Tema 2: Programación de aplicaciones para dispositivos móviles

¿Qué aprenderás?

- Crear proyectos en Android Studio y construir aplicaciones móviles.
- Usar recursos de tipo layout que permiten separar la capa de presentación de la capa lógica.
- Distinguir entre los distintos tipos de almacenamiento y elegir el más adecuado en cada situación.
- Utilizar distintas clases para establecer comunicaciones a través de Internet con servidores de datos externos.

¿Sabías que…?

- Un layout es un contenedor pensado para controlar la distribución, posición y tamaño de elementos gráficos que se colocan en su interior.
- Los widgets o elementos gráficos se utilizan para crear una interfaz gráfica que permita interactuar con el usuario, recogiendo datos y mostrando información en la pantalla del dispositivo.

2.1. Programación de aplicaciones móviles

Para programar aplicaciones para dispositivos móviles es necesario utilizar una serie de herramientas que nos permitan escribirla, depurarla, compilarla y ejecutarla. Este conjunto de herramientas se conoce como entorno de desarrollo integrado (IDE, *Integrated Development Environment*) y su función es proporcionar un marco de trabajo para que el programador pueda programar aplicaciones en un determinado lenguaje de programación.

La selección y utilización óptima del entorno de desarrollo es una decisión muy importante en el procedimiento de desarrollo de software. El entorno de desarrollo es la herramienta con la que el programador deberá trabajar durante la mayor parte de tiempo que dedique a la creación de nuevas aplicaciones.

Si el entorno de desarrollo es el más adecuado para un determinado lenguaje de programación y para el desarrollo de una aplicación determinada, y si el programador es conocedor de la mayoría de las funcionalidades y sabe aprovechar todas las facilidades que ofrece dicho entorno, se podrá optimizar el tiempo de desarrollo de software y facilitar la obtención de un producto de calidad en el menor tiempo posible.

2.2. Herramientas y fases de construcción

El primer paso para programar aplicaciones para dispositivos móviles que utilizan el sistema operativo Android es instalar el Android SDK (Software Development Kit) y preparar el entorno de desarrollo por primera vez.

Android Studio se ha convertido en el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android. Consta de un editor de código y diferentes herramientas para desarrolladores pensadas para aumentar la productividad, un sistema de ayuda para la construcción de interfaces gráficas de usuario (GUI), un editor de texto, un compilador y un depurador. Además, entre sus características destacan:

1. Un entorno unificado donde es posible desarrollar para todos los dispositivos Android.
2. Un sistema de compilación flexible basado en Gradle, herramienta que permite automatizar la creación de proyectos.
3. Un emulador rápido con múltiples opciones.
4. Funcionalidad para insertar cambios en el código y los recursos de la aplicación en ejecución sin reiniciar la aplicación.
5. Integración con GitHub.
6. Herramientas para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones.
7. Funcionalidades que facilitan la integración con otras herramientas Google, como por ejemplo Google Cloud Messaging y App Engine.

Un proyecto de Android Studio incluye uno o más módulos con archivos de código fuente y archivos de recursos. Por defecto, los archivos del proyecto se muestran en la vista de proyecto de Android, facilitando el rápido acceso a los archivos del proyecto.

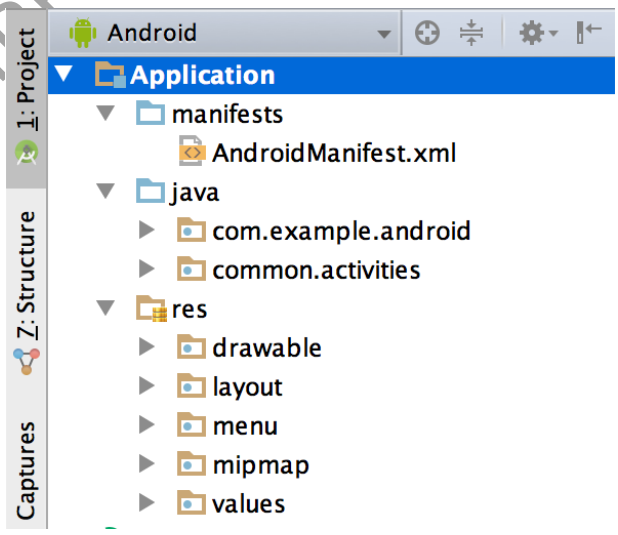


Figura: Estructura de un proyecto en Android Studio

La estructura típica de un proyecto contiene las siguientes carpetas:

- Carpeta manifests: Contiene el archivo AndroidManifest.xml.
- Carpeta java: Contiene los archivos de código fuente Java, incluido el código de prueba de JUnit.
- Carpeta res: Contiene todos los recursos sin código, como diseños XML, strings de IU e imágenes de mapa de bits.

