiedades más utilizadas. Al crear un ToggleButton en Android Studio, aparece lo siguiente:




```
<ToggleButton
 android:id="@+id/toggleButton2"
 android:layout_width="61dp"
 android:layout_height="36dp"
 android:layout_marginLeft="304dp"
 android:layout_marginTop="16dp"
 android:text="ToggleButton"
 android:checked="false"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent" />
```

El componente ToggleButton queda definido por las mismas propiedades ya conocidas de los anteriores componentes. La propiedad que define el estado del ToggleButton (on/off) es checked (true/false), respectivamente.

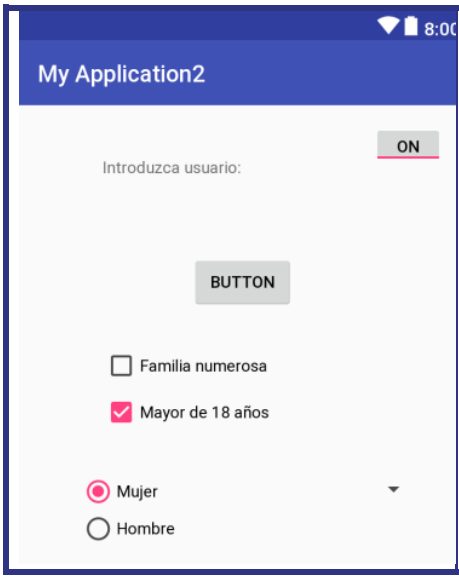
ENLACE DE INTERÉS

En el siguiente enlace encontrarás más información y ejemplos sobre todos los componentes e interfaces como ToggleButton:

https://developer.android.com/guide/topics/ui/controls/togglebutton?hl=es

5.2.6 La clase Spinner

Esta clase hace referencia al componente combo de selección. En el diseño de la interfaz de usuario de ejemplo se añade este componente (en principio, vacío). Tras ello, se analizan las propiedades más utilizadas. Al crear un Spinner en Android Studio, aparece lo siguiente:




```
<Spinner
 android:id="@+id/spinner"
 android:layout_width="166dp"
 android:layout_height="28dp"
 android:layout_marginLeft="180dp"
 android:layout_marginTop="316dp"

app:layout_constraintLeft_toLeftOf="pare
nt"

app:layout_constraintTop_toTopOf="parent
" />
```

El componente Spinner queda definido por las mismas propiedades ya conocidas de los anteriores componentes. Atendiendo a la definición de este componente, se hace necesario añadir elementos dentro de él, ya que en este momento se encuentra totalmente vacío. Para ello, se propone añadir en él una lista de ciudades. Por ejemplo, las siguientes 5: Madrid, Barcelona, Valencia, Sevilla y Bilbao.

Antes de realizarlo, es importante conocer otro fichero de Android Studio que hasta ahora no se había mencionado, pero es sumamente importante para el desarrollo de aplicaciones: el fichero MainActivity.java. Este fichero incluye la parte Java de la aplicación. Atendiendo a su contenido, existe un método que se ejecuta nada más comenzar la actividad, el método onCreate(). Por tanto, es ahí donde se realizará la carga de la información que tendrá el Spinner a través de las siguientes líneas:

```
Spinner combo = (Spinner) findViewById(R.id.spinner);
```

Con esta primera línea se crea un objeto combo que hace referencia al Spinner creado anteriormente y que se encuentra visible en la interfaz.

```
String [] ciudades = {"Madrid", "Barcelona", "Valencia", "Sevilla",
"Bilbao"};
```

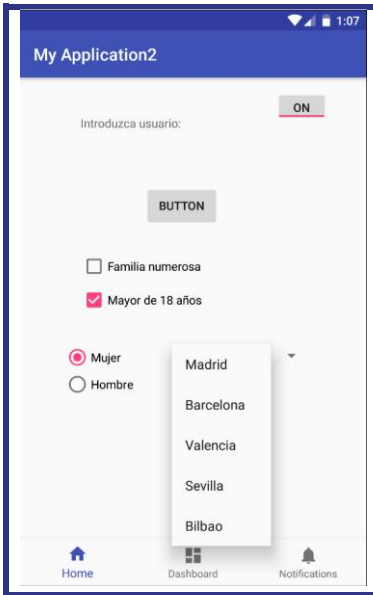
Con esta segunda línea se crea un array (conjunto de elementos) llamado ciudades e incluye las ciudades que van a conformar el contenido del Spinner.

```
ArrayAdapter<String> adapter = new ArrayAdapter<String> (this,
android.R.layout.simple_list_item_1,ciudades);
```

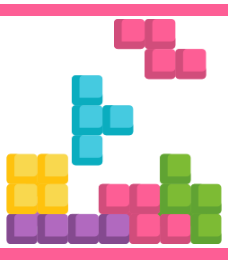
Con esta tercera línea se crea un objeto llamado ArrayAdapter que permite establecer una relación entre las ciudades y el tipo de función que van a tomar (simple list ítem).

```
combo.setAdapter(adapter);
```

Esta cuarta y última línea permite asociar el adapter al combo ya creado anteriormente, para que las ciudades formen parte de su contenido. El resultado tras realizar esta implementación en el fichero MainActivity.java es el siguiente:



El elemento Spinner no se ve alterado en ninguna parte de su definición. Por defecto, aparece el primero de los elementos, en este caso, Madrid, pero sí que es posible modificar esto a través del fichero Java, de esta forma: `combo.setSelection(1);` Para Barcelona, ya que por posición Madrid sería la 0, Barcelona la 1, Valencia la 2, Sevilla la 3 y Bilbao la 4.



EJEMPLO PRÁCTICO

Se propone crear un Spinner en cualquier lugar de la interfaz, siempre que respete la ubicación de los demás componentes añadidos.

5.2.7 La clase EditText

Esta clase hace referencia al componente cuadro de texto (PlainText). En el diseño de la interfaz de usuario de ejemplo se añade este componente. Tras ello, se analizan las propiedades más utilizadas. Al crear un PlainText en Android Studio, aparece lo siguiente:

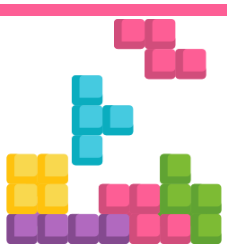

```
<EditText
 android:id="@+id/editText2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginLeft="72dp"
 android:layout_marginTop="60dp"
 android:ems="10"
 android:inputType="textPersonName"
 android:text="Name"

app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent"
/>
```

El componente EditText (PlainText) queda definido, fundamentalmente, por las mismas propiedades ya conocidas de los anteriores componentes.

COMPRUEBA LO QUE SABES

¿Cómo podríamos conocer el número de elementos que contiene un Spinner sin acceder a su contenido a través de la interfaz?

Coméntalo en el foro de la unidad.



EJEMPLO PRÁCTICO

Se propone crear un EditText (PlainText) en cualquier lugar de la interfaz, respetando la ubicación de los demás componentes añadidos.

6. CONTEXTO GRÁFICO. IMÁGENES

En el apartado anterior se muestra una serie de componentes gráficos habituales que forman parte de las interfaces de usuario en una gran cantidad de aplicaciones implementadas para dispositivos móviles. En esta sección se detallan 2 componentes gráficos relacionados con las imágenes: ImageView e ImageButton.

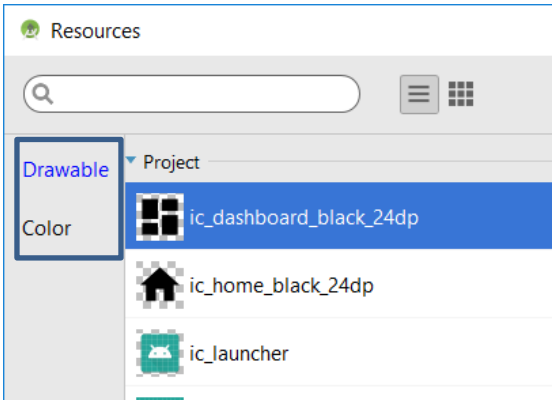
ENLACE DE INTERÉS

A continuación, te proponemos un enlace interesante sobre el uso de los componentes ImageView y ImageButton:

https://www.journaldev.com/9474/android-imageview-imagebutton-example

6.1 El componente ImageView

Siguiendo la idea de componente-clase del apartado anterior, al añadir un componente ImageView, se hace referencia a la propia clase ImageView. Al añadir una imagen en una interfaz de Android Studio, se ofrece la posibilidad de seleccionar un tipo de ImageView llamado Drawable o Color:

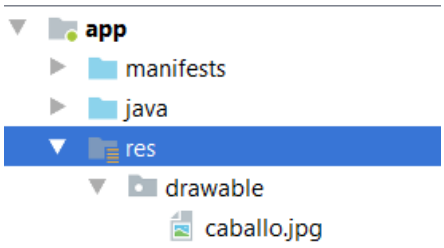


El primero de ellos (Drawable) permitirá seleccionar una imagen a partir de un fichero y el segundo de ellos (Color) rellenar el componente con un color o bien con una mezcla de colores.

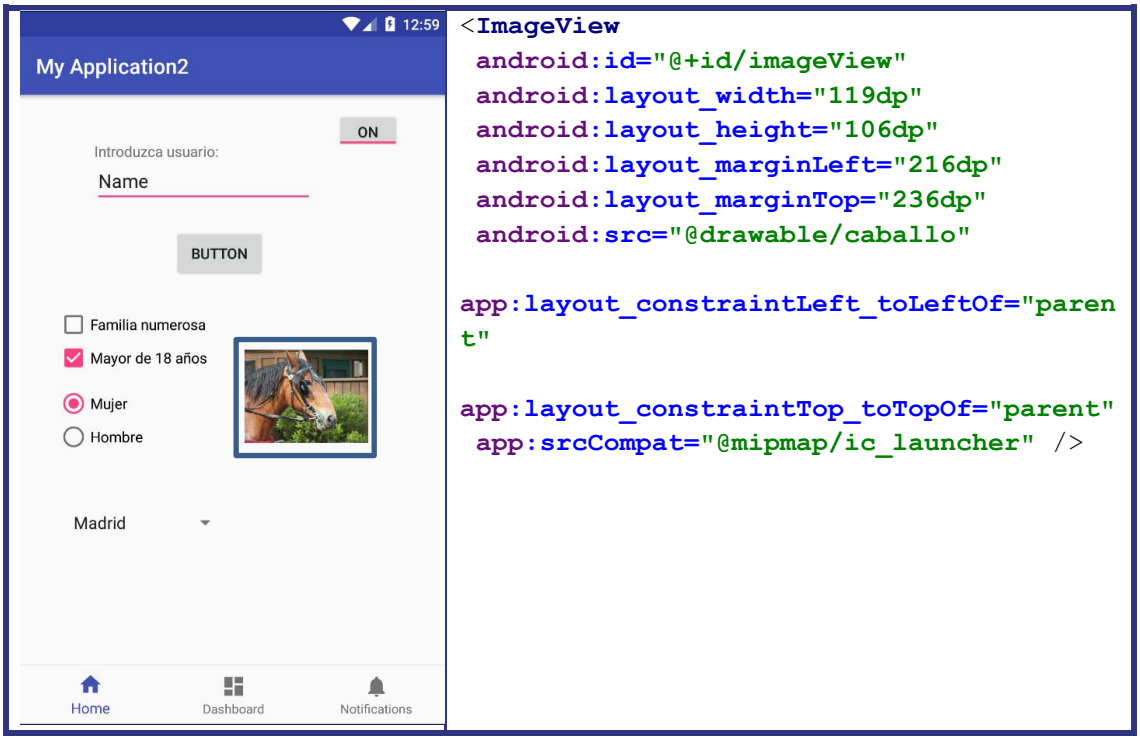
En este sentido, se selecciona una opción cualquiera dentro de Drawable y se añade este componente al diseño de la interfaz de usuario. Para poder asociar una imagen al componente ImageView, es recomendable disponer del fichero

de la imagen y ubicarlo dentro del propio proyecto, concretamente dentro de la carpeta res/drawable.

La siguiente imagen muestra un ejemplo de este proceso. Se ha añadido la imagen caballo.jpg a res/drawable y esta imagen será mostrada en la interfaz de usuario a través del componente ImageView:



Para asociar esta imagen al componente ImageView, se añade la siguiente línea a la definición de su componente: android:src="@drawable/caballo". Al ejecutarse la aplicación, puede observarse el resultado esperado:



El componente ImageView queda definido por las mismas propiedades ya conocidas de los anteriores componentes.

