Práctica 1
Introducción al desarrollo de aplicaciones Android

Grado en Ingeniería Informática - Ingeniería del Software
Dpto. de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura

1. Introducción y objetivos

Esta práctica os va a proporcionar una primera toma de contacto con el desarrollo de aplicaciones para dispositivos Android en el entorno de desarrollo AndroidStudio. El guión está basado en los tutoriales básicos de desarrollo con Android que están disponibles en `http://developer.android.com/training/index.html`.

En particular, los objetivos de esta práctica son los siguientes:

1. Conocer los fundamentos del desarrollo Android y tomar contacto con el entorno de desarrollo AndroidStudio.
2. Implementar y probar una pequeña aplicación Android de ejemplo.

Como resultado de la práctica crearéis una pequeña aplicación Android que podréis probar en un emulador o, si os apetece y tenéis un móvil o una tableta Android, en un dispositivo real.

2. Arranque y configuración básica del entorno de desarrollo AndroidStudio

El kit de desarrollo de software (SDK) de Android es un conjunto de aplicaciones y bibliotecas necesarias para desarrollar aplicaciones para móviles, tabletas y otros dispo-

sitivos que lleven como sistema operativo Android.

Aunque el SDK puede usarse desde línea de órdenes, Android ofrece un entorno de desarrollo denominado AndroidStudio, que está basado en el entorno de desarrollo IntelliJ y que ya integra el SDK de Android. Este entorno es el que usaremos en estas prácticas.

En el laboratorio de prácticas, laboratorio 0.04 del Edificio Ada Byron, el entorno AndroidStudio ya se encuentra instalado en los equipos arrancando con el sistema operativo CentOS e iniciando una sesión con el siguiente usuario y clave:

- **user:** guest
- **password:** diis7d5

Si queréis instalar AndroidStudio en vuestros propios computadores, podéis descargar el entorno de desarrollo desde `https://developer.android.com/sdk/index.html#Other`. Se recomienda instalar la opción "All Android Studio packages" para la plataforma que utilicéis habitualmente.

Adicionalmente debéis crear dos links simbólicos al `scratch`[1] de cada equipo con los siguientes comandos:

```
mkdir /mnt/scratch/android-guest
cd /tmp
ln -s /mnt/scratch/android-guest .
mkdir /mnt/scratch/.android-guest
cd $HOME
ln -s /mnt/scratch/.android-guest .android
```

Los primeros pasos que tenéis que dar en los equipos del laboratorio son los siguientes:

1. Abrir Android Studio. Abrid una terminal y teclead `/usr/local/android/android-studio/bin/studio.sh &`
   Podéis crear también un lanzador en el Escritorio haciendo clic con el botón derecho del ratón (`Create launcher`). Lo podéis denominar AndroidStudio y poner esa misma ruta en el campo Comando.
2. En el caso de abrir AndroidStudio por primera vez (o siempre que detecte que se haya eliminado la carpeta por defecto de almacenamiento de proyectos), os solicitará información para configurar el entorno:
   a) Aparecerá un diálogo que os preguntará si queréis importar la configuración de una instalación previa. Seleccionaréis la segunda opción: `"I do not have a previous ..."`.

[1] Espacio del disco duro local dedicado únicamente a almacenamiento temporal.

b) Aparecerá un asistente de configuración (Android Studio Setup Wizard). Pulsaréis `Next`.

c) Aparecerá un diálogo solicitando el tipo de instalación. Seleccionareis el tipo `Standard` y pulsaréis `Next`.

d) A pesar de que el SDK ya está descargado en `/opt/android/sdk`, aparecerá un diálogo con el resumen de los elementos que se deben descargar y actualizar (ver Figura 1). Pulsaréis el botón `Finish`.

e) Aparecerá un diálogo de error avisando de que no se puede instalar algún componente. Esto es debido a la configuración de los laboratorios, pero no pasa nada. Pulsaréis el botón `Cancel` y a continuación el botón `Finish`.

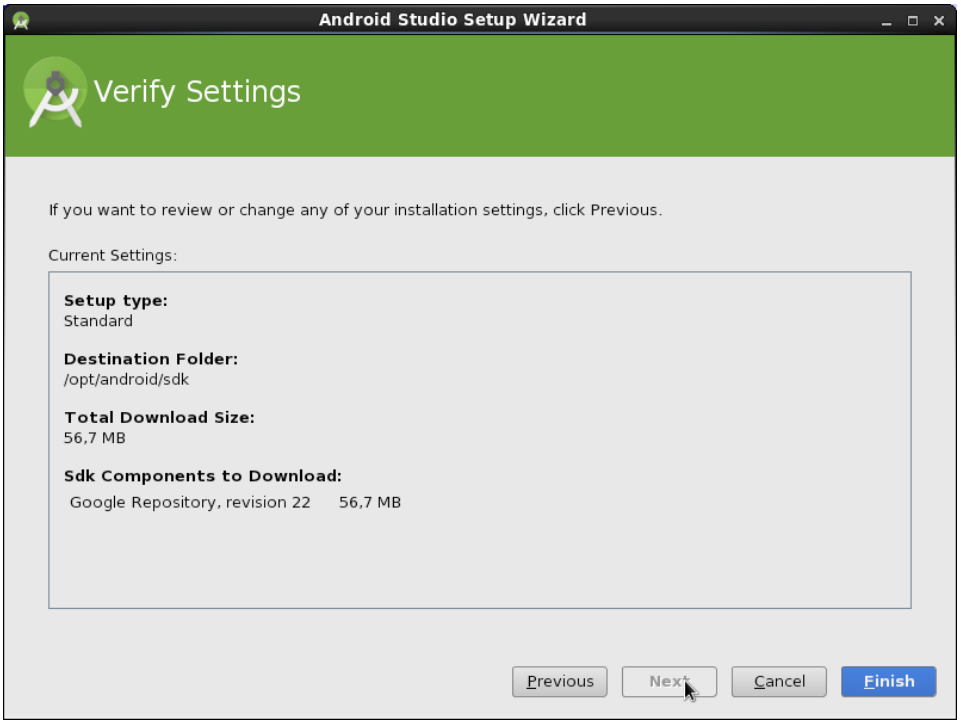

Figura 1: Verificando configuración con el asistente

3. Aplicación de ejemplo

Esta sección es un tutorial que os guiará para que podáis crear y probar una primera aplicación para Android, utilizando para ello AndroidStudio.

3.1. Crear proyecto Android: una aplicación con una actividad

En esta sección se describe cómo crear un proyecto para una pequeña aplicación Android, así tendréis ocasión de dar un vistazo rápido a, y probar, algunos de los conceptos básicos explicados en el Apéndice A.