En la imagen anterior, se aprecia el fichero **AndroidManifest.xml** que contiene información sobre la configuración del proyecto (número de versión, nombre de la Actividad principal, permisos que requiere la aplicación, etcétera). La carpeta **java** contiene los archivos de código fuente Java, incluido el código de prueba de JUnit.

Por su parte, la carpeta **res** contiene recursos como diseño de la interfaz en formato XML, cadenas de mensajes de entrada y salida (strings) o imágenes de mapa de bits, entre otros.

Android Studio usa Gradle como base del sistema de compilación. El archivo de compilación de Android Studio tiene el nombre build.gradle. Es un archivo de texto que utiliza la sintaxis Groovy para configurar la compilación del proyecto, buscar las dependencias y hacer que estén disponibles en tu compilación.

Android Studio también dispone de herramientas de depuración permiten revisar el código en la vista del depurador y verificar referencias, expresiones y valores de variables.

Finalmente, la herramienta Lint se utiliza para comprobar los archivos fuente del proyecto de Android en busca de posibles errores y realizar mejoras relacionadas con la seguridad, el rendimiento, la usabilidad, la accesibilidad y la internacionalización.

El siguiente video explica el proceso de creación de un proyecto y los primeros pasos en Android Studio.

Video: Primeros pasos en Android Studio
https://youtu.be/X7wvLtCw19k

2.3.Componentes básicos de una aplicación Android

Los componentes de una aplicación Android son los bloques esenciales que, combinados entre sí, sirven para construir una aplicación. Estos componentes son además el punto de entrada a la aplicación y son los siguientes:

- Actividades
- Fragmentos
- Servicios
- Proveedores de contenidos
- Receptores de emisiones
- Intents

Para mejorar el rendimiento del emulador, si nuestro microprocesador es compatible, se recomienda instalar el Intel® Hardware Accelerated Execution Manager siguiendo las instrucciones del siguiente enlace: Intel HAXM.

2.3.1. Actividades

Una Actividad (de la clase *Activity*) es cada una de las "pantallas" de una aplicación. Una definición más técnica, podría ser cada uno de los componentes de la aplicación que proveen al usuario una pantalla con la que puede interactuar.

Por ejemplo, una aplicación de mensajería instantánea puede tener una actividad que muestre la lista de contactos, otra para enviar mensajes y otra para leer mensajes recibidos. Si bien estas tres actividades trabajan juntas dentro de la aplicación de mensajería, cada una de ellas es independiente de las otras.

En Android, las aplicaciones pueden llamar a actividades de otras aplicaciones. Por ejemplo, la aplicación de mensajería anterior podría abrir una actividad de la aplicación de fotografía para enviar una foto junto con un mensaje. Del mismo modo, la aplicación de fotografía podría abrir la actividad de una aplicación de correo electrónico para enviar la foto como adjunto de un correo.

2.3.2. Fragmentos

Un fragmento es una porción de una actividad. Podremos tener varios fragmentos y modificarlos durante la ejecución de la aplicación con el fin de combinarlos según las necesidades.

Los Fragmentos aparecieron por primera vez en la versión de Android 3.0 Honeycomb, una versión de Android exclusiva para tabletas. Las mayores dimensiones de las pantallas en las tabletas permiten mostrar más información que en un teléfono móvil y por tanto los Fragmentos fueron la manera de poder combinar varias pantallas de información en una sola.

2.3.3. Servicios

Un Servicio es un componente que se ejecuta en el segundo plano para realizar tareas para trabajos remotos. Los servicios se ejecutan como cualquier otra aplicación, pero no tienen componentes de interfaz de usuario. Son útiles para realizar tareas durante un tiempo determinado como por ejemplo puede tomar datos del receptor GPS del teléfono.