EJEMPLO PRÁCTICO

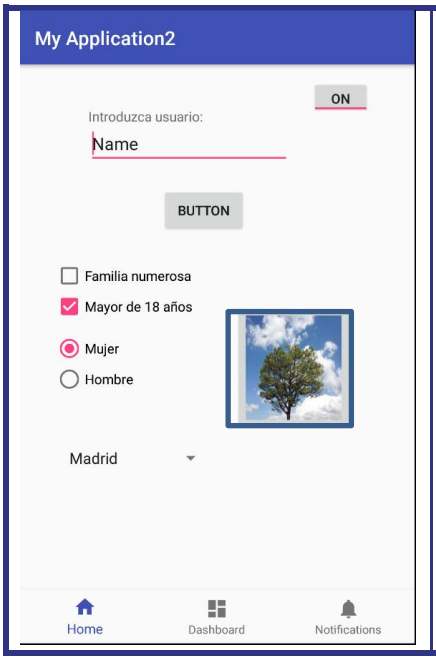
Investigar y probar las diferentes elecciones propuestas por Android Studio tanto para la opción Drawable como para Color.

6.2 El componente ImageButton

Se trata de un componente similar al ImageView, con la diferencia de que la imagen puede ser pulsada como si de un botón se tratase. Al añadir este componente a la interfaz de usuario, Android Studio permite seleccionar el tipo de ImageButton (Drawable para asociar una imagen o Color para rellenar el componente con un color o mezcla de colores). Se procede a realizar los mismos pasos que se han aplicado en el componente ImageView, optando por una nueva imagen, en este caso arbol.jpg, que de igual manera se ha añadido en la ubicación res/drawable del proyecto.

Para asociar esta imagen al componente ImageButton, se añade la misma línea que en el componente anterior en su definición de componente: android:src="@drawable/arbol" y, además, android:scaleType="fitXY" para ajustarse al tamaño del botón.

Al ejecutarse la aplicación, puede observarse el resultado esperado:




```
<ImageButton
 android:id="@+id/imageButton"
 android:layout_width="123dp"
 android:layout_height="128dp"
 android:layout_marginLeft="224dp"
 android:layout_marginTop="252dp"
 android:scaleType="fitXY"
 android:src="@drawable/arbol"

app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintTop_toTopOf="parent"
 app:srcCompat="@mipmap/ic_launcher" />
```

El componente ImageButton queda definido por las mismas propiedades ya conocidas de los anteriores componentes. A diferencia del ImageView, este componente realmente se comporta como si de un botón se tratase.

COMPRUEBA LO QUE SABES

¿A partir de un componente Button se puede añadir una imagen de fondo simulando el comportamiento del componente ImageButton?

Coméntalo en el foro de la unidad.

7. EVENTOS

Un evento puede definirse como una acción sobre un componente concreto. Son parte fundamental en el desarrollo de aplicaciones tanto para dispositivos móviles como para cualquier otro tipo de sistema informático. En este sentido, los eventos son los responsables de aportar dinamismo a las aplicaciones gracias a los comportamientos asociados a cada componente que conforma las interfaces de usuario. Un ejemplo de evento puede ser el hecho de pulsar con el ratón sobre un botón o seleccionar una opción de un radioButton o de un checkBox para desencadenar una acción concreta. Otro ejemplo de evento puede ser al pulsar una tecla del teclado en un EditText desencadenar otra cierta acción. Cada componente proporcionado por Android Studio para el desarrollo de aplicaciones es susceptible de poder generar eventos. Los eventos pueden ser de diferente índole, en función de los requisitos que se pretendan desarrollar dentro de la interfaz de usuario y, más concretamente, dentro del fichero Java, lugar indicado para realizar la implementación de los eventos.

Como se ha comentado anteriormente, un componente puede implementar eventos de diferente índole. Por ejemplo, se puede implementar un evento al pulsar con el ratón sobre una imagen, otro evento al pasar con el ratón por encima de la imagen (sin necesidad de hacer clic) y otro evento al salir con el ratón de la imagen. En esta sección se van a describir los eventos que se consideran más interesantes a la hora de ser utilizados. La siguiente tabla muestra un resumen de los componentes y sus correspondientes eventos:

Componente	Evento y método	Acción
Button	Evento: setOnClickListener Método: onClick	Hacer clic (pulsar) con el ratón sobre él.
EditText (PlainText)	Evento: addTextChangedListener Métodos: beforeTextChanged onTextChanged y afterTextChanged	Pulsar una tecla y escribir en él. Los métodos responden al estado anterior al escribir, al momento de escribir y al finalizar la escritura.
CheckBox	Evento: setOnClickListener Método: onClick	Hacer clic sobre él para chequearlo o deschequearlo.
RadioButton	Evento: setOnClickListener Método: onClick	Hacer clic sobre él para seleccionarlo.
RadioGroup	Evento: setOnCheckedChangeListener Método: onCheckedChanged	Seleccionar cualquier radioButton de su grupo
ToggleButton	Evento: setOnClickListener Método: onClick	Hacer clic sobre él para encenderlo o apagarlo.
Spinner	Evento: setOnItemSelectedListener Métodos: onItemSelected y onNothingSelected	Seleccionar un elemento del combo.
ImageButton	Evento: setOnClickListener Método: onClick	Hacer clic (pulsar) con el ratón sobre la imagen,

Los diferentes eventos se implementarán a partir de la interfaz de usuario del ejemplo anteriormente diseñada. Antes de comenzar a implementar eventos, es conveniente acceder al fichero Java y preparar en él cada componente de la interfaz de usuario para ser utilizado posteriormente. Esta preparación consiste en crear un objeto mediante la clase de cada componente y asociarlo a cada uno de ellos. En el caso de la interfaz de usuario del ejemplo, se va a crear un objeto para el botón (clase Button), el cuadro de texto (EditText), los CheckBox (clase CheckBox), los RadioButton (clase RadioButton), el ToggleButton (clase ToggleButton), el combo (clase Spinner) y el botón con imagen (clase ImageButton). La asociación de cada componente con su objeto correspondiente ya se ha realizado en la sección que detalla la clase Spinner (unas páginas atrás) y su procedimiento será exactamente el mismo para cada componente.

El primer paso consiste en la creación de un objeto por cada componente que interviene en la interfaz:

```
public class MainActivity extends AppCompatActivity {
 private Button boton;
 private EditText cuadroTexto;
 private CheckBox familia;
 private CheckBox mayor18;
 private RadioButton mujer;
 private RadioButton hombre;
 private ToggleButton onoff;
 private ImageView imagen;
 private ImageButton imagenBoton;
 private Spinner combo;
      …
```

El segundo paso consiste en asociar a cada objeto el componente que le corresponde. Esto se realiza dentro del método onCreate de esta forma:

```
boton = (Button) findViewById(R.id.button);
cuadroTexto = (EditText) findViewById(R.id.editText2);
familia = (CheckBox) findViewById(R.id.checkBox1);
mayor18 = (CheckBox) findViewById(R.id.checkBox2);;
mujer = (RadioButton) findViewById(R.id.radioButton1);
hombre = (RadioButton) findViewById(R.id.radioButton2);;
onoff = (ToggleButton) findViewById(R.id.toggleButton2);
imagenBoton = (ImageButton) findViewById(R.id.imageButton);;
combo = (Spinner) findViewById(R.id.spinner);
```

Para realizar el segundo paso hay que revisar el nombre del identificador (id) asociado a cada componente. Las siguientes líneas (copiadas del fichero xml activity_main) recuerdan cada id:

```
android:id="@+id/button"
android:id="@+id/editText2"
android:id="@+id/checkBox1"
android:id="@+id/checkBox2"
android:id="@+id/radioButton1"
android:id="@+id/radioButton2"
android:id="@+id/toggleButton2"
android:id="@+id/imageButton"
android:id="@+id/spinner"
```

Volviendo al segundo paso, puede comprobarse cómo cada objeto es creado de la siguiente forma: **objeto = (clase) findViewById(R.id.nombreId)**. A partir de este instante ya es posible hacer uso de cada componente a través de cada objeto (botón, cuadroTexto, familia, mayor18, mujer, hombre, onoff, imagenBoton y combo).

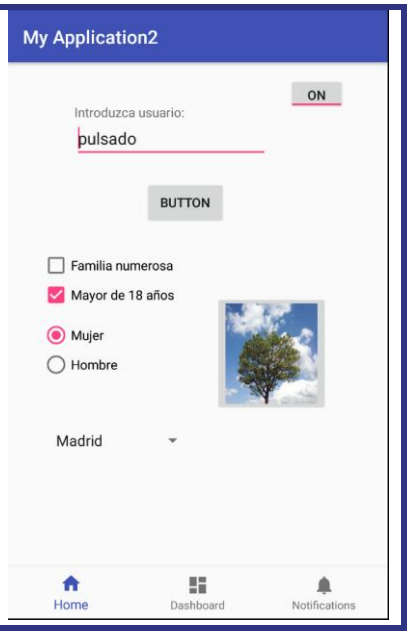
7.1 Evento en componente Button

El evento más interesante utilizable sobre un botón es pulsar sobre él. Este evento es conocido como setOnClickListener y el método que lo ejecuta es onClick.

Antes de implementar el evento se hace necesario conocer qué acción se va a llevar a cabo sobre el componente botón. Como ejemplo, se propone que, al hacer clic sobre el botón, se muestre en el cuadro de mensaje la palabra "pulsado". Para desarrollar la acción se hace uso del evento y se implementa el método onClick de la siguiente forma:

```
boton.setOnClickListener(new
View.OnClickListener()
 {
 @Override
 public void onClick(View v) {
 cuadroTexto.setText("pulsado");
 }
});
```

Como puede comprobarse, el objeto botón invoca al evento setOnClickListener el cual implementa el método onClick. Es dentro de ese método donde se implementa la acción del ejemplo que, en este caso, cambia el texto del cuadro de mensaje escribiendo la palabra "pulsado" al hacer clic en el botón.



A la hora de desarrollar los eventos es importante tener en cuenta dos aspectos: el primero, conocer la acción o acciones que se desea implementar y, el segundo, saber programar el código para generar dicha acción.

VIDEO DE INTERÉS

En el siguiente enlace encontrarás un vídeo en el que podrás visualizar un detalle de la clase y vista Button:

https://www.youtube.com/watch?v=fsPFRwQo-zI

7.2 Evento en componente EditText

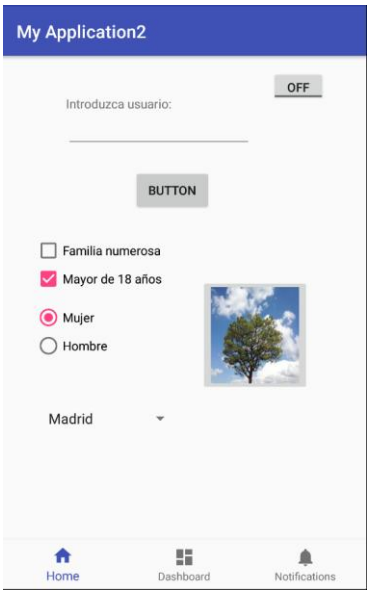
El evento más interesante utilizable sobre un cuadro de texto es escribir sobre él (pulsar tecla). Este evento es conocido como addTextChangedListener y los métodos que lo ejecutan son beforeTextChanged (antes de escribir en el cuadro de texto), onTextChanged (al momento de escribir en el cuadro de texto) y afterTextChanged (una vez se ha terminado de escribir en el cuadro de texto).

Antes de implementar el evento se hace necesario conocer qué acción o acciones se van a llevar a cabo sobre el componente cuadro de texto. Como ejemplo, se propone que, antes de pulsar tecla alguna, el componente ToggleButton se encuentre en OFF, al escribir cualquier texto sobre el cuadro de texto, el componente ToggleButton se encienda (ON) y al finalizar la escritura, el texto del botón cambie a "Pulsar".

Para desarrollar las acciones se hace uso del evento y se implementan los métodos beforeTextChanged, onTextChanged y afterTextChanged de la siguiente forma:

```
cuadroTexto.addTextChangedListener(new TextWatcher() {
 @Override
 public void beforeTextChanged(CharSequence s, int start, int count,
 int after) {
 onoff.setChecked(false);
 }

 @Override
 public void onTextChanged(CharSequence s, int start, int before,
 int count) {
 onoff.setChecked(true);
 }

 @Override
 public void afterTextChanged(Editable s) {
 boton.setText("Pulsar");
 }
});
```

Como puede comprobarse, el objeto cuadroTexto invoca al evento llamado addTextChangedListener, el cual implementa los 3 métodos descritos para realizar las acciones antes de escribir, al momento de escribir y posterior a escribir.

		En el momento inicial, la interfaz carga los componentes y al no escribir nada en el cuadro de texto, el estado del ToggleButton se encuentra en OFF. Una vez se pulsa una tecla, se escribe el texto y el ToggleButton pasa a estado ON. Al terminar de escribir, el botón muestra el texto "Pulsar".

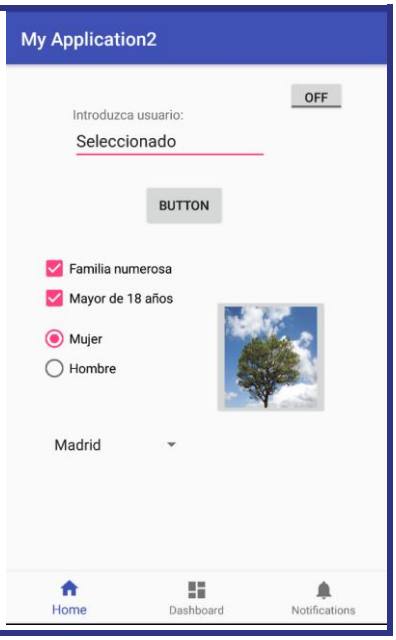
7.3 Evento en componente CheckBox

El evento más interesante utilizable sobre un checkBox es pulsar sobre él bien para seleccionarlo o bien para deseleccionarlo. Este evento es conocido como setOnClickListener y el método que lo ejecuta es onClick.