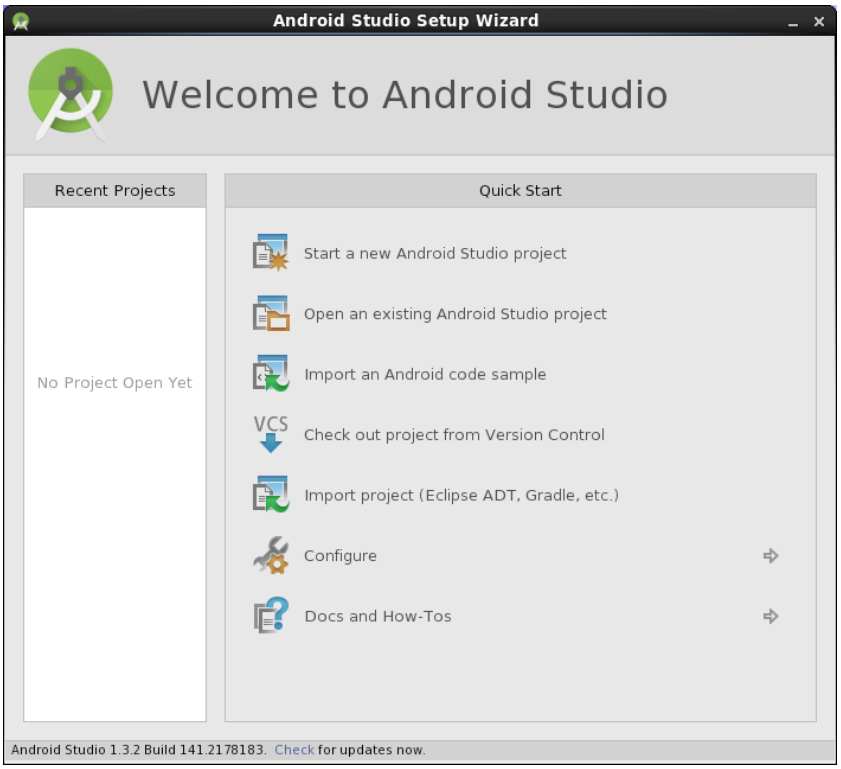

Figura 2: Solicitar la creación de un proyecto nuevo

1. Al finalizar el arranque de AndroidStudio aparecerá un diálogo que permitirá entre, otras opciones, crear un proyecto nuevo haciendo clic sobre `Start a new Android Studio project` (ver Figura 2).

2. En el formulario que os aparece debéis rellenar los valores tal y como se ven en la Figura 3:

   - El nombre de la aplicación.
   - El dominio de la empresa/organización que desarrolla el software. A partir de este dominio y el nombre de la aplicación se construye el paquete que incluirá nuestras clases. Notad que hemos usado un dominio razonablemente adecuado a las convenciones habituales de nombrado de paquetes de Java; podéis poner otro si queréis.
   - La ubicación del proyecto. Por defecto, AndroidStudio os sugerirá una ubicación en $HOME/AndroidStudioProjects. Debéis ser conscientes de que esta

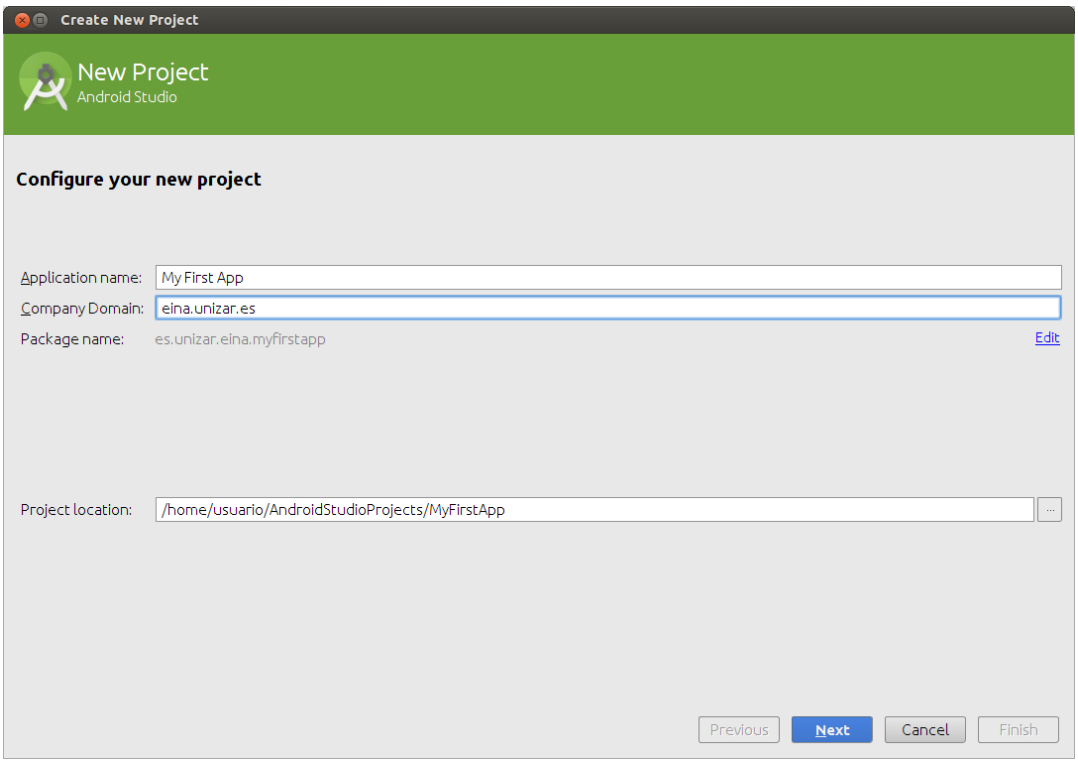

Figura 3: Dialogo de creación de un proyecto nuevo

carpeta puede ser eliminada una vez finalizada la sesión de prácticas. Al finalizar vuestra sesión prácticas se recomienda exportar el proyecto, y eliminar la carpeta $HOME/AndroidStudioProjects. En la siguiente sesión de prácticas deberíais importar el proyecto previamente exportado si queréis seguir trabajando en él.

A continuación haced clic en `Next`.

3. En el siguiente diálogo se puede configurar el SDK mínimo que requerirá la aplicación. Aquí generalmente se quiere el más antiguo que soporte todo lo que usa nuestra aplicación, para funcionar en un número mayor de dispositivos. Para el caso de la práctica nos da igual el que pongamos.

4. A continuación, hay que elegir si queréis crear una `Activity`, y en qué plantilla la queréis basar. Aseguraos de que está seleccionada `Blank Activity` y haced clic en `Next`. Para entender qué es una actividad leed el Apéndice A.

5. Entonces aparecerá la ventana de configuración de la Activity que vamos a crear (ver Figura 4). Dejad todos los valores por defecto y haced clic en `Finish`.

6. Finalmente, se abrirá la ventana del entorno de desarrollo. Es probable que el entorno os avise de que la versión de Java por defecto no es la adecuada. Simplemente hay que seleccionar Java 7 entre las opciones ofrecidas.