Un servicio puede haber sido iniciado por otro componente, como por ejemplo una actividad, y continúan en ejecución independientemente del componente que haya hecho la llamada. El ciclo de vida de un Servicio es más sencillo que el de una Actividad y se implementa como clase derivada de la clase *Service*.

2.3.4. Proveedores de contenidos

Un Proveedor de contenidos (*Content Provider*) es una interfaz destinada a compartir datos entre las aplicaciones. Por defecto, Android ejecuta las aplicaciones de forma independiente, por lo que los datos de la aplicación están aislados del resto de aplicaciones del sistema. Si bien se pueden traspasar pequeñas cantidades de información entre las aplicaciones a través de los intentos (*Intents*), los Proveedores de contenidos están pensados para compartir un mayor volumen de datos entre aplicaciones.

La información que se comparte se puede insertar, actualizar, borrar y consultar. Por ejemplo, el Proveedor de contenidos de los Contactos proporciona la información de los contactos del teléfono a las diferentes aplicaciones. De este modo, una aplicación como WhatsApp que posea los permisos adecuados puede consultar los Contactos para leer y escribir información. Otro ejemplo: el Almacén de Medios se encarga de almacenar y servir diferentes tipos de datos para compartir, como fotos y música, entre las diferentes aplicaciones.

Un Proveedor de contenidos también puede ser útil en el caso de necesitar leer y escribir datos que son privados y no están compartidos. Por ejemplo, una aplicación de tipo editor de texto podría usar un Proveedor de contenidos para guardar notas.

Por otro lado, la separación entre el Almacén de Datos y la interfaz de usuario proporciona mucha flexibilidad a Android. Por ejemplo, se podría instalar una aplicación alternativa para ver los contactos, o una aplicación podría acceder desde el escritorio a algunos de los datos de configuración del sistema (que también están almacenados en un proveedor de contenidos) para modificar algunas funcionalidades, como la de activar y desactivar el WiFi. Los Proveedores de contenidos tienen interfaces sencillas con métodos para insertar, actualizar, borrar y consultar la información, de manera parecida a los métodos de acceso a bases de datos.

Un Proveedor de contenidos está implementado como una subclase de *ContentProvider* y debe implementar una API que permita a otras aplicaciones interactuar.

2.3.5. Receptores de emisiones

Los Receptores de emisiones (Broadcast Rreceivers) son un tipo de componente que responde a mensajes del sistema dirigidos a todas las aplicaciones del dispositivo. Podríamos considerarlo como una especie de sistema de suscripción de eventos en Android, es decir, el receptor simplemente se queda en estado latente esperando a activarse cuando ocurra un evento al que está suscrito.

El sistema Android está emitiendo mensajes constantemente. Por ejemplo, cuando se recibe una llamada, se apaga la pantalla o bien la batería tiene un nivel de carga bajo. Todos estos eventos son controlados y emitidos por el sistema. Por su parte, también las aplicaciones pueden emitir mensajes, por ejemplo, cuando una aplicación anuncia que se han terminado de descargar algunos datos y que estos ya están disponibles para ser usados.

Los Receptores de emisiones no tienen una interfaz visual, pero cuando se lanzan ejecutan una parte de código (como poner en marcha una actividad o servicio) y pueden crear una notificación en la barra de estado para alertar al usuario cuando ha ocurrido un evento.

Un Receptor de emisiones se implementa como una subclase de *BroadcastReceiver* y cada mensaje que emite se envía como un objeto de la clase *Intent*.

2.3.6. Intents

La mayor parte de los componentes de las aplicaciones que acabamos de ver se comunican a través de mensajes asíncronos llamados *Intents*. Estos mensajes se crean como objetos de la clase *Intent* y su comportamiento depende del componente que lo utiliza.

En las Actividades y Servicios, un intent define una acción a realizar (por ejemplo, "ver" o "enviar" algo) y puede especificar la URI de los datos sobre las que se debe actuar. Por ejemplo, un intent puede enviar una petición de abrir una página web y especificar la dirección de la web, para que la actividad correspondiente pueda abrir la página correctamente.