Antes de implementar el evento se hace necesario conocer qué acción se va a llevar a cabo sobre el componente checkBox. Como ejemplo, se propone que, al hacer clic sobre el checkBox y este pasar a estar seleccionado (es decir, antes de hacer clic se encuentra deseleccionado), se muestre en el cuadro de mensaje la palabra "seleccionado". Si al hacer clic sobre el checkBox este pasa a estar deseleccionado (es decir, antes de hacer clic se encuentra seleccionado), se mostrará el mensaje "no seleccionado". Para desarrollar la acción se hace uso del evento y se implementa el método onClick de la siguiente forma:

```
familia.setOnClickListener(new
View.OnClickListener()
 {
 @Override
 public void onClick(View v) {
 if (familia.isChecked())
 cuadroTexto.setText("Seleccionado");
 else cuadroTexto.setText("No
seleccionado");
 }
});
```

En este primer caso, el checkBox no estaba seleccionado y, al pulsar sobre él, se selecciona, mostrando en el cuadro de texto la palabra "Seleccionado".




```
familia.setOnClickListener(new
View.OnClickListener()
 {
 @Override
 public void onClick(View v) {
 if (familia.isChecked())
 cuadroTexto.setText("Seleccionado");
 else cuadroTexto.setText("No
seleccionado");
 }
});
```

En este segundo caso, el checkBox estaba seleccionado y, al pulsar sobre él, se deselecciona, mostrando en el cuadro de texto la palabra "No seleccionado".

7.4 Evento en componente RadioButton

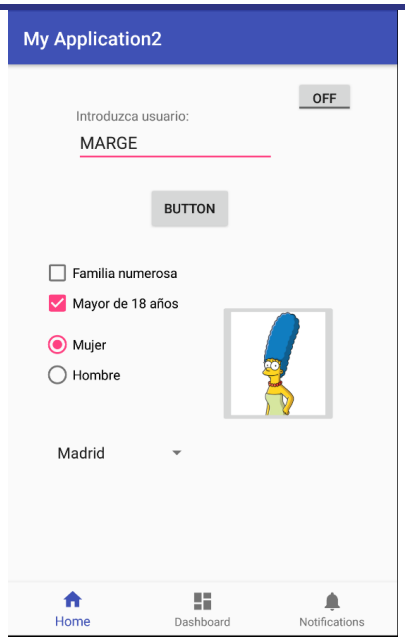
El evento más interesante utilizable sobre un radioButton es pulsar sobre él, cuya acción seleccionaría el radioButton pulsado y deseleccionaría cualquiera que estuviera seleccionado previamente. Este evento es conocido como setOnClickListener y el método que lo ejecuta es onClick.

Antes de implementar el evento se hace necesario conocer qué acción se va a llevar a cabo sobre el componente radioButton. Como ejemplo, se propone que, al hacer clic sobre el radioButton "Mujer", el componente ImageButton muestre una imagen de una mujer y el cuadro de texto muestre su nombre.

Por su parte, al hacer clic sobre el radioButton "Hombre", muestre una imagen de un hombre y el cuadro de texto muestre su nombre. Para desarrollar la acción se hace uso del evento y se implementa el método onClick de la siguiente forma:

```
mujer.setOnClickListener(new
View.OnClickListener() {
 @Override
 public void onClick(View v) {
 cuadroTexto.setText("MARGE");

imagenBoton.setImageResource(R.drawable.marge);
 }
});
```

En este primer caso, el radioButton "Mujer", al ser seleccionado, muestra el resultado tanto en el cuadro de texto como en la imagen del componente ImageButton. Nótese cómo es necesario almacenar el fichero de imagen (Marge) en el directorio res/drawable.




```
hombre.setOnClickListener(new
View.OnClickListener() {
 @Override
 public void onClick(View v) {
 cuadroTexto.setText("HOMER");

imagenBoton.setImageResource(R.drawable.homer);
 }
});
```

En este segundo caso, el radioButton "Hombre", al ser seleccionado, muestra el resultado tanto en el cuadro de texto como en la imagen del componente ImageButton. Nótese cómo es necesario almacenar el fichero de imagen (Homer) en el directorio res/drawable.

Para el caso de eventos en radioButtons, también es importante conocer el evento setOnCheckedChangeListener para el componente ButtonGroup junto con su método onCheckedChanged, el cual realiza una acción similar:

```
grupo.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener()
{
 @Override
 public void onCheckedChanged(RadioGroup group, int checkedId) {
 // checkedId is the RadioButton selected
 switch(checkedId)
 {
 case R.id.radioButton1:
 mujer.isChecked();
 break;

 case R.id.radioButton2:
 hombre.isChecked();
 break;
 }
 }
});
```

7.5 Evento en componente ToggleButton

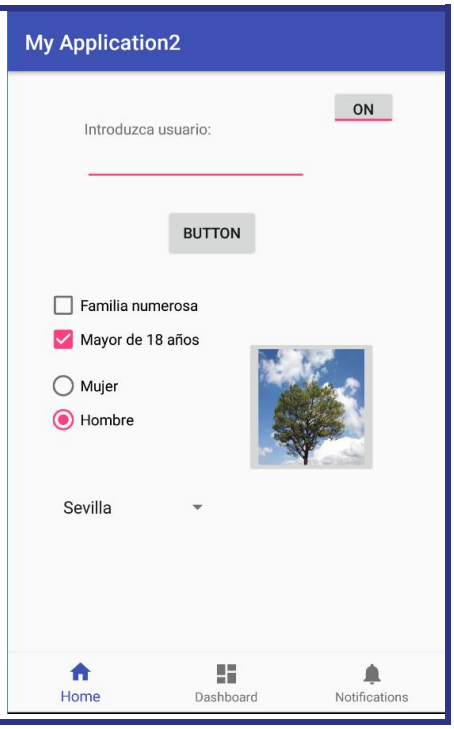
El evento más interesante utilizable sobre un toggleButton es pulsar sobre él bien para activarlo (ON) o bien para desactivarlo (OFF). Este evento es conocido como setOnClickListener y el método que lo ejecuta es onClick.

Antes de implementar el evento se hace necesario conocer qué acción se va a llevar a cabo sobre el componente toggleButton. Se propone que, al hacer clic sobre él y este pasar a estar activo (ON), se seleccione en el Spinner la ciudad "Sevilla" (posición cuarta, por tanto, índice 3). Si al hacer clic sobre el toggleButton, este pasa a estar desactivo (OFF), se seleccionará en el Spinner la ciudad "Bilbao" (posición quinta, por tanto, índice 4).

Para desarrollar la acción se hace uso del evento y se implementa el método onClick de la siguiente forma:

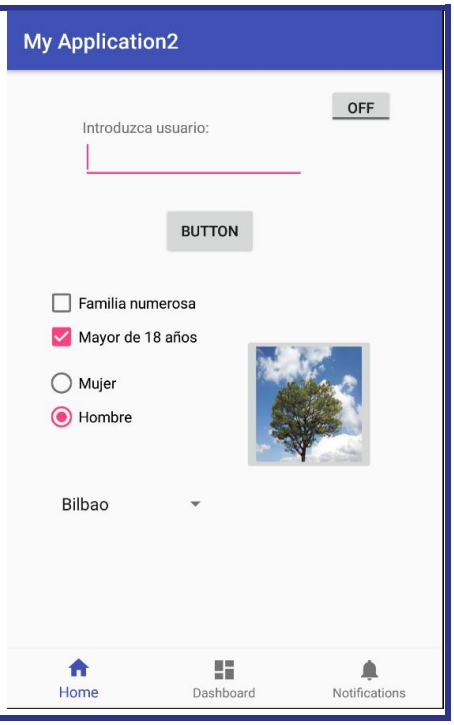
```
onoff.setOnClickListener(new
View.OnClickListener() {
 @Override
 public void onClick(View v) {
 if (onoff.isChecked()) { //En ON
 combo.setSelection(3);
 }
 else //En OFF
 combo.setSelection(4);
 }
});
```

En este primer caso, el ToggleButton se ha pulsado y se encuentra en ON, por tanto, activo. Esto conlleva seleccionar en el componente Spinner la ciudad Sevilla.




```
onoff.setOnClickListener(new
View.OnClickListener() {
 @Override
 public void onClick(View v) {
 if (onoff.isChecked()) { //En ON
 combo.setSelection(3);
 }
 else //En OFF
 combo.setSelection(4);
 }
});
```

En este segundo caso, el ToggleButton se ha pulsado y se encuentra en OFF, por tanto, desactivo. Esto conlleva seleccionar en el componente Spinner la ciudad Bilbao.



ENLACE DE INTERÉS

A continuación, te proponemos un enlace interesante sobre el uso del componente ToggleButton:

http://www.sgoliver.net/blog/interfaz-de-usuario-en-android-controles-basicos-i/

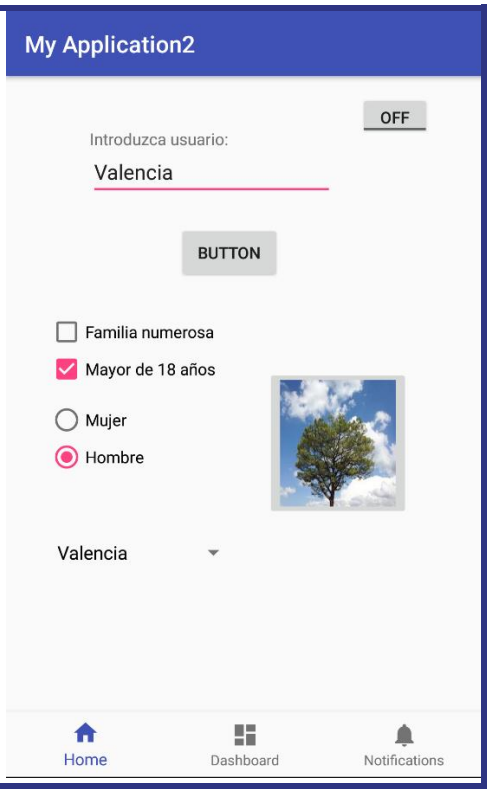
7.6 Evento en componente Spinner

El evento más interesante utilizable sobre un combo es pulsar sobre él y elegir un valor de entre sus diferentes opciones. Este evento es conocido como setOnItemSelectedListener y los métodos que lo ejecutan son dos: uno para la selección de un ítem (onItemSelected) y otro para cuando la selección es nula (onNothingSelected).

Antes de implementar el evento se hace necesario conocer qué acción se va a llevar a cabo sobre el componente botón. Como ejemplo, se propone que, al seleccionar un ítem cualquiera, se muestre en el cuadro de mensaje el nombre del ítem (ciudad seleccionada). Si no se selecciona ninguna ciudad, el cuadro de mensaje pasa a estar vacío. Para desarrollar la acción se hace uso del evento y se implementan los métodos anteriormente mencionados de la siguiente forma:

```
combo.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
 @Override
 public void onItemSelected(AdapterView<?> parent, View view, int
 position, long id) {
 cuadroTexto.setText(combo.getSelectedItem().toString());
 }

 @Override
 public void onNothingSelected(AdapterView<?> parent) {
 cuadroTexto.setText("");
 }
});
```

En este ejemplo siempre va a existir un elemento del Spinner seleccionado, con lo cual se ejecutará el método onItemSelected, escribiendo en el cuadro de texto la ciudad seleccionada en el combo (Spinner). En otros IDEs, el componente Spinner es conocido como ComboBox. Su funcionalidad es exactamente la misma que el Spinner.



COMPRUEBA LO QUE SABES

¿Sería posible establecer un Spinner sin ningún elemento seleccionado por defecto? En caso afirmativo, ¿cómo lo implementarías?

Coméntalo en el foro de la unidad.

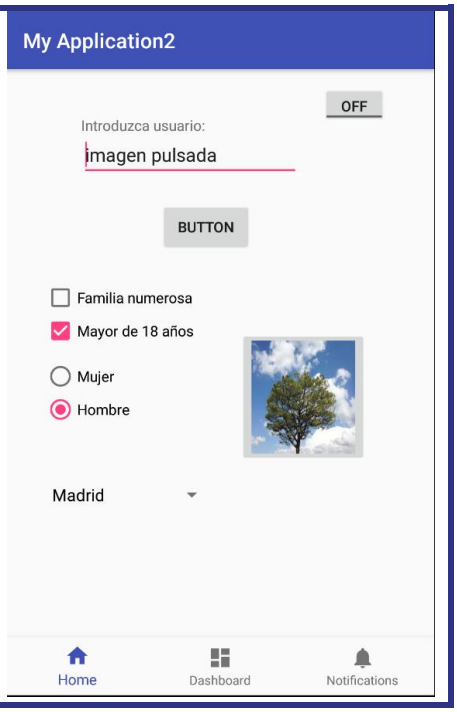
7.7 Evento en componente ImageButton

El evento más interesante utilizable sobre un botón de imagen es pulsar sobre ella. Este evento es conocido como setOnClickListener y el método que lo ejecuta es onClick.