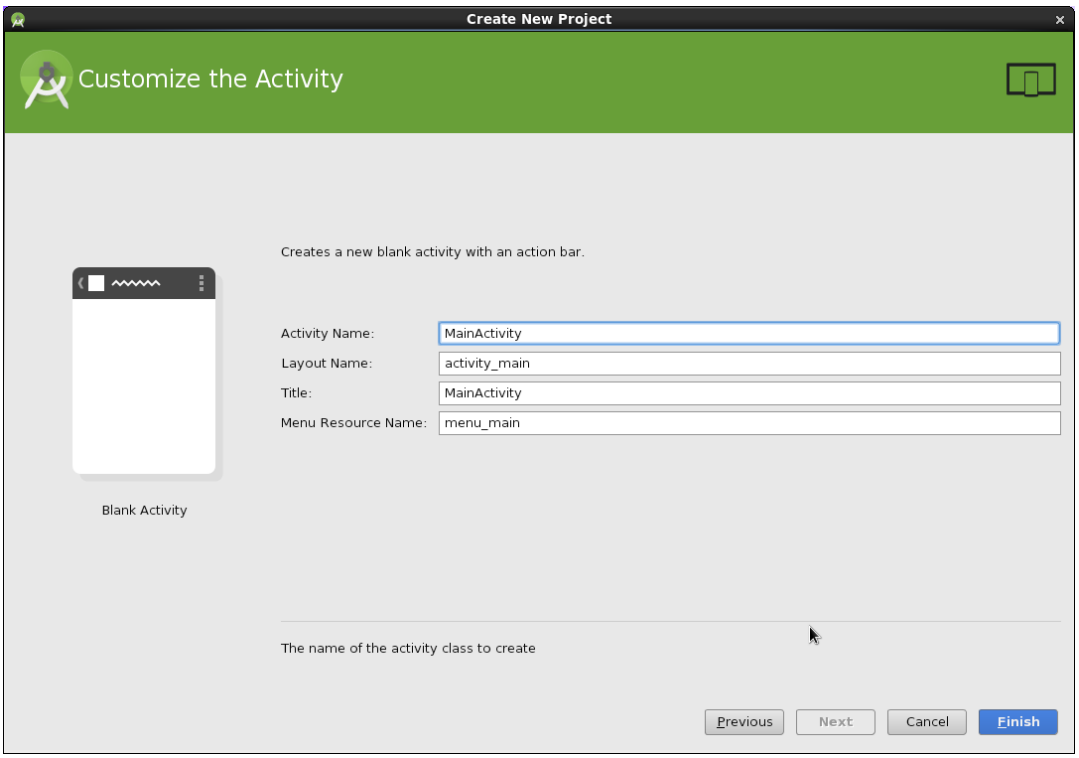

Figura 4: Dialogo de configuración de una actividad

3.2. Ejecutar la aplicación de demostración

El ejemplo de aplicación se puede ejecutar para poder comprobar que todo funciona correctamente. Esta sección describe cómo probarla en el emulador que viene con el SDK. Si tenéis un móvil o tableta con Android también podréis instalarla y probarla allí.

Si miráis el explorador del contenido de un proyecto Android (ver zona izquierda Figura 5), veréis que el proyecto que habéis creado tiene varias carpetas. La carpeta `java` contiene los fuentes de la aplicación. La carpeta `res` tiene distintos recursos (*resources*) necesarios para la aplicación, como imágenes, ficheros de *layout* (la GUI de la aplicación), cadenas de texto[2], etc.

Para ejecutar la aplicación en el emulador que viene con el SDK, deberéis seguir los siguientes pasos:

1. Primero tendréis que crear un dispositivo virtual Android (AVD). Para ello abrid el *AVD manager* seleccionando `Tools ->Android ->AVD Manager` (o buscad el icono en la barra de herramientas).

[2]En casi cualquier aplicación es generalmente una mala idea tener cadenas de texto dentro del código. Tenerlas separadas facilita chequear su ortografía o añadir soporte para nuevos idiomas.

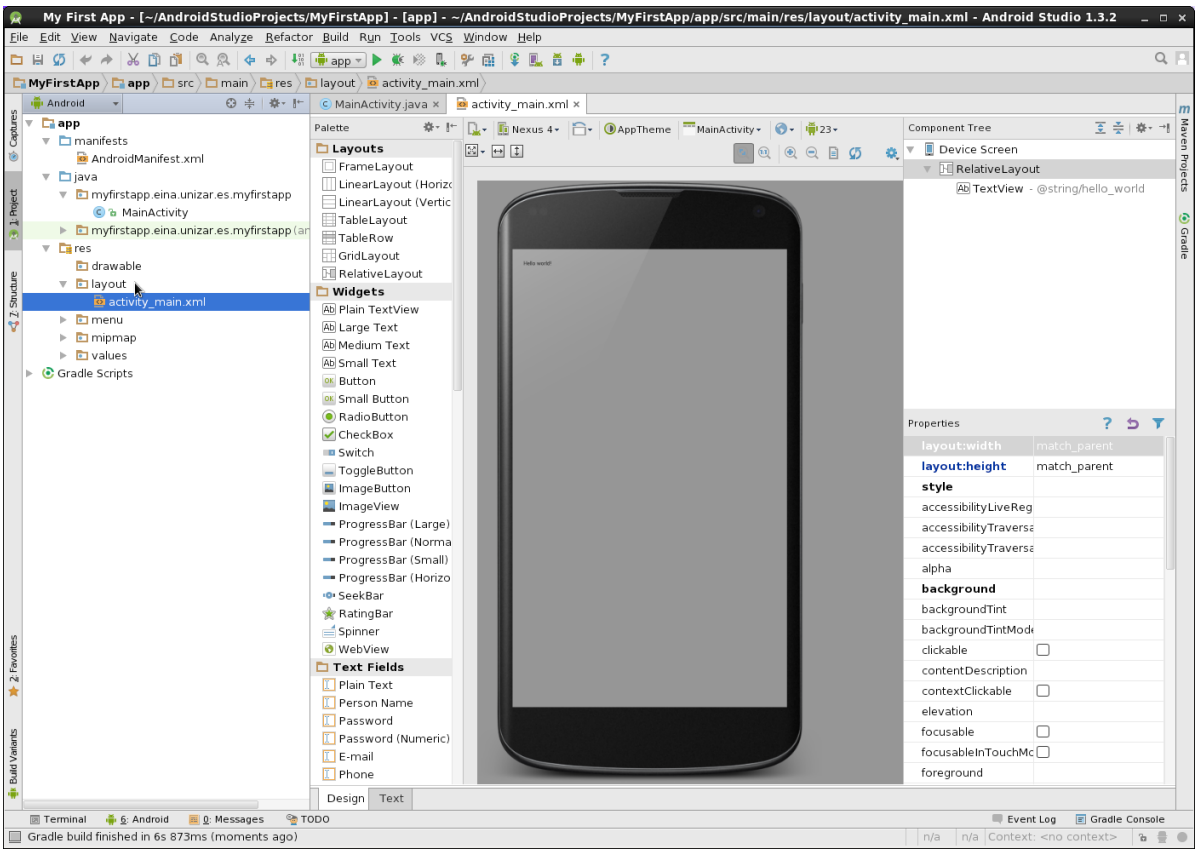

Figura 5: Exploración del contenido del proyecto

2. Haced clic en el botón `New` y rellenad los datos del dispositivo virtual, por ejemplo, como se muestra en la Figura 6: un dispositivo Nexus 4 con un nivel de API 23. Sobre todo, es importante que deshabilitéis la utilización de la unidad de procesamiento gráfico (`Use host GPU`) para evitar problemas en los equipos de los laboratorios. Además, es probable que tengáis problemas para copiar la imagen en disco del dispositivo.