En otros casos se puede comenzar una actividad y esperar a recibir un resultado. Por ejemplo, se podría iniciar una actividad para seleccionar una persona de la agenda de contactos del teléfono y la actividad devolvería el resultado en un intent (éste incluiría la URI apuntando al contacto seleccionado).

En el caso de los Receptores de emisiones, un intent simplemente define el mensaje que se está emitiendo; por ejemplo, un mensaje para indicar que el nivel de batería del dispositivo es bajo incluiría únicamente una cadena de texto con un mensaje.

2.4. Interfaz de usuario y clases asociadas

La programación de una intefície gráfica de usuario (GUI) en el lenguaje de programación Java se realiza utilizando librerías especializadas: se programa íntegramente usando las clases de la API correspondiente. Es un proceso tedioso que dificulta además la separación entre el diseño gráfico y la programación de la interfaz.

En Android, si bien es posible mantener este enfoque a la hora de trabajar con interfaces gráficas, se recomienda definir totalmente la interfaz de una Actividad a partir de lo que se conoce como Layout. Un Layout es un archivo XML que utiliza una estructura predefinida para definir cómo se verá la aplicación en pantalla.

Desde el código fuente en Java podemos decirle a una Actividad que utilice un Layout concreto, y de forma automática se visualizarán los elementos gráficos necesarios respetando la disposición definida en el Layout. De este modo, la Actividad sólo debe responsabilizarse de gestionar los eventos (es decir, la interacción con el usuario) y no la creación de la interfaz.