Antes de implementar el evento se hace necesario conocer qué acción se va a llevar a cabo sobre el componente botón de imagen. Como ejemplo, se propone que, al hacer clic sobre el botón, se muestre en el cuadro de mensaje la palabra "imagen pulsada". Para desarrollar la acción se hace uso del evento y se implementa el método onClick de la siguiente forma:

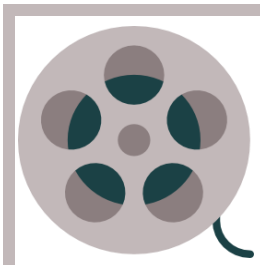
```
imagenBoton.setOnClickListener(new
 View.OnClickListener() {
 @Override
 public void onClick(View v) {
 cuadroTexto.setText("imagen pulsada");
 }
});
```

El objeto botón de imagen invoca a setOnClickListener, el cual implementa el método onClick. Es dentro de ese método donde se implementa la acción del ejemplo que, en este caso, cambia el texto del cuadro de mensaje escribiendo la palabra "imagen pulsada" al pulsar en ella.



8. TÉCNICAS DE ANIMACIÓN Y SONIDO

El entorno de desarrollo Android Studio proporciona funcionalidad multimedia en la implementación de sus aplicaciones. En este sentido, existe una clase llamada MediaPlayer que es la encargada de controlar la reproducción de audio y vídeo. Por tanto, en esta sección, se hará uso de la clase MediaPlayer a través del ejemplo de la aplicación desarrollada en los apartados anteriores.



VIDEO DE INTERÉS

En el vídeo que encontrarás en el siguiente enlace podrás visualizar una implementación de la clase MediaPlayer en Android Studio:

https://www.youtube.com/watch?v=12xhoO7A1v4

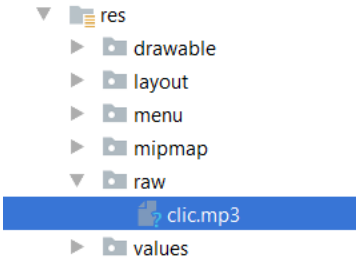
8.1 La clase MediaPlayer

En primer lugar, es necesario declarar un objeto de la clase MediaPlayer, llamado, por ejemplo, mp (hace referencia a las iniciales de la clase).

```
private MediaPlayer mp;
// Dentro del método onCreate…
mp = new MediaPlayer();
```

Posteriormente al objeto mp se le debe asociar el contenido multimedia que se pretenda ejecutar mediante el método create(). Por ejemplo, a través de un recurso (fichero de audio/vídeo que debe estar almacenado en el directorio res/raw), de una ruta a un fichero almacenado en cualquier dispositivo, de una URI para ser ejecutado online... En el ejemplo se va a crear el directorio res/raw y se va a almacenar el fichero clic.mp3 para asociarlo a un botón.

```
mp = MediaPlayer.create(this, R.raw.clic);
```



Este proceso se implementa de la siguiente forma, aprovechando la existencia del botón y a través del método start():

```
boton.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 cuadroTexto.setText("pulsado");
 mp.start();
 }
});
```

Es importante conocer que el objeto mp (MediaPlayer) permite controlar la ejecución a través de 2 métodos más: pause() y stop() para pausar y parar la reproducción, respectivamente.

8.2 Las clases MediaController y VideoView

Por otra parte, existe una clase llamada MediaController que facilita un control más exhaustivo de las reproducciones multimedia en Android Studio. Esta clase ofrece las opciones de reproducción: Play, Pause, Rewind (rebobinado), Fast Forward (avance rápido) y barra de progreso. Es más utilizada para la reproducción de vídeo junto con la clase VideoView.

A continuación, se va a realizar un pequeño desarrollo para la reproducción de vídeo. El primer paso para hacer uso de esta clase es instanciar un objeto

de la clase MediaController, que va a ser llamado mc (por razones obvias) y un objeto de la clase VideoView llamado vv de la siguiente forma:

```
private MediaController mc;
private VideoView vv;
// Dentro del método onCreate…
mc = new MediaController(contexto);
vv = (VideoView)findViewById(R.id.videoView);
```

Previamente se ha tenido que añadir un componente VideoView a la interfaz de usuario, con id VideoView. La siguiente línea permite añadir los controles al VideoView, de la siguiente forma:

```
vv.setMediaController(mc);
```

A continuación, se debe cargar el contenido multimedia, es decir, el vídeo o la animación en el componente VideoView que se encuentra almacenado en el directorio res/raw. En este caso será a través de una URI que se forma de la siguiente forma:

```
String uriPath = "android.resource://" +
getPackageName() + "/" + R.raw.video;
Uri uri = Uri.parse(uriPath);
vv.setVideoURI(uri);
```

9. DESCUBRIMIENTO DE SERVICIOS

Un servicio puede definirse como un programa que se ejecuta en segundo plano, sin interfaz gráfica de usuario. El usuario, al no poder interactuar con él mediante interfaz, se comunica a través de otros programas con una comunicación denominada IPC (*Inter Process Comunication*). Grosso modo, un servicio se utiliza principalmente para:

- Realizar operaciones de entrada y salida de forma transparente al usuario sin que repercuta en la funcionalidad de la interfaz de usuario.
- Cálculos con bastante consumo de recursos.
- Comunicaciones en red.

Un servicio puede ser de tipo vinculado o no vinculado. El primero de ellos se caracteriza por poseer una clara relación con el componente que lo ha creado, a diferencia del segundo, que se ejecuta de manera independiente.

Para la creación de ambos servicios se hace uso de una subclase de la clase Service. En el caso de los vinculados se debe implementar el método llamado onBind() y de los no vinculados el método onStartCommand().

ENLACE DE INTERÉS

A continuación, encontrarás un enlace sobre la creación de servicios en Android:

https://www.develou.com/tutorial-para-crear-un-servicio-en-android/

10. BASES DE DATOS Y ALMACENAMIENTO

Las bases de datos y el almacenamiento de la información son elementos fundamentales en el desarrollo de aplicaciones. Gracias a ellas es posible mantener los datos y poder acceder a ellos para su lectura, modificación, actualización o/y borrado. En el sistema operativo Android existe un gestor de base de datos llamado SQLite que permite almacenar datos de forma estructurada y sencilla. Para poder realizar esta operación, se debe crear un objeto de la clase SQLiteDatabase, invocando a un método llamado openOrCreateDatabase (nombre, modo, cursor) cuya ejecución crea una nueva base de datos si no existe previamente y retorna un objeto para que, a partir de él, se pueda acceder a dicha base de datos.

A partir de la interfaz de usuario desarrollada en las primeras secciones, se va a implementar un ejemplo de uso de este gestor de base de datos. Es importante, llegado este punto, conocer nociones básicas del lenguaje SQL para seguir con garantías el siguiente ejemplo. En primer lugar, se declara una variable de tipo SQLiteDatabase (llamado db) en el fichero Java MainActivity de la siguiente forma:

```
SQLiteDatabase db;
// Dentro del método onCreate…
db = openOrCreateDatabase("Usuarios", Context.MODE_PRIVATE, null);
```

De esta manera se crea, si no existe, una base de datos de usuarios para almacenar la información de los usuarios añadidos a través de la interfaz.

A través del método execSQL se crea una tabla llamada usuarios con 4 campos: nombre (alfanumérico), sexo (alfanumérico), familia (bit) y mayor (bit). Estos campos hacen referencia al nombre del usuario, su sexo, si tiene o no familia numerosa y si es o no mayor de edad. Por ejemplo, una entrada para la tabla usuarios puede ser: Juan, Hombre, 0, 1. Los campos de tipo bit se han asignado para familia y mayor; si es 0 en el campo familia, quiere decir que no tiene familia numerosa y 1, sí. Si es 0 en el campo mayor, quiere decir que no ha alcanzado la mayoría de edad y 1, sí.

```
db.execSQL("CREATE TABLE IF NOT EXISTS usuarios(Nombre VARCHAR, Sexo
VARCHAR, familia BIT, mayor BIT)");
```

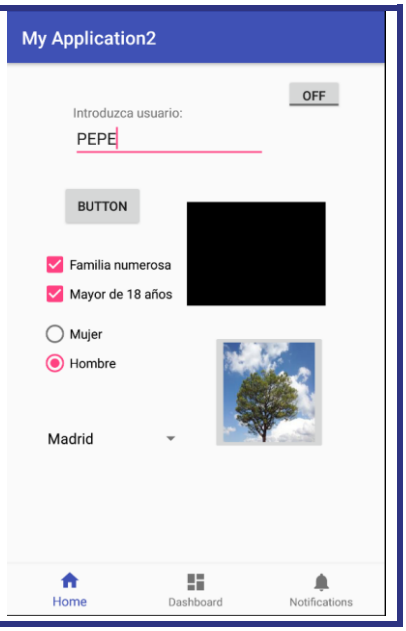
10.1 Insertando datos en la base de datos

El siguiente paso consiste en añadir un evento al botón para poder insertar los datos introducidos. La implementación de esta acción se puede realizar de la siguiente forma:

```
boton.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 String sexo;
 int family, m18;
 if (mujer.isChecked()) sexo=mujer.getText().toString();
 else sexo=hombre.getText().toString();
 if (familia.isChecked()) family=1;
 else family=0;
 if (mayor18.isChecked()) m18=1;
 else m18=0;
 String consulta = "INSERT INTO usuarios VALUES
        ('"+cuadroTexto.getText()+"','"+sexo+"',"+family+","+m18+")";
 Log.d("MyApp",consulta);
 db.execSQL(consulta);

 listar();
 }
}
```

Antes de realizar la consulta, es importante crear variables para cada dato a introducir en la base de datos. En este sentido, se han creado las variables sexo, family y m18 para almacenar en ellas el tipo de sexo (Mujer u Hombre), si tiene o no familia numerosa y si es o no mayor de edad. Los siguientes if-else estudian la condición marcada por el usuario para comprobar si dichos campos están o no chequeados y asociarlos a las variables correspondientes. Una vez realizado, ya se puede construir la sentencia SQL que permite introducir los datos en la base de datos. Para el ejemplo mostrado, la consulta sería la siguiente:

```
cuadroTexto.getText() → PEPE
if (mujer.isChecked())
sexo=mujer.getText().toString();
else   sexo=hombre.getText().toString();   →
HOMBRE
if (familia.isChecked()) family=1; → 1
else family=0;
if (mayor18.isChecked()) m18=1; → 1
else m18=0;
INSERT INTO usuarios VALUES ('PEPE',
'HOMBRE', 1, 1);
```



Por último, y a través de la llamada al método execSQL, se ejecutará la consulta sobre la base de datos usuarios.

10.2 Mostrando datos de la base de datos

A la hora de mostrar los elementos de la base de datos, se ha creado el método listar(), que realiza una consulta SQL con una sentencia SELECT para mostrar todo el contenido de la Base de datos usuarios:

```
private void listar() {
 List<String> lista = new ArrayList<String>();
 Cursor c = db.rawQuery("SELECT * FROM usuarios", null);
 if (c.getCount()==0) {
 lista.add("No hay registros");
 }
 else {
 int i =0;
 while(c.moveToNext()) {
 lista.add(c.getString(0) + "-" + c.getString(1)+ "-" +
                c.getString(2)+ "-" + c.getString(3));
 Log.d("Elemento: ", lista.get(i));
 i++;
 }
 }
 Log.d("Elementos: ", String.valueOf(lista.size()));
 c.close();
 db.close();;
}
```

Este método, básicamente, crea la consulta y posteriormente almacena en una variable de tipo ArrayList (lista dinámica de elementos) toda la salida que

se obtiene. Si no hay elementos que mostrar, añade "No hay registros". Si existen elementos que mostrar va recorriendo uno a uno y los muestra a través del Log de Android Studio. Además, muestra el número de elementos que se han introducido.

```
12-10    03:03:27.842    2577-2577/com.example.josecarlos.myapplication2
D/Elemento:: PEPE-Hombre-1-1
12-10    03:03:27.842    2577-2577/com.example.josecarlos.myapplication2
D/Elementos:: 1
```

Por último, es importante cerrar bien tanto el cursor (variable utilizada para la sentencia SQL) como la propia base de datos cuando ya no estén en uso para liberar todos los recursos abiertos.