   Finalmente, haced clic en `Finish`.

3. En el *AVD manager* seleccionad el AVD que acabáis de crear y haced clic en el icono verde de `Play` para lanzar el dispositivo en el emulador.

4. Esperad a que el dispositivo virtual arranque (puede tardar uno o dos minutos) y desbloquear la pantalla del mismo como haríais con un móvil. El dispositivo virtual arrancando se debe ver como el de la Figura 7 (salvo si habéis creado el vuestro con datos distintos a los sugeridos aquí). Podéis cerrar el *AVD manager*, pero no cerréis el dispositivo virtual.

5. Para ejecutar la aplicación por defecto, abrid uno de los ficheros de vuestro proyecto (por ejemplo `MainActivity.java` que estará dentro de la carpeta `java`, dentro

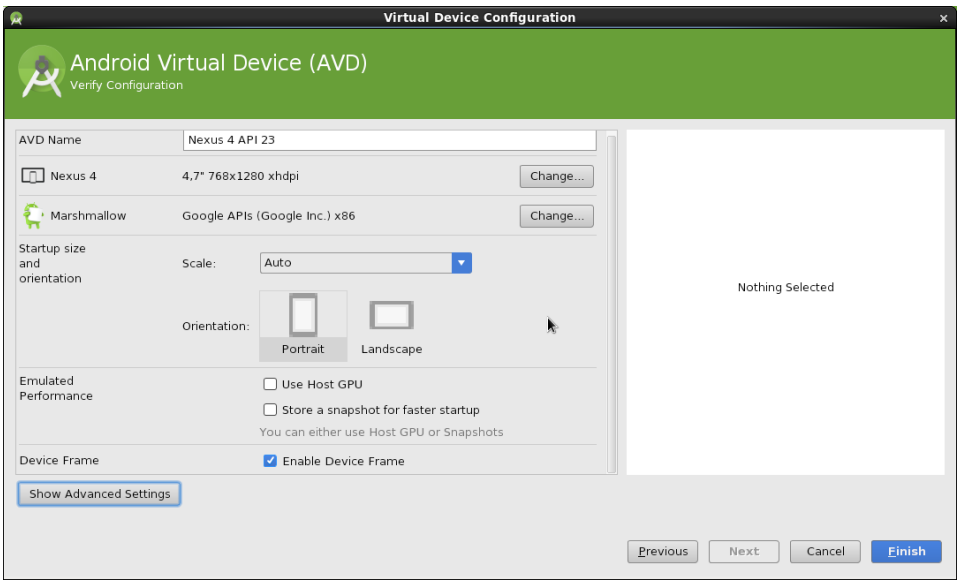

Figura 6: Creación de un dispositivo virtual Android

del paquete que habéis creado) y elegid `Run 'Main Activity'`, o usad el botón correspondiente de la barra de herramientas. Esto instalará y lanzará la aplicación en el dispositivo virtual mostrando el texto "Hello world!".

6. Si queréis probar la aplicación en un móvil real, hay varias opciones. El tutorial de Android señala una (conectando el dispositivo vía USB con el computador). Otra opción, igual de sencilla o más, y que funciona aunque no tengáis el cable apropiado, es coger el fichero de la aplicación y transferirlo a vuestro móvil via Internet, con una aplicación tipo Dropbox se hace en un momento, e instalarlo allí. La aplicación es el fichero con extensión `apk` que se crea cuando se construye en AndroidStudio (por ejemplo, al ejecutarla para verla en un dispositivo virtual). Este fichero se encuentra dentro de la carpeta del proyecto en la siguiente ubicación: `app/build/outputs/apk/app-debug.apk`, `$HOME/AndroidStudioProjects/MyFirstApp/app/build/outputs/apk/app-debug.apk` en el caso de este guión de prácticas. Copiad este fichero, luego navegad hasta una carpeta compartida con vuestro móvil y pegadlo allí. Si lo descargáis en el móvil con Dropbox, al terminar os saldrá un menú para que lo podáis instalar a través del "Instalador del paquete". Para que esta aplicación se pueda instalar y ejecutar, antes tendréis que haber ido a los Ajustes de Aplicaciones en vuestro móvil y marcar que se permite la instalación desde fuentes desconocidas (los detalles de esto difieren entre versiones de Android).

Figura 7: Dispositivo virtual Android funcionando

3.3. Manejo básico de la GUI

La GUI de las aplicaciones Android se construye a base de objetos de tipo `View` y `ViewGroup`. Los objetos `View` son elementos de la GUI como botones y campos de texto, mientras que los objetos `ViewGroup` son contenedores de objetos `View`, y de otros objetos `ViewGroup`, que determinan cómo deben organizarse los elementos de la GUI en la pantalla (cual es su `Layout`). Para construir la GUI de una aplicación Android hay que crear un fichero XML con los objetos de tipo `View`, `ViewGroup`, y sus descendientes, organizados con el `Layout` que se desee. Vais a crear una aplicación con un campo de texto y un botón, de modo que cuando se pulse el botón envíe el contenido del campo de texto a otra actividad.

1. Buscad el fichero /res/layout/activity_main.xml en AndroidStudio y abridlo. Por defecto os aparecerá un editor gráfico de este fichero, que os permitiría crear una GUI arrastrando y soltando componentes. En esta práctica queremos ver y modificar el XML directamente, así que buscad la pestaña activity_main.xml debajo del editor (estará junto a una que se llama Graphical Layout) y haced clic en ella para pasar a la vista de editor XML.