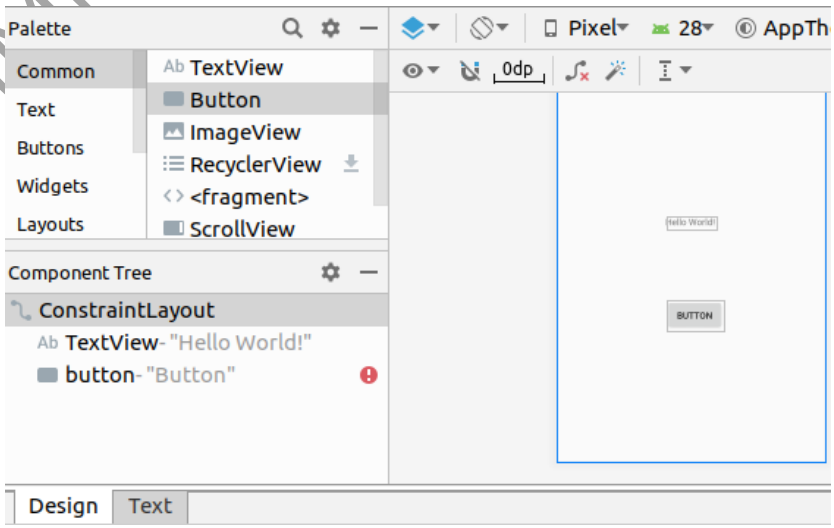
Un ejemplo del código fuente Java que asigna un Layout a una Actividad sería el siguiente:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.miActivityLayout);
   }
}
```

En el ejemplo anterior, el método *setContentView* acepta como parámetro un identificador de recurso correspondiente a un archivo XML con una descripción de interfaz gráfica. El Layout actúa como contenedor de los elementos gráficos que son la base del desarrollo de interfaces gráficas en Android. Todos los elementos gráficos, llamados widgets, son subclases de *View*.

Usar recursos de tipo Layout permite separar la capa de presentación de la capa lógica, proporcionando flexibilidad para cambiar la interfaz gráfica sin necesidad líneas de código. En el ejemplo anterior, el mismo código podría cargar cualquier Layout que hayamos previamente creado en el lugar de *miActivityLayout* y la aplicación ofrecería una interfaz de usuario diferente.

Android Studio nos facilita la creación de layouts con formato XML y que se guardan en la carpeta **res/layout/** del proyecto. En el IDE se nos muestran dos pestañas, una en modo diseño y otra en modo texto, que muestran respectivamente el contenido del archivo XML y la interpretación gráfica del mismo.



Creación de layouts en Android Studio

2.4.1. Widgets

La interfaz de usuario de una aplicación Android se construye utilizando objetos del tipo ViewGroup y View. Por un lado, la clase android.view.ViewGroup sirve de clase base para las subclases de tipo layout, que proporcionan los diferentes tipos de disposiciones de los elementos. Por otro lado, los objetos de tipo vistas son la unidad básica de una interfaz de usuario ya que heredan de la clase android.view.View que sirve de clase base para todas las subclases llamadas widgets, que como ya hemos visto son objetos de interfaz de usuario totalmente implementados, como por ejemplo campos de texto y botones.

La siguiente tabla muestra algunos de los widgets de uso más habitual en las aplicaciones Android. Todos estos widgets están disponibles para arrastrar y soltar desde la Paleta del editor gráfico de layouts.

Componente	Descripción
Botón	Clase *Button*. Este es uno de los widgets más utilizados. Puede ser pulsado por el usuario y lanzar una determinada acción.
Caja de texto	Clase *EditText*. Es un campo de texto editable que permite al usuario introducir datos.
Casilla de verificación	Clase *CheckBox*. Widget que ofrece dos estados (marcado/desmarcado) y permite al usuario seleccionar una o más opciones.
Casilla de selección	Clase *RadioButton*. Similar a una casilla de verificación, con la diferencia de que el usuario únicamente puede seleccionar una opción.
Lista desplegable	Clase *ListView*. Muestra verticalmente una lista de opciones.

Tabla: Widgets más habituales en Android Studio

El siguiente video explica (divido en dos partes) el proceso de creación de una aplicación en Android Studio.

Video: Primera App en Android Studio parte 1

Video: Primera App en Android Studio parte 2

2.4.2. Recursos adicionales de un proyecto

Además del código de la aplicación y los layouts que conforman la interfaz de usuario, un proyecto Android también puede constar de una serie de recursos como imágenes, iconos y archivos multimedia (principalmente archivos de audio y vídeo). Estos recursos adicionales facilitan la actualización de numerosas características de la aplicación sin cambiar el código y permiten optimizar una aplicación para diferentes idiomas y tamaños de pantalla.

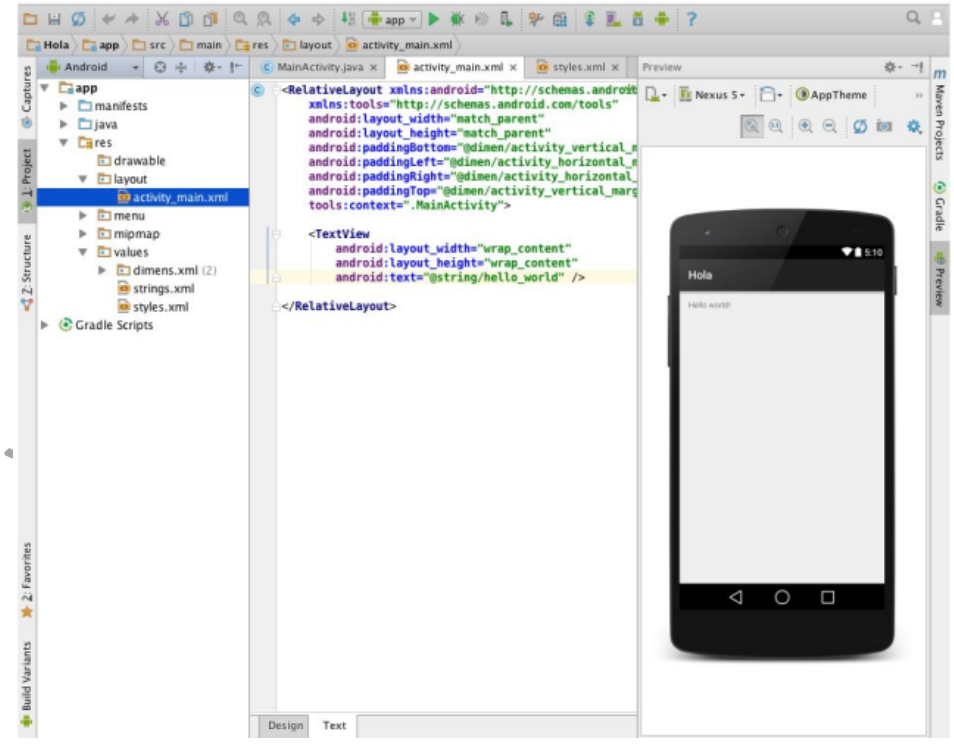
Cada recurso de un proyecto de Android tiene asignado un ID con número entero único que se puede usar para hacer referencia al recurso desde el código Java de la aplicación. Por ejemplo, si la aplicación contiene un archivo de imagen denominado icono.jpg en el directorio del proyecto **res/drawable/**, este recurso puede ser referenciado en el código como *R.drawable.icono*.

Entre los recursos más habituales se encuentran las cadenas de texto utilizadas en la interfaz de usuario. De esta forma, estas cadenas de texto se pueden traducir a otros idiomas y guardarlas en archivos independientes. En tiempo de ejecución, Android utilizará las cadenas de idioma apropiadas según la configuración de idioma del usuario, en función de un calificador de idioma anexado al nombre del directorio de recursos; por ejemplo, el directorio **res/values-es/** hace referencia a los valores de las cadenas de texto español.

2.5. Contexto gráfico

Una de las mayores ventajas que ofrece Android Studio es la posibilidad de crear la interfaz de usuario de manera dinámica, a medida que desarrollamos el código fuente.

El AVD (Dispositivo Virtual Android) es un emulador de un dispositivo real que permite al programador ejecutar programas sin requerir un dispositivo físico y comprobar cuál será el comportamiento de la aplicación que se está desarrollando. Se pueden crear tantos dispositivos AVD como sean necesarios, con configuraciones diferentes en lo relativo a sistema operativo, tamaños de pantalla, versión de la API, configuración del teclado o memoria interna, entre otras muchas características.



Creación de layouts en Android Studio

Los componentes gráficos de la aplicación se guardan en el fichero R.java junto al resto de recursos del proyecto; mediante un identificador único, se accederá a esos componentes gráficos para recoger entradas de datos o bien mostrar información al usuario.

2.5.1. Enlaces de datos

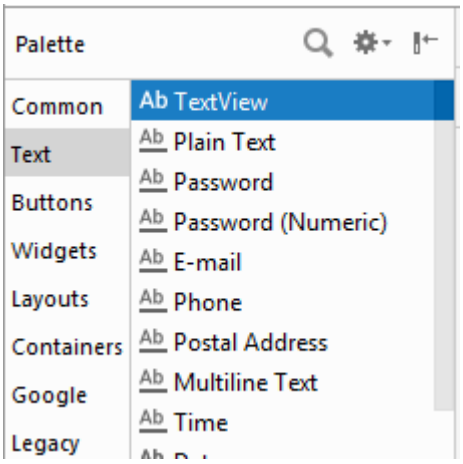
El *Data Binding* o enlace de datos es un mecanismo que permite enlazar los elementos de la interfaz de usuario con el código de una aplicación. Podemos decir que es el enlace entre el *frontend* y el *backend* de una aplicación, independientemente del lenguaje de programación que se esté usando.

En Android, el método *findViewById()* se utiliza para buscar un componente gráfico (objetos que heredan de la clase View) por su identificador dentro del layout. Como ya se ha dicho, todos los controles se encuentran en el fichero R.java generado automáticamente por el compilador.

2.5.2. Etiquetas de texto (TextView)

Una etiqueta de texto es un componente gráfico que permite mostrar un texto en la interfaz de usuario. Por lo general, el texto a mostrar se define en tiempo de diseño asignando un valor al atributo *text*. También es posible definir el valor del texto a mostrar en tiempo de ejecución mediante el método *setText()*.

En el siguiente ejemplo, en primer lugar, se define la variable tvMensaje del tipo TextView. A continuación, se utiliza el método *findViewById()* para enlazar con un TextView; una vez se ha enlazado con el componente, se utiliza el método *setText()* para asignar un valor y que muestre un mensaje en pantalla.

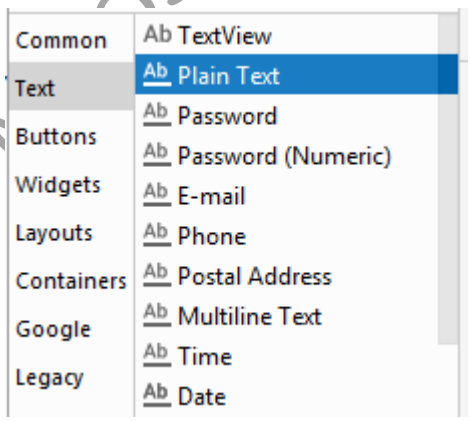