10.3 Eliminando datos de la base de datos

Para eliminar datos de la base de datos se utilizará la consulta SQL de tipo DELETE. Esta acción se puede implementar, por ejemplo, sobre el evento de un botón de la siguiente forma:

```
private void borrar(View v) {
 db.execSQL("DELETE FROM usuarios WHERE Nombre =
                '"+cuadroTexto.getText().toString()+"'");
 listar();
}
```

Dependiendo del contenido de la sentencia DELETE, se borrarán unos registros u otros. En este caso se ha optado por borrar aquellos usuarios cuyo nombre coincida con el nombre que esté escrito en el campo Usuario de la interfaz de usuario. Por ejemplo, antes de ejecutar el método borrar existen los siguientes registros:

```
MARGE-Mujer-1-1
PEPE-Hombre-1-1
PEPE-Hombre-1-1
```

Y, posteriormente, se ejecuta el método borrar con PEPE como nombre de usuario. Ya solo quedaría el registro MARGE almacenado en la base de datos.

COMPRUEBA LO QUE SABES

¿Cómo sería la implementación de un método para modificar un valor ya insertado en la base de datos?

Coméntalo en el foro de la unidad.

11. PERSISTENCIA

El término persistencia siempre va asociado a los datos y la información. En este sentido, es importante saber que, en todo tipo de aplicación, almacenar y cargar los datos es una tarea fundamental. En una aplicación móvil se debe almacenar, mínimamente, el estado de la misma tras el último uso del usuario para que, al regresar nuevamente, se encuentre con el estado anterior. La persistencia en el sistema Android viene determinada por varias formas, por ejemplo, el gestor de base de datos SQLite descrito en el apartado anterior. Otras alternativas pueden ser las preferencias y los ficheros estáticos.

11.1 Preferencias

Las preferencias almacenan información simple (datos primitivos de tipo numérico, booleano, cadenas de caracteres, etc.) a través de pares nombre y valor (key/value, name/value). Un ejemplo de preferencias puede ser el tono de llamada o de mensajes.

Para crear una preferencia de tipo nombre/valor se hace uso de un método llamado getSharedPreferences(). Este método requiere de 2 parámetros: el primero de ellos (prefs) es el nombre asignado al fichero de preferencias, pudiendo existir tantos como se deseen; el segundo parámetro es el modo de creación del fichero de preferencias. Puede tomar diferentes valores como los siguientes: MODE_PRIVATE, las preferencias solo pueden ser accesibles desde la propia aplicación. Los dos siguientes, MODE_WORLD_READABLE y MODE_WORLD_WRITEABLE, permiten que las preferencias sean visibles desde el exterior, pero no se aconseja para evitar problemas de seguridad.

```
SharedPreferences misPreferencias = getSharedPreferences("prefs",
MODE_PRIVATE);
```

Una vez que se tiene acceso a dichas preferencias, es posible crearlas, borrarlas o modificarlas a través del interfaz SharedPreferences.Editor mediante el método edit() de la siguiente forma:

```
SharedPreferences.Editor editor = misPreferencias.edit();
```

A través de esta interfaz se pueden establecer, borrar o modificar las preferencias:

```
editor.putString("nombre", cuadroTexto.getText().toString());
if (mujer.isChecked()) editor.putString("sexo",
mujer.getText().toString());
else editor.putString("sexo", hombre.getText().toString());
if (familia.isChecked()) editor.putBoolean("familia", true);
```

```
else editor.putBoolean("familiar", false);
if (mayor18.isChecked()) editor.putBoolean("mayor18", true);
else editor.putBoolean("mayor18", false);
```

Para almacenar las preferencias es necesario llamar al método commit() o apply(). La diferencia entre uno y otro es que commit() almacena de forma síncrona e informa del éxito o del fracaso del almacenamiento. Por su parte, apply() almacena de forma asíncrona y no informa del resultado final.

```
editor.apply();
```

Para recuperar las preferencias basta con realizar la llamada correspondiente y asociarla a una variable del mismo tipo. Por ejemplo:

```
String usuario = misPreferencias.getString("nombre", "en blanco");
String sexo = misPreferencias.getString("sexo", "en blanco");
boolean familiaNum = misPreferencias.getBoolean("familia", false);
boolean esmayor18 = misPreferencias.getBoolean("mayor18", false);
```

11.2 Ficheros estáticos

Son una solución para la persistencia de datos como alternativa al gestor de base de datos SQLite y a las preferencias. Se caracterizan porque solo se van a poder realizar operaciones de lectura, siendo imposible crearlos o/y modificarlos en tiempo de ejecución. Los ficheros estáticos se almacenan dentro de un directorio llamado /res/raw (utilizado en secciones anteriores para los ficheros multimedia, por ejemplo), pudiendo contener un diccionario u organizador de palabras, datos, etc.

Para acceder a estos ficheros se hace uso de dos clases llamadas Resources e InputStream, proporcionando los mismos mecanismos que el tratamiento de ficheros en Java. Por ejemplo, para acceder a un fichero estático llamado datos, se puede hacer de la siguiente forma:

```
Resources recursos = getResources();
InputStream archivo = recursos.openRawResource(R.raw.datos);
```

A través del objeto archivo se puede acceder al contenido del fichero datos con las técnicas de programación ofrecidas por Java.

VIDEO DE INTERÉS

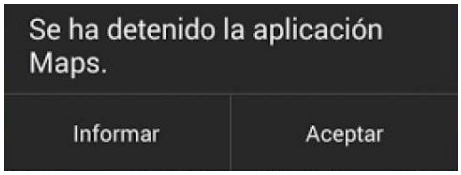
En el vídeo que encontrarás en el siguiente enlace podrás visualizar una implementación de Java con ficheros mediante la clase InputStream:

https://www.youtube.com/watch?v=4TFi4tyBQGw

12. MODELO DE HILOS

En la programación de dispositivos móviles puede ocurrir la situación de tener que ejecutar una operación en segundo plano para no obstaculizar el uso de la aplicación, produciendo esperas no deseadas. Algunas operaciones, como por ejemplo la descarga de un archivo de gran capacidad, son lentas y tediosas. Otras operaciones, que pueden parecer rápidas, pueden resultar igualmente lentas, si dependen del tamaño de un archivo o de algún otro factor externo como la red. Los dispositivos continuamente pierden calidad de la señal o pueden cambiar de Wifi a 3G sin preguntar previamente al usuario, lo que conlleva perder conexiones o demorarlas durante el proceso. En este sentido, son los famosos hilos los que van a aparecer para ejecutar tareas simultáneas o para realizar operaciones que se ejecutan con una periodicidad temporal determinada.

En cuanto a la interfaz gráfica, los hilos son parte fundamental para una interacción fluida con el usuario. Si una aplicación realiza una operación lenta en el mismo hilo de ejecución de la interfaz gráfica, el lapso de tiempo que dure la conexión, la interfaz gráfica dejará de responder. Este efecto es indeseable ya que el usuario no lo va a comprender. En algunos casos, si la "congelación" tiene una duración mayor a dos segundos, es probable que el sistema operativo muestre el diálogo ANR (*Application not responding*), invitando al usuario a acabar con la aplicación:



Para evitar esto se debe crear otro hilo (Thread) de ejecución que realice la operación lenta.

ENLACE DE INTERÉS

A continuación, te proponemos un enlace interesante sobre los hilos:

http://www.jtech.ua.es/dadm/2011-2012/restringido/android-av/sesion01-apuntes.html

12.1 Creación y ejecución de hilos

Un hilo o thread es un objeto con un método llamado run(). Hay dos formas de crearlos: por herencia a partir de la clase Thread o bien mediante la interfaz Runnable, que obliga a implementar un método run().

```
public class Hilo1 extends Thread {
 @Override
 public void run() {
 while(condicion_de_ejecucion){
 //Realizar operaciones
 //...
 try {
 // Dejar libre la CPU durante
 // unos milisegundos
 Thread.sleep(100);
 } catch (InterruptedException e) {
 return;
 }
 }
 }
}
public class Hilo2 implements Runnable {
 @Override
 public void run() {
 while(condicion_de_ejecucion){
 //Realizar operaciones
 //...
 try {
 // Dejar libre la CPU durante
 // unos milisegundos
 Thread.sleep(100);
 } catch (InterruptedException e) {
 return;
 }
 }
 }
}
```

La diferencia está en la forma de crearlos y ejecutarlos:

```
Hilo1 hilo1 = new Hilo1();
hilo1.start();

Thread hilo2 = new Thread(new Hilo2());
hilo2.start();
```

Una forma todavía más compacta de crear un hilo sería la declaración de la clase en línea:

```
new Thread(new Runnable() {
 public void run() {
 //Realizar operaciones ...
 }
}).start();
```

Si el hilo necesita acceder a datos de la aplicación se le puede pasar a través del método constructor. Por ejemplo:

```
@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 new Hilo1(getApplicationContext());
}

public class Hilo1 extends Thread {
 Context context;
 public Hilo2Thread(Context context){
 this.context = context;
 this.start();
 }
 @Override
 public void run() {
 while(condicion_de_ejecucion){
 //Realizar operaciones
 //...
 try {
 // Dejar libre la CPU durante
 // unos milisegundos
 Thread.sleep(100);
 } catch (InterruptedException e) {
 return;
 }
 }
 }
}
```

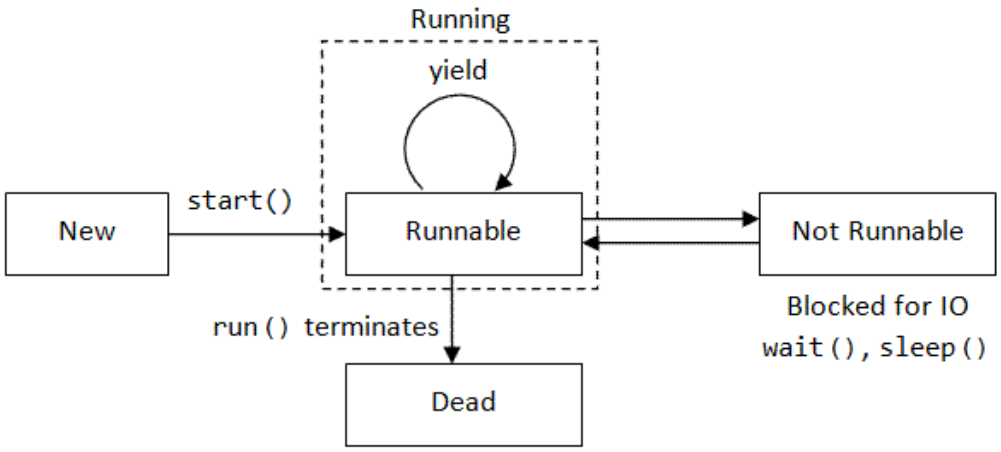
En el anterior ejemplo también se ha ejecutado el hilo desde su propio constructor, de manera que la creación del objeto Hilo1 ha sido suficiente para ejecutarlo.

12.2 Ciclo de vida de los hilos

Los hilos tienen un ciclo de vida pasando por diferentes estados:

- New: El primer estado de un hilo recién creado. Permanece en este estado hasta que el hilo es ejecutado.
- Runnable: Una vez ejecutado pasa a este estado, durante el cual ejecuta su tarea.
- Not runnable: Estado que permite al hilo desocupar la CPU en espera a que otro hilo termine o le notifique que puede continuar, o bien a que termine un proceso de E/S, o bien a que termine una espera provocada por la función Thread.sleep(100). Tras ello, volverá al estado Runnable.
- Dead: Pasa a este estado una vez finalizado el método run().

El siguiente diagrama resume las transiciones entre los estados del ciclo de vida de un hilo.



13. COMUNICACIONES: CLASES ASOCIADAS. TIPOS DE CONEXIONES

En esta sección se detalla el proceso de comunicación existente entre unas aplicaciones y otras haciendo uso de Android Studio. Esta comunicación se va a realizar a través del envío y recibo de mensajes entre ellas a través del IDE. En sistemas operativos Android las aplicaciones se ejecutan en espacios de memoria separados, aumentando así la seguridad al impedir que una aplicación afecte al funcionamiento de cualquier otra.

13.1 Clases asociadas

En aquellas ocasiones en las que sea necesario enviar mensajes entre aplicaciones, Android Studio provee de una clase que permite la transmisión y la recepción automática de los mismos, siempre y cuando la aplicación que recibe el mensaje se encuentre preparada para ello. En este sentido, para la transmisión de mensajes se hace uso de un objeto de la clase Intent. Al inicializar el objeto se debe tener en cuenta el hecho de indicar el nombre del paquete y la clase que debe recibir el mensaje, así como la acción a realizar dentro del mensaje, permitiendo al receptor obtener toda la información que le sea necesaria.

13.2 Tipos de conexiones

En este apartado se van a mostrar 2 tipos: enviar un mensaje por parte de una aplicación origen y recibir un mensaje por parte de una aplicación destino. Lo ideal es crear 2 proyectos con interfaz sencilla en Android Studio, uno para el origen y otro para el destino y comprobar cómo los mensajes son enviados y recibidos, respectivamente.