2. Sustituid el XML que aparece (es el de la aplicación “Hello world!” que acabáis de probar) con este:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
</LinearLayout>
```

Ahí definimos un `LinearLayout`, que es una subclase de `ViewGroup`, o sea un contenedor, que indica que sus componentes (hijos) deberán organizarse o bien en horizontal o bien en vertical, según el valor del atributo `android:orientation` (en este caso en horizontal). Estos hijos aparecerán en pantalla en el mismo orden en que los escribamos en el fichero XML (de momento no hemos añadido ninguno). Los otros atributos (`android:layout_width` y `android:layout_height`) son obligatorios en todos los objetos `View` y definen su tamaño. En este caso, como estamos definiendo la vista principal de la aplicación, queremos que ocupe toda la pantalla; por eso indicamos `match_parent` (que tome el mismo tamaño que su "padre" que en este caso será toda la pantalla).

3. Para añadir un campo de texto editable a la aplicación, añadid un objeto `EditText` dentro del `LinearLayout`. El XML a añadir es el siguiente:

```
<EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
```

   - El `android:id` proporciona un identificador único para el `EditText`, para luego poder referenciarlo fácilmente desde el código de la aplicación. La `@` es necesaria cuando se accede a un objeto recurso[3] desde XML; le sigue el tipo de recurso (`id`), una barra y luego el nombre del recurso (`edit_message`). El signo `+` sólo se pone cuando se define un recurso de tipo ID por primera vez; otras referencias posteriores a este recurso no necesitan el `+`, y otros tipos de recursos no lo necesitan nunca.
   - De nuevo aparecen los atributos `android:layout_width` y `android:layout_height`. En este caso su valor es `wrap_content`, lo que indica que el campo de texto editable será tan grande como necesite para su contenido, pero no más. Si hubiéramos puesto `match_parent`, como antes, ocuparía el mismo tamaño de su padre, o sea toda la pantalla.
   - El `android:hint` nos permite indicar el texto que saldrá en el campo cuando está vacío. En lugar de usar una cadena literal, indicamos un recurso (`@string/edit_message`) que ya definiremos en otro fichero.

[3] Un objeto recurso es un nombre asociado con un recurso de la aplicación (imagen, fichero de *layout*, cadena de texto...). Cada recurso tiene su correspondiente objeto recurso definido en el fichero `app/build/generated/source/r/.../R.java` en el proyecto. Este fichero no se debe editar a mano, es el SDK el que lo mantiene.

4. Todas las cadenas de texto de la interfaz de usuario deberían definirse como recursos, nunca como literales en el código. Por defecto, el proyecto Android incluye un fichero de recursos de tipo cadena en `res/values/strings.xml`. Abrid este fichero y editadlo para que quede así, o bien poned otros textos si lo preferís pero no cambiéis los nombres de los recursos (atributo `name`):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My First App</string>
    <string name="edit_message">Enter a message</string>
    <string name="button_send">Send</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

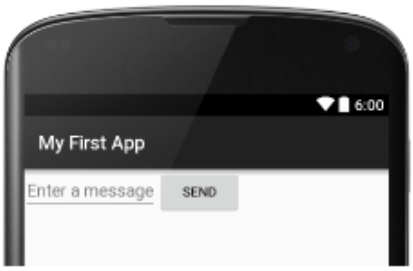

Figura 8: La caja de texto no ocupa todo el espacio libre horizontal

5. Añadid un botón al fichero de *layout*, justo después del elemento `EditText`:

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />
```

   Aunque podríamos darle un identificador, igual que al campo de texto, no lo hacemos porque no vamos a referirnos a él desde el código.

6. El *layout* no es exactamente como queremos. Si veis la Figura 8, el campo de texto va a ocupar lo necesario para mostrar su contenido, y el botón también, pero desaprovecharemos pantalla en horizontal. Sería mejor que el campo de texto ocupara todo el espacio disponible que no necesite el botón. Para ello, vamos a usar el campo `android:layout_weight` en el `LinearLayout`. El valor de peso es un número que dice cuanto del espacio libre debe usar cada `View`, teniendo en cuenta la cantidad de espacio que usan las `View` hermanas[4]. Si por ejemplo

[4]Recordad que tanto `EditText` como `Button` son subclases de `View`. Hermanas es en el sentido "hijas del mismo padre", es decir, que están contenidas dentro del mismo `ViewGroup` en el XML del *layout*; en nuestro caso, el `EditText` y el `Button` son vistas hermanas.

tenemos dos vistas hermanas, y a una le damos un peso de 2 y a la otra de 1, el espacio libre disponible[5] se repartirá así: 2/3 para la primera, 1/3 para la segunda. Si fueran tres vistas, y los pesos fueran 2, 2 y 1, se repartirían 2/5 para la primera, 2/5 para la segunda y 1/5 para la tercera. Por defecto, el peso de una vista es 0, así que si sólo se especifica el peso para una hermana, ésta hermana ocupará todo el espacio libre disponible. En nuestro caso, dadle un peso de 1 a la vista `EditText` (añadid el atributo `android:layout_weight="1"`) para que ocupe todo el espacio libre. Además, para mejorar algo la eficiencia de la aplicación, podéis decir que el ancho del `EditText` sea de 0, así no lo tendrá que calcular cada vez (que es lo que pasa cuando tiene el valor `wrap_content`) y luego volverlo a calcular para ocupar todo el espacio libre disponible; para ello añadid también el atributo `android:layout_width="0dp"`. El fichero de layout completo debería estar así:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />
</LinearLayout>
```

7. Ejecuta la aplicación para comprobar los resultados (recuerda que necesitarás tener abierto un dispositivo virtual, o bien que tendrás que copiar el fichero .apk a tu móvil o tableta).

3.4. Añadir una segunda actividad a la aplicación

Esta sección indica cómo ampliar la aplicación que habéis creado con una nueva actividad, que se mostrará al pulsar el botón y que contendrá el texto que aparezca en el campo de texto.

1. Para poder responder a la pulsación (*clic*) sobre el botón, abre el fichero activity_main.xml y añade el atributo `android:onClick="sendMessage"` al elemento

[5]Importante, el `weight` es para repartirse el espacio libre que queda disponible, no todo el espacio.

Button. El `sendMessage` es el nombre del método de la clase `MainActivity` que será llamado cuando el usuario pulse el botón.

2. Abre el fichero MainActivity.java y añade el método sendMessage:

```
/** Llamado cuando el usuario pulsa el botón Send */
public void sendMessage(View view) {
    // Hacer algo
}
```

En este punto, AndroidStudio mostrará un error *"Cannot resolve symbol View"*. Esto es porque hay que importar la clase `View` en `MainActivity`. Pulsad `Alt+Enter` en la señal del error en el editor (sugerencia en azul al pasar el ratón sobre la palabra View) y elegid la primera opción que os da AndroidStudio (*Import class...*). Para que funcione el método debe ser público, devolver `void` y tener `View` como único parámetro (que será el `View` que fue pulsado, en nuestro caso el botón).

3. Queremos que al pulsar el botón, se lea el campo de texto y se mande su contenido a otra actividad. Para ello hay que construir, dentro de `sendMessage`, un objeto de la clase `Intent` (que se usan para enlazar elementos de la aplicación en tiempo de ejecución y, típicamente, para lanzar actividades):

```
Intent intent = new Intent(this, DisplayMessageActivity.class);
```

Este *intent* pondrá en marcha una actividad de tipo `DisplayMessageActivity`. El primer parámetro del constructor debe ser un objeto `Context`[6] (`Activity` es una subclase de `Context`, luego podemos pasarle `this`), y el segundo la clase de componente de aplicación que queremos que reciba el objeto `intent` (en este caso la clase de actividad que se pondrá en marcha).

4. Un *intent* no sólo puede iniciar una actividad, sino pasarle además unos datos. Lo que hacemos es usar `findViewById` para acceder a `EditText`, de ahí sacamos su contenido y se lo pasamos a `intent`:

```
EditText editText = (EditText) findViewById(R.id.edit_message);
String message = editText.getText().toString();
intent.putExtra(EXTRA_MESSAGE, message);
```

Los extras son pares clave-valor que podemos asociar a un intent. Las claves son cadenas de texto, así que habrá que definir `EXTRA_MESSAGE` en nuestro código como una constante pública poniendo como primera línea de la clase MainActivity la siguiente:

```
public final static String EXTRA_MESSAGE = "es.unizar.eina.myfirstapp.MESSAGE";
```

Es recomendable que el nombre de esta clave no pueda colisionar con los que usen otras aplicaciones (los objetos de la clase `Intent` también se pueden usar para

[6]Una forma de acceder a información del entorno de una aplicación.

comunicarse con otras aplicaciones) así que es recomendable que lleve como prefijo el nombre de paquete de nuestra aplicación.

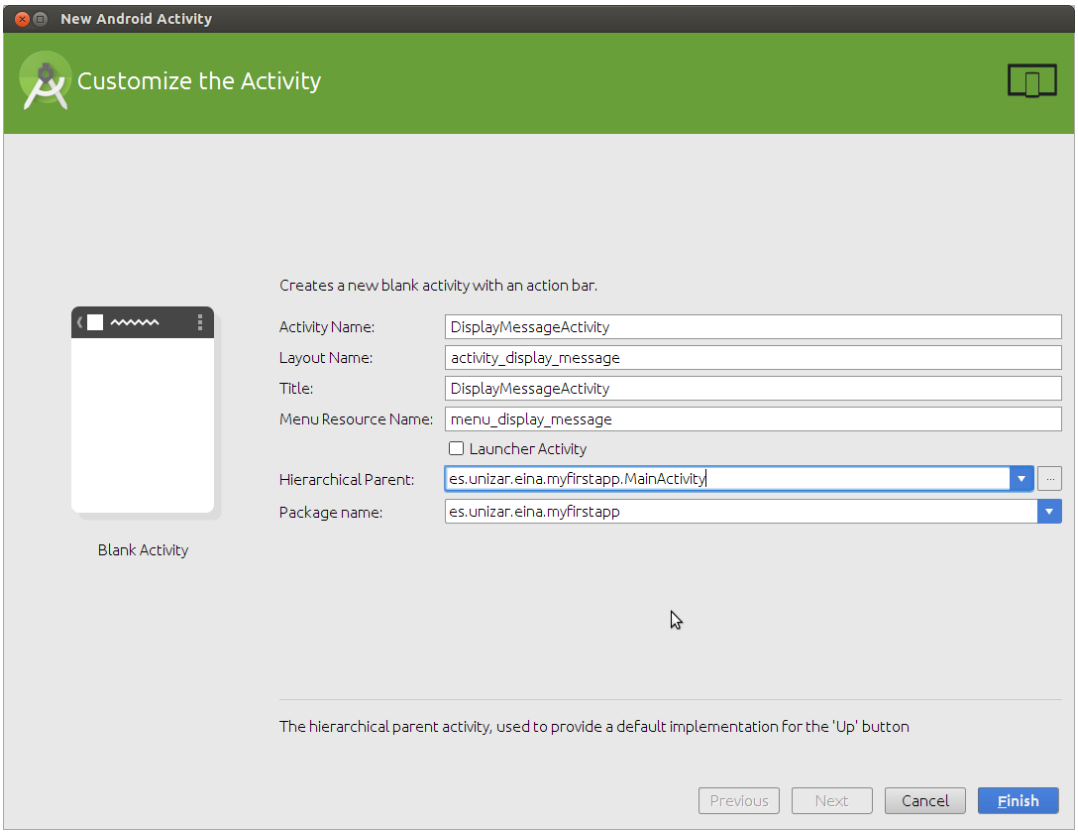

Figura 9: Creación de una nueva actividad en AndroidStudio

5. Para iniciar la nueva actividad basta con llamar al método `startActivity()` pasándole el *intent* que acabamos de crear. El código completo del método `sendMessage` sería:

```
public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
```

6. Y ahora hay que crear la clase para la nueva actividad para que todo funcione. Para ello en AndroidStudio id a `File ->New ->Activity ->Blank Activity` y rellenad los detalles de la actividad a crear: el nombre será `DisplayMessageActivity`, el nombre de su *layout* será `activity_display_message`, la madre jerárquica será `es.unizar.eina.myfirstapp.MainActivity` y el título puede ser Mi Mensaje. Lo

demás, dejadlo con los valores por defecto (ver Figura 9) y haced clic en `Finish`. Abrid `DisplayMessageActivity.java` y aseguraos de que la implementación del método `onCreate()` sea como esta:

```
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_display_message);
}
```

Toda actividad tiene que tener un método `onCreate`, al que se llama cuando se instancia la actividad. Cualquier otro método que AndroidStudio haya añadido a esta clase borradlo porque no nos hace falta.

7. Las aplicaciones de Android deben llevar un fichero AndroidManifest.xml con cierta información de las mismas, incluyendo todas sus actividades. En el Apéndice A tenéis una descripción del mismo. Al crear una nueva actividad, AndroidStudio ha añadido una entrada por defecto a este fichero como la siguiente:

```
<activity
  android:name=".DisplayMessageActivity"
  android:label="@string/title_activity_display_message" >
  <meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value="es.unizar.eina.myfirstapp.MainActivity" />
</activity>
```

Al decirle que la actividad madre de `DisplayMessageActivity` es `MainActivity` (en el elemento `meta-data`), Android puede implementar cierto comportamiento por defecto, como que cuando le demos al botón "atrás" desde `DisplayMessageActivity` vayamos a `MainActivity`.

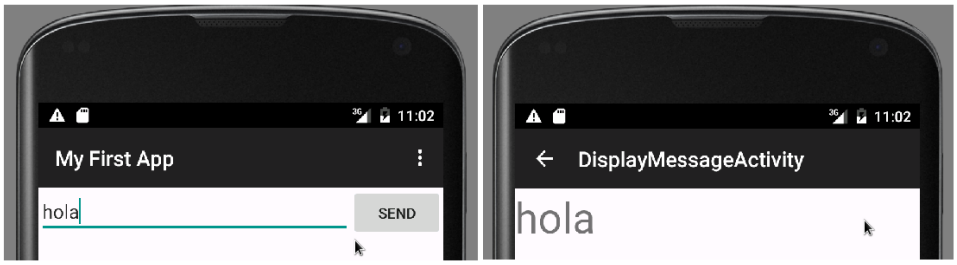

Figura 10: Resultado de la ejecución con la segunda actividad

8. Ahora que ya hemos implementado el comportamiento de enviar el texto al pulsar el botón, sólo nos falta hacer algo cuando se reciba. Toda actividad es invocada por un *intent* independientemente de cómo se ha navegado hasta ella, y se puede acceder a él usando el método `getIntent()`. Añadid el siguiente código al método `onCreate` de `DisplayMessageActivity` para acceder y obtener el mensaje que

hemos asociado al *intent*:

```
Intent intent = getIntent();
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

Para mostrar el mensaje que acabamos de recuperar, podemos crear un `TextView` y poner el texto en él. Después podemos pasar este `TextView` al *layout* de la actividad con el método `setContentView()`. El código de `onCreate` debería quedar así:

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   // Get the message from the intent
   Intent intent = getIntent();
   String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

   // Create the text view
   TextView textView = new TextView(this);
   textView.setTextSize(40);
   textView.setText(message);

   // Set the text view as the activity layout
   setContentView(textView);
}
```

Notad que para el *layout* de esta actividad no hemos escrito nada de XML, y todo lo hemos hecho desde código Java. Es una alternativa que para *layouts* sencillos es perfectamente válida pero que para otros más complejos puede ser desaconsejable.

9. Ahora ya puedes ejecutar la aplicación y comprobar que tras escribir algo en el campo de texto y pulsar el botón se abre una nueva actividad con este texto (ver Figura 10).

3.5. Depuración sencilla de aplicaciones Android

Si algo no funciona, una forma rápida y sencilla de depurar vuestra aplicación es usar el `logcat`. Para ello en AndroidStudio id a `View ->Tool Windows ->Android`. Esto mostrará en AndroidStudio una ventana en la parte inferior que incluye la pestaña `logcat` y que muestra los mensajes de depuración que escribe Android conforme interacciona con el dispositivo virtual (también se puede depurar sobre un dispositivo real, pero eso se sale del alcance del curso). Para escribir mensajes en `logcat`, y que así podáis ir comprobando cosas de vuestra aplicación, podéis utilizar este comando donde queráis en vuestro código:

```
android.util.Log.d("TAG", "Mensaje");
```

La primera cadena, TAG, debe ser algo que os permita localizar fácilmente vuestros mensajes en el `logcat`. Suele ser el nombre de la clase donde se genera el mensaje, pero podéis poner lo que queráis que os ayude a localizarlo. En la segunda cadena podéis poner el mensaje propiamente dicho (un simple aviso, o podéis poner allí el valor de alguna/s variable/s). El método `d` indica que queremos escribir el mensaje en la categoría de `DEBUG`, aunque hay otras (luego los mensajes se pueden filtrar por categoría en el `logcat` si hace falta). Esto debería ser suficiente para que podáis depurar cualquier aplicación sencilla.

4. Entrega de la práctica

Durante la sesión de prácticas se presentará al profesor el resultado de la misma, incluyendo un ejemplo de depuración de código. Recordad que para superar la asignatura hay que presentar todas las prácticas.

A. Fundamentos del desarrollo Android

Este Apéndice es un resumen de la información que aparece en `http://developer.android.com/guide/components/fundamentals.html` y se recomienda la lectura completa de dicha página web.

Las aplicaciones Android están escritas en java y formadas por uno o más componentes, que pueden ser actividades, servicios, proveedores de contenido y receptores de avisos/mensajes. Las herramientas del SDK de Android compilan el código java, junto con los datos y archivos de recursos en un archivo con un sufijo .apk, que es una aplicación que se puede instalar en un dispositivo Android.

El sistema operativo Android es un Linux multi-usuario donde cada aplicación se considera un usuario diferente. Por defecto, el sistema asigna a cada aplicación un único identificador de usuario. El sistema configura los permisos de todos los archivos de una aplicación para que sólo el identificador asignado a dicha aplicación pueda acceder a ellos. Además, cada proceso tiene su propia máquina virtual, por lo que el código de una aplicación se ejecuta en forma aislada de otras aplicaciones. Sin embargo, hay formas para que las aplicaciones puedan compartir datos y puedan acceder a los servicios del sistema.

A.1. Componentes de las aplicaciones Android

Son los bloques esenciales para construir aplicaciones. El sistema los utiliza para acceder a las aplicaciones. Los componentes son independientes entre sí y todos juntos definen el comportamiento de la aplicación.

Actividades Una actividad se corresponde con una pantalla de una aplicación Android. Por ejemplo, una aplicación de correo electrónico puede tener una actividad que muestra la lista de correos y otra actividad para componer un correo electrónico. Las actividades son independientes una de otra, pero en su conjunto definen una experiencia de usuario coherente, en el caso anterior para una aplicación de correo electrónico. Si la aplicación de correo electrónico lo permite, otra aplicación podrá iniciar cualquiera de esas dos actividades. Por ejemplo, una aplicación de cámara fotográfica puede iniciar la actividad de la aplicación de correo electrónico que compone un correo electrónico, con el fin de que el usuario envíe una imagen. Una actividad se implementará como una subclase de `Activity`. Leer `http://developer.android.com/intl/es/guide/components/activities.html`

Servicios Un servicio es un componente que se ejecuta en segundo plano para realizar o bien operaciones de larga duración o bien un trabajo para un proceso remoto. Un servicio a diferencia de una actividad no proporciona interfaz de usuario. Por ejemplo, un servicio puede reproducir música en segundo plano mientras el usuario está en una aplicación diferente, o puede obtener datos mediante la red sin el bloqueo que supone la interacción del usuario con una actividad. Las actividades pueden iniciar los servicios y dejarlos en ejecución o interactuar con ellos. Un servicio se implementará como una subclase de `Service`. Leer `http://developer.android.com/intl/es/guide/components/services.html`

Proveedores de contenidos Un proveedor de contenido es el encargado de gestionar los datos de una aplicación. Los datos se pueden almacenar en ficheros, bases de datos, en la web, o cualquier otro almacenamiento persistente. A través del proveedor de contenido, otras aplicaciones podrán consultar o incluso modificar los datos de mi aplicación si así lo permito. Por ejemplo, Android proporciona un proveedor de contenido que gestiona los contactos del usuario. Cualquier aplicación con los permisos adecuados puede leer y escribir información sobre un contacto en particular. Los proveedores de contenido también se usan para leer y escribir datos privados. Por ejemplo, la aplicación de Notas utilizará un proveedor de contenido para gestionar sus notas. Un proveedor de contenido se implementa como una subclase de `ContentProvider`. Leer `http://developer.android.com/intl/es/guide/topics/providers/content-providers.html`

Receptores de avisos o mensajes Este componente responde a avisos/mensajes que proporciona el sistema. Por ejemplo, avisos de que la pantalla se apaga, la batería está

baja, o se ha capturado una imagen. Las aplicaciones también pueden emitir avisos/-mensajes, por ejemplo para que otra aplicación sepa que se han descargado datos en el dispositivo y así pueda utilizarlos. Los receptores de avisos no tienen interfaz de usuario, pero se puede crear una barra de estado que alerte al usuario de los avisos. Sin embargo, es más común que un receptor de avisos sea sólo una "puerta de entrada" para otros componentes, por ejemplo, se podría iniciar un servicio para realizar un trabajo basado en eventos.

A.1.1. Activar componentes

A diferencia de los programas java, las aplicaciones Android no tienen un único punto de entrada (no existe el `main()` de java), cada componente es un punto de entrada.

Las actividades, servicios y receptores se activan mediante mensajes asíncronos (*intends*). Un *intend* enlaza en tiempo de ejecución componentes individuales (se puede pensar en ellos como mensajeros que solicitan una acción de otros componentes). Los componentes enlazados pueden pertenecer a diferentes aplicaciones. Un *intend* se crea con un objeto `Intent`, que define un mensaje para activar un componente específico o un *tipo* de componente: explícito o implícito, respectivamente.

Para las actividades y servicios, un *intend* define la acción a realizar. Por ejemplo, un *intend* podría transmitir una solicitud de una actividad para mostrar una imagen o abrir una página web. En algunos casos, un *intend* puede iniciar una actividad para recibir un resultado, en cuyo caso, la actividad también devuelve el resultado en un *intend* (por ejemplo, se puede emitir un *intend* para que el usuario elija un contacto personal y se puede devolver el resultado en un *intend* con el URI que apunta al contacto elegido).

Para los receptores de avisos/mensajes, un *intend* define el aviso que se está anunciando (por ejemplo, el aviso incluye sólo la frase "batería baja").

El componente proveedor de contenido no es activado por *intends* sino por una solicitud de un objeto `ContentResolver`. Este objeto gestiona al proveedor de contenido, de modo que el componente que está realizando peticiones sólo llama a métodos de la clase `ContentResolver`. Esto ofrece una capa de abstracción entre el proveedor de contenido y la información del componente solicitante.

Métodos para activar componentes:

- Las actividades se activan pasando un `Intent` al método `startActivity()` o `startActivityForResult()`.
- Los servicios se activan pasando un `Intent` al método `startService()`. La aplicación se puede unir a un servicio pasando un `Intent` al método `bindService()`.
- Los receptores de avisos/mensajes se activan pasando un `Intent` al método `sendBroadcast()`, `sendOrderedBroadcast()`, o `sendStickyBroadcast()`.

- Se puede realizar una pregunta a un proveedor de contenidos llamando al método `query()` de la clase `ContentResolver`.

A.2. El fichero de manifiesto

Se llama `AndroidManifest.xml` y está en el directorio raíz del proyecto. Para activar un componente, Android debe saber que éste existe leyéndolo en este fichero.

Además de la declaración de los componentes, el manifiesto hace más cosas:

- Identificar los permisos de usuario, como acceso a Internet o acceso de lectura a los contactos del usuario.
- Declarar el nivel de API mínima requerida por la aplicación.
- Declarar funciones de hardware y software utilizadas o requeridas por la aplicación, tales como una cámara o los servicios de Bluetooth.
- Declarar las API que la aplicación necesita aparte de las de Android, como por ejemplo la API de Google Maps.

Declarar componentes Se declaran en este orden: $activity, service, receiver, provider$. Las actividades, servicios y proveedores que estén en el código pero no en este fichero no se podrán utilizar. Sin embargo los receptores se pueden declarar en el fichero o crear dinámicamente con objetos `BroadcastReceiver` y registrarlos en el sistema llamando a `registerReceiver()`.

Declarar las capacidades de los componentes Como se mencionó anteriormente, algunos componentes se activan con *intends* nombrando explícitamente el componente destino. Sin embargo, el verdadero poder de los *intends* está en el concepto de *acción*. Sólo hay que decir el tipo de *acción* que se desea realizar (y, opcionalmente, los datos sobre los que actúa) y el sistema buscará en el dispositivo un componente que pueda realizar la *acción*. Si encuentra múltiples componentes, el usuario selecciona cuál de ellos utilizar.

El sistema identifica los componentes que pueden responder a una *acción* comparando el *intend* recibido con los *filtros intend* proporcionados en el fichero de manifiesto de las aplicaciones instaladas en el dispositivo.

Cuando se declara un componente en el manifiesto de la aplicación, si se desea, se pueden incluir *filtros intend* que declaran las capacidades del componente. Así se podrá responder a los *intend* de otras aplicaciones. Se puede declarar un filtro de intención para el componente mediante el elemento $< intent - filter >$ como hijo en la declaración del componente.

Declarar requisitos de la aplicación No todos los dispositivos Android ofrecen las mismas características. Es importante declarar las necesidades de nuestra aplicación para evitar que sea instalada en dispositivos que no la podrían ejecutar. Muchas de estas declaraciones son sólo informativas, el sistema no las lee, pero los servicios externos como Google Play sí, a fin de filtrar aplicaciones a los usuarios cuando las buscan.

Por ejemplo, si la aplicación requiere una cámara y utiliza la API introducidas en Android 2.1 (API Level 7), se deben declarar estos requisitos en el fichero de manifiesto. De esta forma, los dispositivos que no cuentan con una cámara y tienen una versión de Android inferior a 2.1 no podrán instalar la aplicación.

Sin embargo, también se puede declarar que la aplicación utiliza la cámara, pero no lo requiere. En ese caso, la aplicación realiza una comprobación en tiempo de ejecución para determinar si el dispositivo tiene cámara y deshabilitar cualquier característica que utilice la cámara.

Estas son algunas de las características importantes a tener en cuenta al diseñar y desarrollar una aplicación:

- Tamaño y densidad de la pantalla. Soportado por el elemento $< supports - screens >$.
- Mecanismos de entrada. Se debe declarar en $< uses - configuration >$ si se requiere joystick, trackball, etc.
- Dispositivos. Se debe declarar en $< uses - feature >$ si se requiere cámara, sensor de luz, giroscopio, una versión de OpenGL, etc.
- Versión de Android. Se debe declarar en $< uses - sdk >$ la versión mínima de la API de Android que se necesita.

Para saber más sobre cómo estructurar este fichero leed `http://developer.android.com/intl/es/guide/topics/manifest/manifest-intro.html`

A.3. Los recursos de la aplicación

Una aplicación Android además de código requiere recursos: imágenes, archivos de audio, y todo lo relativo a la presentación visual de la aplicación, por ejemplo, menús, estilos, colores y diseño de interfaces de usuario. Usando recursos es fácil actualizar las características de la aplicación sin modificar código, además la aplicación puede ofrecer alternativas de configuración (por ejemplo, idiomas, colores y tamaños de pantalla).

Cada recurso tiene un identificador único que lo referencia en la aplicación y en el fichero XML de otros recursos. Por ejemplo, si la aplicación contiene un archivo de imagen llamado logo.png (guardado en `res/drawable/directory`), se tendrá un ID denominado `R.drawable.logo`, que se puede utilizar para hacer referencia a la imagen

e insertarla en la interfaz de usuario.

Para saber más sobre recursos leed `http://developer.android.com/intl/es/guide/topics/resources/index.html`