```
public class MainActivity extends AppCompatActivity {
    // 1. declaración de variables
    private TextView tvMensaje;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //2. Data binding: enlace de datos
        tvMensaje = findViewById(R.id.textView);
        //3. Uso del componente gráfico
        tvMensaje.setText("Hola");
    }
}
```

2.5.3. Campos de texto editables (EditText)

Los campos de texto editables se utilizan principalmente para recoger una entrada de datos por parte del usuario. Existen diversos tipos de campo de texto, en función del tipo de datos que se espera que entre el usuario. Los tipos de entrada de datos más habituales están especificados en la Paleta del modo Diseño de Android Studio: texto plano, contraseña, correo electrónico, teléfono, dirección postal, campo con varias líneas de texto, hora y fecha.



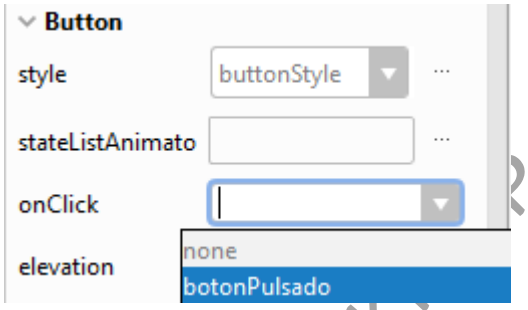
También puede utilizarse un tipo genérico para recoger un dato de tipo String, que posteriormente puede ser analizado y convertido a un tipo de datos diferente. En el siguiente ejemplo, se solicita el nombre al usuario, se recoge el valor de entrada en un campo de texto y a continuación ese valor se asigna a una variable.