En el primero de los casos, este sería el código de la clase que envía el mensaje (origen):

```
Intent intent = new Intent();
intent.setClassName("NOMBRE_DEL_PAQUETE", "NOMBRE_DE_LA_CLASE");
intent.setAction("NOMBRE_DE_LA_ACCIÓN");
intent.putExtra("NOMBRE_DEL_PARÁMETRO_1", "VALOR_DEL_PARÁMETRO_1");
…
intent.putExtra("NOMBRE_DEL_PARÁMETRO_N", "VALOR_DEL_PARÁMETRO_N");
```

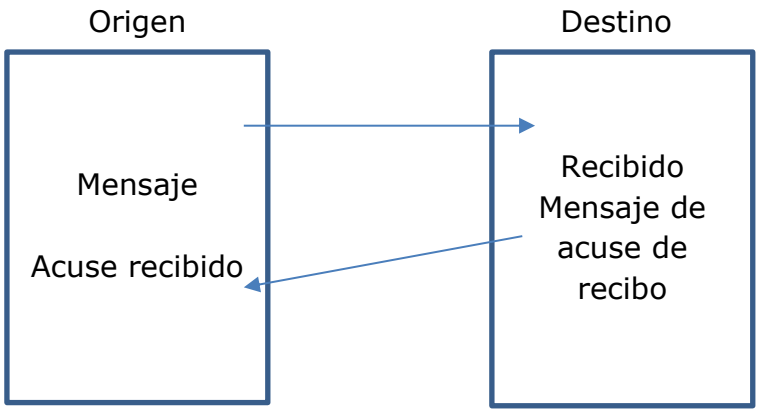
Una vez creado el mensaje, se envía mediante una llamada al método startActivity(intent).

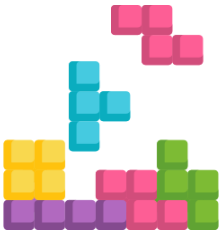
En el segundo de los casos, para recibir un mensaje se debe definir la acción en el manifiesto de la aplicación asociándolo a la misma clase que se usa al enviarlo, tal y como se indica a continuación:

```
<receiver android:name="NOMBRE_DE_LA_CLASE" android:exported="true">
  <action android:name="NOMBRE_DE_LA_ACCIÓN" />
</receiver>
```

El atributo exported a true indica que la clase puede ser activada desde otras aplicaciones, como es en este caso (desde el origen).

A la hora de recibir el mensaje por parte de la aplicación destino, esta debe devolver algún resultado que corrobore la llegada del mensaje origen. Para ello se puede crear un nuevo objeto de la clase Intent para ser enviado a la aplicación origen.





EJEMPLO PRÁCTICO

A través del siguiente enlace, se propone realizar un caso práctico para la comunicación entre 2 interfaces a través de sus actividades. Se trata de una práctica interesante ya que rescata conceptos aprendidos en secciones anteriores junto con la comunicación detallada anteriormente.

https://www.develou.com/desarrollo-android-intents/

14. GESTIÓN DE LA COMUNICACIÓN INALÁMBRICA Y BÚSQUEDA DE DISPOSITIVOS

En cuanto a la gestión de la comunicación inalámbrica, se propone como ejemplo la creación de un proyecto que permita activar y desactivar la WiFi en un dispositivo móvil a través de Android Studio. Para ello, lo primero a realizar es crear dicho proyecto (en blanco) e ir obteniendo dos imágenes que permitan reconocer si la conectividad inalámbrica WiFi está activa o no. Por ejemplo, las siguientes imágenes dan una idea de ello:

El proyecto va a ser llamado WiFi y, simplemente, tendrá una interfaz con un componente que permita ser pulsado para activar la WiFi y ser nuevamente pulsado para desactivar la WiFi.

Una vez creado el proyecto y descargadas ambas imágenes, estas se añaden en la carpeta /res/drawable del proyecto.

En primer lugar, se asocia la imagen WiFi desactivado (wifioff) al componente, puesto que inicialmente no se desea activar la WiFi. Para ello, se añade la siguiente línea de código a la definición de su componente en el fichero XML: android:src="@drawable/wifioff" y, además, android:scaleType="fitXY" para ajustarse al tamaño del botón, obteniéndose como resultado lo siguiente:


```
<ImageButton
    android:id="@+id/imbutton"
    android:layout_width="wrap_content"
    android:layout_height="128dp"
    android:layout_marginLeft="120dp"
    android:layout_marginTop="104dp"
    android:src="@drawable/wifioff"
    android:scaleType="fitXY"

app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"
/>
```

Una vez diseñada esta sencilla interfaz, es necesario establecer permisos para poder acceder al estado de la conectividad inalámbrica WiFi. Esto se realiza a través del fichero de manifiesto (AndroidManifest.xml) añadiendo las siguientes líneas:

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

La primera de ellas va a permitir tener acceso a la conectividad WiFi y la segunda dar permisos para cambiar el estado de la WiFi (ON/OFF). El siguiente paso consiste en crear un ImageButton para ser asociado al componente creado anteriormente con id imageButton. Para ello, es necesario implementar la siguiente línea de código en el ActivityMain.java:

```
ImageButton im = (ImageButton) findViewById(R.id.imbutton);
```

Para hacer uso de la tecnología WiFi y poder realizar acciones gestiones sobre ella, se implementa un objeto de la clase WifiManager:

```
WifiManager admin;
```

Para continuar con la gestión de la WiFi será necesario acceder a los servicios del sistema mediante el objeto admin de la siguiente forma:

```
admin = (WifiManager) this.getSystemService(Context.WIFI_SERVICE);
```

Si la anterior línea de código no fuese del agrado de Android Studio y se produjera un error, podría sustituirse por esta otra equivalente:

```
admin = (WifiManager)
this.getApplicationContext().getSystemService(Context.WIFI_SERVICE);
```

El siguiente paso va a ser la creación de un método llamado estadoWiFi que será llamado cada vez que se pulse sobre la imagen. Este método comprobará si la WiFi está activa, para desactivarla, o si la WiFi está desactiva, para activarla. Puede ser implementado de la siguiente forma:

```
public void estadoWifi() {
    if (admin.isWifiEnabled()) admin.setWifiEnabled(false);
    else admin.setWifiEnabled(true);
}
```

El penúltimo paso será la creación del evento para el ImageButton (onClick) y asociarle el método estadoWiFi:

```
im.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        estadoWifi();
    }
});
```

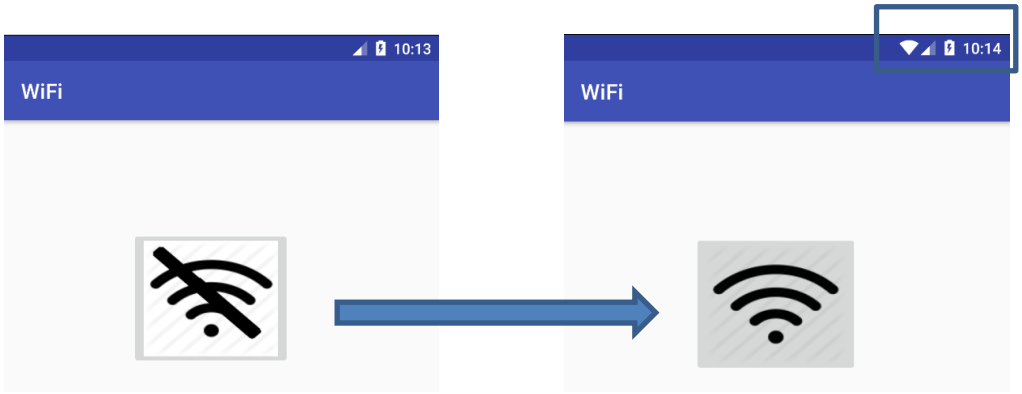
El último paso será la creación de otro método para poder cambiar la imagen siempre que se cambie el estado de la WiFi:

```
public void setImagenWiFi(boolean estado) {
    if (estado) im.setImageResource(R.drawable.wifion);
    else im.setImageResource(R.drawable.wifioff);
}
```

Tras la implementación de este último método se ha de modificar el método estadoWiFi y añadirle una llamada para el cambio de la imagen:

```
public void estadoWifi() {
    setImagenWiFi(!admin.isWifiEnabled());
    if (admin.isWifiEnabled()) admin.setWifiEnabled(false);
    else admin.setWifiEnabled(true);
}
```

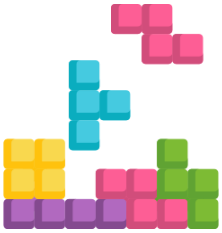
El resultado final de la implementación es el siguiente:



COMPRUEBA LO QUE SABES

¿Qué otros tipos de conectividad inalámbrica existen en los dispositivos móviles?

Coméntalo en el foro de la unidad.



EJEMPLO PRÁCTICO

De igual manera que se ha estudiado la gestión de la comunicación vía WiFi a través de la siguiente configuración en el archivo de manifiesto:

```
<uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission
android:name="android.permission.CHANGE_WIFI_STATE"/>
```

Y de la utilización de la clase WifiManager para la creación de un objeto admin:

```
admin = (WifiManager) this.getApplicationContext().
getSystemService(Context.WIFI_SERVICE);
```

Realiza una interfaz con un ImageButton para la gestión de la comunicación vía Bluetooth, activándolo y desactivándolo. Usa el siguiente enlace para ayudarte a realizar la práctica:

https://danielggarcia.wordpress.com/2013/10/19/bluetooth-i-activando-y-desactivando-el-bluetooth-en-android/

15. ESTABLECIMIENTO DE LA CONEXIÓN. CLIENTE Y SERVIDOR

En primer lugar, es importante definir los conceptos de cliente y de servidor. Un cliente es un dispositivo conectado en red (PC, móvil, tablet, etc.) el cual realiza, a través de la red, peticiones de diversa índole (solicitud de página web, descarga de un fichero, dato de una base de datos, etc.). Un servidor es un equipo capaz de atender las peticiones que le llegan y dar respuesta a las mismas.

En este sentido, la arquitectura cliente-servidor presenta un diseño con un reparto de tareas entre los demandantes (clientes) y los proveedores (servidores). Esta comunicación se lleva a cabo a través de los llamados protocolos (HTTP, HTTPS, FTP, DNS, etc.).

En Android Studio existen algunas clases como son URL, URLConnection, HttpURLConnection y HttpsURLConnection capaces de establecer conexión entre clientes y servidores de Internet bajo la funcionalidad de los protocolos del modelo TCP/IP. Se realizan conexiones para la lectura y escritura para:

- Servidores FTP. Establecimiento de una conexión con un servidor de ficheros mediante FTP. Para ello es necesario construir la conexión así: ftp://usuario:password@servidor/ruta
- A través de ficheros locales. Accediendo al sistema de archivos locales mediante una conexión a file:ruta_fichero
- HTTP y HTTPS. Realizando conexiones mediante URLs conocidas por su uso en los navegadores como http://servidor/ruta: puerto o, en el caso de https, mediante https://servidor/ruta:puerto.

La creación de una conexión involucra realizar las siguientes actividades:

1. Crear un proyecto nuevo y, a partir de la clase URL, crear un objeto. Para ello ha sido necesario realizar un import de la clase URL:

   ```
   import java.net.URL;
   ```

   E introducir la creación del objeto mediante un try/catch:

   ```
   URL conexión = null;
   try {
       conexion = new URL("ftp://www.rediris.es/");
   } catch (MalformedURLException e) {
       e.printStackTrace();
   }
   ```

2. A continuación, es necesario usar unas clases de las mencionadas anteriormente, que será del tipo URLConnection, HttpURLConnection o HttpsURLConnection, que realizará la conexión a la URL a través de un método conocido como openConnection(), bajo su try/catch de la siguiente forma:

```
URLConnection conexionURL = null;
try {
    conexionURL = (HttpURLConnection) conexion.openConnection();
} catch (IOException e) {
    e.printStackTrace();
}
```

3. El objeto conexionURL contiene un stream de datos el cual debe ser leído a través de un objeto de la clase BufferedStreamReader o de la clase InputStreamReader a través de las siguientes líneas:

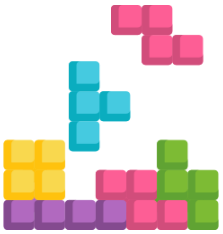
```
InputStream stream = null;
try {
    stream = new
BufferedInputStream(conexionURL.getInputStream());
    String contenido = lectura (stream);
    //stream.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

4. Se debe crear un método llamado lectura, que tome como parámetro el objeto stream y sea leído en su totalidad, línea a línea hasta llegar al final del mismo. En este caso se ha realizado una petición mediante FTP a un servidor de rediris, con lo cual, el funcionamiento del mismo dependerá de la disponibilidad del servidor. No obstante, la llamada a realizar podría ser a una URL de una página web en concreto y obtener su respuesta para ser posteriormente tratada. El método para convertir el stream de datos con la respuesta del servidor a un String (cadena de caracteres) es el siguiente:

```
// convert InputStream to String
private static String lectura(InputStream is) {

    BufferedReader br = null;
    StringBuilder sb = new StringBuilder();

    String line;
    try {

        br = new BufferedReader(new InputStreamReader(is));
        while ((line = br.readLine()) != null) {
            sb.append(line);
```

```
            }

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return sb.toString();
    }
```

De esta manera, línea a línea, se va leyendo el stream de datos y se devuelve un String mucho más manejable con la respuesta del servidor a la petición del cliente.



EJEMPLO PRÁCTICO

Prueba a realizar una conexión vía cliente-servidor a algún otro FTP que conozcas o a una URL cualquiera. Además, haz uso del método lectura para obtener la respuesta en forma de String.

Antes de realizar la conexión con el FTP, es importante cerciorarse de que el servidor está activo y responde a las peticiones. Para ello, puede ser útil descargarse un programa cliente FTP y realizar dicha comprobación. A continuación, se facilita una URL interesante:

https://www.redeszone.net/2016/12/22/los-tres-mejores-clientes-ftp-podemos-encontrar-gratis/

16. ENVÍO Y RECEPCIÓN DE MENSAJES TEXTO. SEGURIDAD Y PERMISOS

Los mensajes de texto (SMS, de las siglas en inglés *Short Message Service*) han sido utilizados sobre todo al comienzo del uso de los dispositivos móviles, cuando las aplicaciones de mensajería web (WhatsApp, Telegram, Line, chats, redes sociales, etc.) no habían visto la luz. A pesar de que el número de SMS ha descendido considerablemente, este tipo de mensajería aún está disponible para el usuario y para Android Studio, proporcionando funcionalidad para ello.

Para el envío de mensajes (SMS) a través de Android Studio, se hace uso de una clase llamada SMSManager y, a su vez, se realiza una llamada a un método estático conocido como getDefault que va a ser el responsable de obtener una referencia al objeto de la clase SMSManager. A la hora de enviar la información mediante un SMS corto, se llama al método sendTextMessage, como se verá en unas líneas más adelante.

Es importante, como se indica en el título de esta sección, atender a la seguridad y a los permisos. Para ello, en Android Studio se requiere escribir en el fichero de manifiesto (AndroidManifest.xml) la siguiente línea de código:

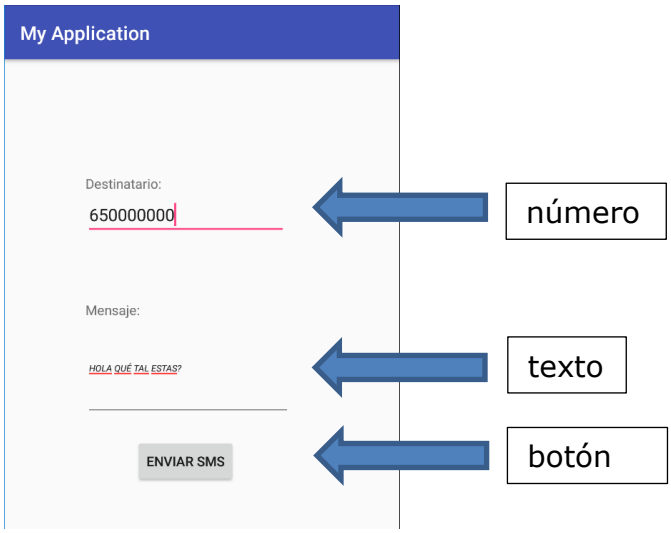
```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

A través de un proyecto Android Studio, se va a crear una interfaz sencilla con un cuadro de texto para añadir el número de teléfono del destinatario, un segundo cuadro de texto para añadir el contenido del SMS y un botón que realice el envío. Para ello lo primero es crear estos 3 componentes:

```
EditText numero = null;
EditText texto = null;
Button boton = null;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    numero = (EditText)findViewById(R.id.editText2);
    texto = (EditText)findViewById(R.id.textView3);
    boton = (Button)findViewById(R.id.button);
```

El aspecto de la interfaz tras la creación de los componentes mencionados es el siguiente:



En este momento, se creará un evento onClick en el botón y este, a su vez, llama al método Enviar_SMS:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    boton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Enviar_SMS(v, numero, texto);
        }
    });
```

El método Enviar_SMS recoge como parámetros el View actual, el número del destinatario y el texto del mensaje a enviar. Es en este método donde se hará uso de la clase SMSManager para el envío del mensaje de texto:

```
private void Enviar_SMS(View view, EditText numero, EditText texto) {
    SmsManager sms = SmsManager.getDefault();
    // Pasar datos a String
    String destinatario = numero.getText().toString();
    String mensaje = texto.getText().toString();
    sms.sendTextMessage(destinatario, null, mensaje, null, null);
}
```

Para la llamada al método sendTextMessage ha sido necesario pasar tanto el número de teléfono (numero) como el mensaje de texto (texto) a String, ya que sus parámetros así lo requerían.

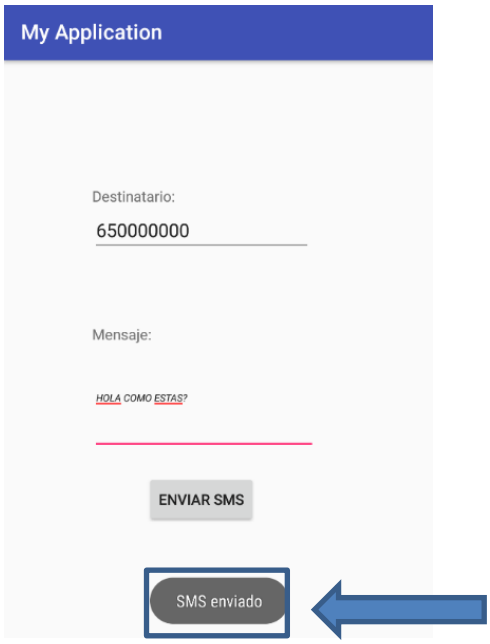
Es importante controlar los posibles errores que puedan surgir a la hora de enviar un SMS. Por tanto, el método Enviar_SMS sería mucho más eficiente si implementase todo bajo un try/catch, de tal forma que tanto si el SMS llega al destino como si no, se muestra un mensaje informativo:

```
private void Enviar_SMS(View view, EditText numero, EditText texto) {
    try {
        SmsManager sms = SmsManager.getDefault();
        // Pasar datos a String
        String destinatario = numero.getText().toString();
        String mensaje = texto.getText().toString();
        sms.sendTextMessage(destinatario, null, mensaje, null, null);
        // Mensaje OK
        Toast.makeText(getApplicationContext(), "SMS enviado",
                Toast.LENGTH_LONG).show();
    } catch (Exception e) {
        // Mensaje fallido
        Toast.makeText(getApplicationContext(), "SMS NO enviado",
                Toast.LENGTH_LONG).show();
        e.printStackTrace();
    }
}
```

Es posible que, llegado a este punto, el envío de SMS dé como resultado un error. Esto es debido a que la API que se está utilizando requiere añadir una línea en el fichero MainActivity.java para que el usuario pueda realizar el envío de SMS. La línea clave es la siguiente:

```
ActivityCompat.requestPermissions(this,new String[]
{Manifest.permission.SEND_SMS}, 1);
```

Gracias a ella, una vez se realice el envío pulsando el botón se muestra información al usuario para que permita realizar el envío. De esta manera, el resultado tras enviar el SMS es el siguiente:

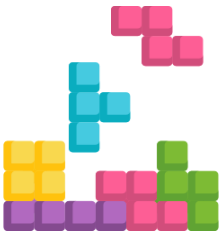


Para el envío real de SMS es necesario tener contratado un proveedor de servicios. Por tanto, desde el simulador no se realizará dicho envío, pero de forma simulada e implementando este software en un dispositivo se podrá realizar el envío de SMS.

COMPRUEBA LO QUE SABES

¿Qué otros tipos de aplicaciones de mensajería web existen en la actualidad y son soportados por los dispositivos móviles?

Coméntalo en el foro de la unidad.



EJEMPLO PRÁCTICO

Se pretende realizar una interfaz de usuario a través de Android Studio que contenga los diferentes componentes estudiados. Uno de ellos debe dejar escribir un número de teléfono, otro de ellos debe dejar escribir un mensaje de texto y uno de los botones debe tener implementado un evento para probar a enviar dicho mensaje a dicho número de teléfono móvil. Puedes hacer uso del ejemplo mostrado anteriormente para realizar la práctica. Recuerda que, al ejecutar el programa en un simulador, el SMS no llegará a su destino, pero existe una comprobación en el ejemplo anterior para corroborar su envío en caso de éxito o su no envío en caso de error.

17. CONTENIDO MULTIMEDIA

El sistema operativo Android ofrece varios formatos multimedia (audio, imagen y vídeo) y protocolos de red. El principal problema es que existe una dependencia de la versión de Android para soportar algunos de ellos, es decir, según la versión utilizada, se podrán visualizar unos u otros. No obstante, a continuación, se muestra un listado de aquellos contenidos multimedia que Android, en sus diferentes versiones, soporta:

Protocolos de red para streaming de audio/vídeo	Audio	Imagen	Vídeo
RTSP (RTP, SDP) HTTP/HTTPS streaming progresivo HTTP/HTTPS streaming en vivo (a partir de Android 3.0) HTTPS no soportado en versiones anteriores a Android 3.1	AAC LC FLAC MP3 MIDI Ogg Vorbis	JPEG GIF PNG BMP WebP	H.263 H.264 AVC MPEG-4 SP VP8

ENLACE DE INTERÉS

En el siguiente enlace encontrarás documentación relacionada con los formatos multimedia que están soportados por Android:

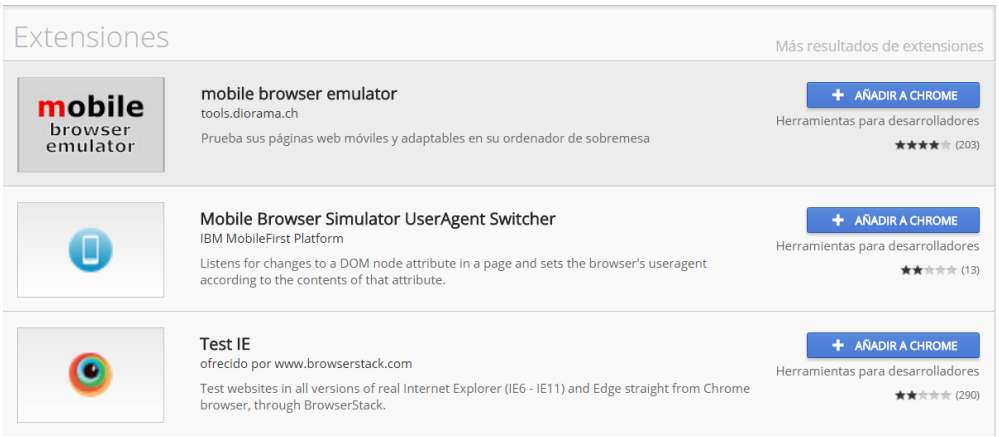
https://developer.android.com/guide/topics/media/media-formats.html

18. COMPLEMENTOS DE LOS NAVEGADORES PARA VISUALIZAR EL ASPECTO DE UN SITIO WEB EN UN DISPOSITIVO MÓVIL

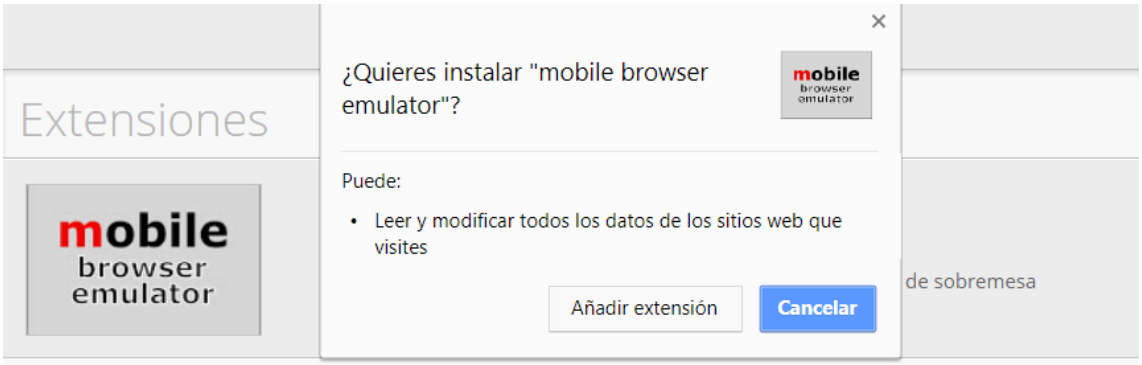
A la hora de desarrollar una página web para dispositivos móviles, es importante ver el aspecto final que tendría la página en diferentes formatos. A esto se le conoce como Web Responsive. Ciertos navegadores ofrecen complementos que permiten simular el comportamiento de la página web desarrollada en un dispositivo móvil, como el caso de Google Chrome a través del complemento "Mobile Browser Emulator".

Para descargar y probar este complemento es importante tener instalado Google Chrome. Tras ejecutarlo se debe acceder a la Chrome Web Store a través de: https://chrome.google.com/webstore/category/extensions?hl=es En ella existe un buscador donde se introduce el nombre del complemento que se desea instalar. En este caso: Mobile Browser Emulator.

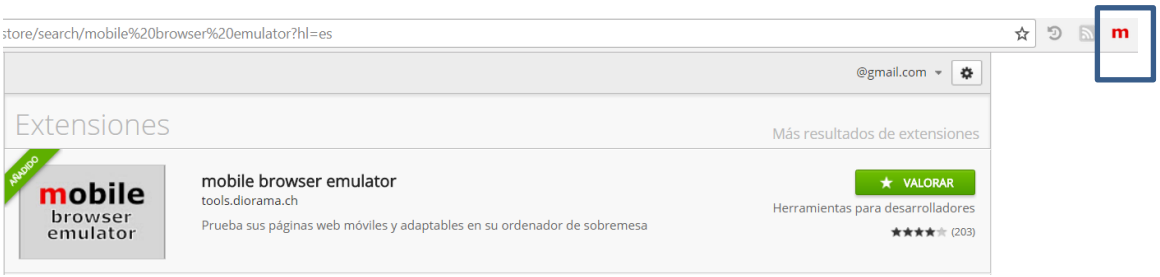
Google Chrome, tras la búsqueda del complemento, muestra alternativas como se observa en la siguiente figura:



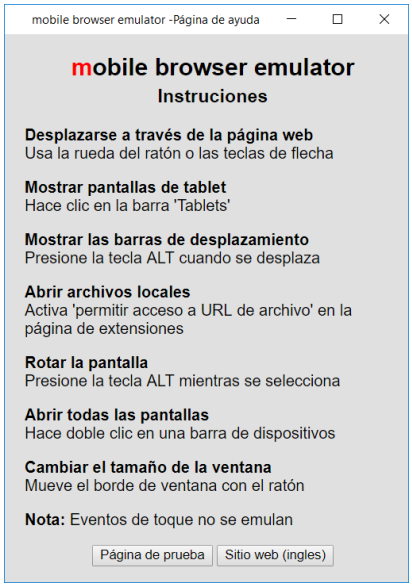
Y se procede a instalar la primera de ellas que, como puede comprobarse, es la que mejor puntuación tiene. Para ello, basta con seleccionar el botón "Añadir a Chrome". Es muy posible que el navegador realice una pregunta al usuario para corroborar que realmente desea instalar la extensión o el complemento Mobile Browser Emulator, como se muestra en la siguiente imagen. Bastará con aceptar esta instalación a través del botón "Añadir extensión".



Una vez instalada, el complemento aparecerá en forma de icono situado a la derecha del navegador, como se muestra en la siguiente imagen:

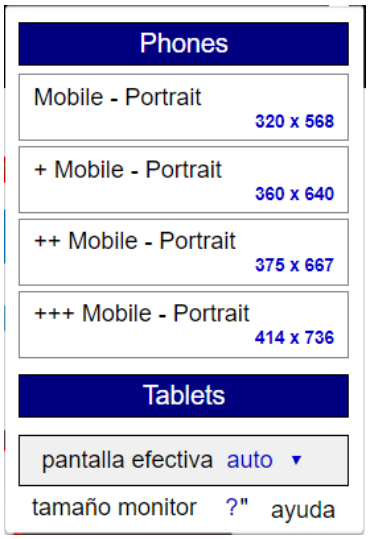


La ejecución de este complemento se realiza haciendo clic sobre dicho icono. Tras ello, se muestra al usuario la siguiente interfaz:



En esta interfaz se muestra una guía de usuario para probar las diferentes opciones del complemento.

Para probar verdaderamente el simulador, se puede acceder, por ejemplo, a la página de Cesur a través del navegador: www.cesurformacion.com/ y, una vez abierta, pulsar sobre el icono del complemento. El simulador ofrece la posibilidad de visualizar la página a través de diferentes formatos:



Basta con seleccionar cualquiera de ellos para poder visualizar la web, en este caso, de Cesur, a través del simulador:

Para visualizar la página web en su totalidad, se hará uso del ratón, a través de la rueda, mediante el teclado o a través de las flechas de desplazamiento.

COMPRUEBA LO QUE SABES

¿Qué otros tipos de simuladores como complementos de navegadores conoces? Si no has utilizado ninguno, prueba a investigar sobre alguno, por ejemplo, para Firefox.

Coméntalo en el foro de la unidad.

ENLACE DE INTERÉS

A través del siguiente enlace se puede comprobar cómo existen otras posibilidades para poder visualizar páginas web con Google Chrome simulando un dispositivo móvil. Accede a ella para obtener mayor detalle:

http://www.vissit.com/prueba-tu-web-en-diferentes-dispositivos-con-chrome/

19. PRUEBAS Y DOCUMENTACIÓN

Tanto las pruebas como la documentación son dos fases del desarrollo de software de gran importancia. Las pruebas son necesarias para corroborar el correcto funcionamiento del aplicativo y para la resolución de errores antes de que el software esté disponible para los clientes o usuarios. Por su parte, la documentación es una tarea que se debe ir haciendo a lo largo de todo el desarrollo de la aplicación, desde la recogida de requisitos antes de llevarse a cabo la implementación, hasta la implantación o publicación final. Es muy importante que todos los procesos involucrados estén documentados para que, en el caso de tener que realizar revisiones o modificaciones, se pueda recurrir a ella para ayudar a comprender su desarrollo y puesta a punto.

19.1 Pruebas

La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. El hecho de realizar pruebas del software nos proporciona una garantía de calidad del software. Además, representa una revisión final de las especificaciones, del diseño y de la codificación. La prueba no puede asegurar la ausencia de errores en el programa, solo puede indicar su existencia. Un buen caso de prueba será aquel que muestre un error no descubierto hasta entonces.

Se deben seguir los siguientes principios para la realización de pruebas:

- Las pruebas deben ser llevadas a cabo por personas diferentes a las que desarrollaron el programa, tanto para verificar que el programa funciona correctamente como para validar que ese programa ha sido concebido e interpretado correctamente.
- Los casos de pruebas deben ser escritos tanto para condiciones de entradas inválidas o inesperadas como para condiciones válidas y esperadas.
- La posibilidad de encontrar errores adicionales en una sección del programa es proporcional al número de errores ya encontrados en la misma sección.

Podemos distinguir tres fases en el proceso de pruebas a seguir en el desarrollo de un sistema software:

- Pruebas de unidad: Están destinadas a la prueba de cada una de las unidades del sistema, esto es, se centran en la verificación de la menor unidad en el diseño del software (el módulo). Se verifican, por tanto, las estructuras de datos, la interfaz, las estructuras de control, el control de errores, las condiciones límite de funcionamiento y la

eficiencia del código. Dentro de cada una de estas, las pruebas se centrarán en unos u otros aspectos.

- Pruebas de integración: Una vez que se ha comprobado la corrección de errores en cada uno de los módulos por separado con las pruebas de unidad, la siguiente fase de pruebas va dirigida a probar la correcta interconexión de los módulos, ya que esto no queda garantizado con las pruebas de unidad. Algunos de los problemas que pueden surgir son la modificación no deseada de variables globales (causando posibles efectos laterales), el acarreo de imprecisiones por parte de varias funciones involucradas en el mismo cálculo.

- Pruebas de sistema: En esta fase, el sistema de software construido debe quedar en perfecto estado de funcionamiento. Para ello, se realizan diversas tareas como:
    - Adaptar y ultimar detalles acerca del hardware sobre el que finalmente se instalará el software.
    - Forzar errores para comprobar su capacidad de recuperación.
    - Verificar mecanismos de protección y seguridad incorporados.
    - Realizar pruebas que evalúen el rendimiento del sistema.
    - Realizar pruebas que pongan al sistema en situaciones anormales, con la finalidad de evaluar su estabilidad, conocidas como pruebas de estrés.

19.2 Documentación

Como se ha descrito anteriormente, la documentación es una tarea transversal que ha de realizarse desde el comienzo hasta el final. Es por ello que deben documentarse todos los procesos involucrados en el ciclo de vida de la aplicación. En este sentido, existen diferentes tipos de documentación, pudiéndose clasificar en los siguientes documentos:

- Documentación técnica: La guía técnica o manual técnico es el documento donde queda reflejado el diseño del proyecto o aplicación, la codificación de los programas y las pruebas realizadas para su correcto funcionamiento. Este documento está destinado al personal técnico en informática (analistas y programadores). El principal objetivo de este documento es el de facilitar el desarrollo, corrección y futuro mantenimiento de los programas de forma rápida y precisa.

- Documentación del diseño: Es el documento donde queda reflejada la solución de la aplicación, basándose en las necesidades del usuario y en el análisis efectuado previamente por los analistas con el fin de obtener los requerimientos de la aplicación. Este documento está

destinado a los programadores que van a desarrollar los programas de la aplicación con objeto de que la codificación sea efectuada con la mayor calidad y rapidez posible. Este documento debe estar realizado de tal forma que permita la división del trabajo de programación para que varios programadores puedan trabajar de forma independiente, aunque coordinados por uno de los programadores o por un analista. En el cuaderno de carga hay que ajustarse a las características del sistema físico donde se va a implantar dicha aplicación, así como al tipo de sistema operativo y lenguaje que se va a utilizar.

- Documentación del código fuente: A través de comentarios dentro del código de la aplicación como fuera de él, a través de documentos aclarativos que indiquen la funcionalidad de cada método, objeto, procedimiento, clase, etc.

- Documentación de las pruebas: En el documento de juego de pruebas se especifican el tipo de prueba, módulos y programas para los que se efectúan dichas pruebas, datos de entrada, datos previstos de salida y los datos reales de salida obtenidos en las pruebas.

- Guía de uso del aplicativo: La guía de uso o manual de usuario es el documento que contiene la información precisa y necesaria para que los usuarios utilicen correctamente los programas de la aplicación. La información contenida en la guía de usuario proviene de la guía técnica de la aplicación, pero se presenta de forma que el usuario la comprenda con toda claridad, prescindiendo de toda la parte técnica de desarrollo y centrándose en los aspectos de la entrada y salida de la información que maneja la aplicación. La guía de uso no debe hacer referencia a ningún apartado de la guía técnica, pues normalmente la guía técnica queda en poder de los desarrolladores y al usuario solamente se le proporciona las guías de uso y de instalación. La guía de uso debe estar redactada con un estilo claro y en el caso de que se haga necesario el uso de terminología informática no conocida por los usuarios, debe ir acompañada por un glosario de términos informáticos. Este documento puede ser utilizado para completar la formación que los usuarios de la aplicación deben recibir para el correcto manejo de la misma.

- Guía de instalación: La guía de instalación o manual de explotación es el documento que contiene la información necesaria para poner en marcha el sistema diseñado y para establecer las normas de explotación. Estas normas harán precisiones sobre las siguientes tareas:
    - Pruebas de implantación de los programas en el sistema físico donde van a funcionar en adelante, con la colaboración entre usuarios y desarrolladores.

- Forma en que se van a capturar los datos existentes en el sistema anterior al que se va a implantar. Esto implicará un trasvase de información que puede estar en soportes de uso manual o en soportes informáticos que exijan una conversión del formato de la información y la realización de programas de captura de datos.
- Pruebas del nuevo sistema con toda la información capturada, lo que equivale a tener funcionando en paralelo los dos sistemas, corrigiendo lo posibles errores de funcionamiento. Cuando estás pruebas sean satisfactorias, el nuevo sistema sustituye al anterior y entra en vigor la nueva documentación.

RESUMEN FINAL

Durante este tema se ha puesto de manifiesto la importancia de conocer el entorno de desarrollo Android Studio y se ha explicado la funcionalidad de las interfaces de usuario y algunos de los componentes más utilizados hoy en día en las aplicaciones móviles, así como diversas acciones y aspectos de la comunicación. Para resumir el tema se puede hacer una división del mismo en 3 partes diferenciadas.

Una primera parte donde Se ha estudiado el entorno de desarrollo Android Studio, su relación con Java y su puesta a punto. Se ha creado un primer proyecto de ejemplo y se han descrito acciones importantes en el desarrollo de software como son los procesos de compilación, empaquetado, ejecución y depuración.

Una segunda parte en la que se ha profundizado en el desarrollo de interfaces gráficas a través de diversos componentes proporcionados por Android Studio como botones, checkbox, radioButton, cuadros de texto, spinner, etc. Además, se ha trabajo con otros componentes como las imágenes y las imágenes de botón, muy utilizadas en aplicaciones actuales. Se han probado diferentes eventos (acciones) sobre los componentes, se ha aprendido la importancia de los servicios y de las bases de datos en sistemas Android y en ficheros persistentes. También se ha mostrado la necesidad del uso de hilos en aplicaciones concretas.

Y una tercera parte que se ha enfocado más en la comunicación entre aplicaciones, así como la comunicación a través de medios no guiados como vía Wifi. También se ha aprendido a realizar comunicaciones entre clientes y servidores, a enviar mensajes de texto (SMS) y a conocer la tecnología multimedia soportada por los sistemas Android. Se ha mostrado la existencia y la importancia de los complementos en algunos navegadores para simular páginas web en dispositivos móviles. Y, por último, se ha repasado la importancia de las pruebas y de la documentación en todo el desarrollo de aplicaciones, ya sean móviles o no.