```
public class MainActivity extends AppCompatActivity {
    // 1. declaración de variables
    private EditText etNombre;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //2. Data binding: enlace de datos
        etNombre = findViewById(R.id.etCampoNombre);
        //3. Uso del valor recogido en el componente gráfico
        String nombre = etNombre.getText().toString();
    }
}
```

2.5.4. Botones (Button)

La clase Button es la encargada de representar botones en la interfaz de usuario. El principal método de un componente Button es el método *OnClick()* que permite capturar el evento cuando el usuario pulsa sobre el botón. Android Studio permite asociar de manera sencilla un método Java al evento *onClick()*: simplemente hay que seleccionar el nombre del método en el desplegable del atributo onClick. El método seleccionado debe declararse como **public** y tener como parámetro un objeto de la clase View.



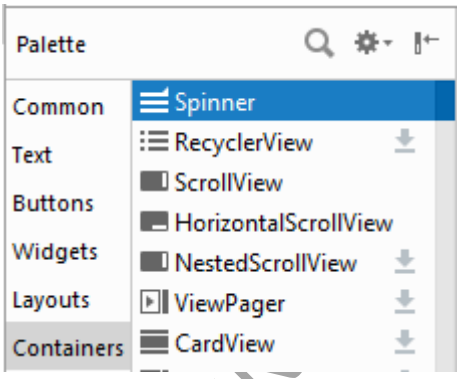
Otra forma de capturar la pulsación de un botón consiste en asociar al botón una interfaz del tipo OnClickListener. A través del método setOnClickListener() asociamos la interfaz OnClickListener que debe implementar el método onClick() tal y como se muestra a continuación.

```
public class MainActivity extends AppCompatActivity {
    // 1. declaración de variables
    private Button miBoton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //2. Data binding: enlace de datos
        miBoton = findViewById(R.id.button);
        //3. Añadir onClickListener al botón
        miBoton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // llamada a un método
                respoderPulsacion();
            }
        });
    }
    public void respoderPulsacion() {
        Toast.makeText(MainActivity.this,
             "Botón pulsado", Toast.LENGTH_LONG).show();
    }
}
```

2.5.5. Control de selección (Spinner)

El control Spinner permite al usuario seleccionar un valor de entre una lista de valores propuestos. Los valores del Spinner se definen en el fichero strings.xmlReceptores de emisiones (Broadcast Rreceivers) son un tipo de componente que responde a mensajes del sistema dirigidos a todas las aplicaciones del dispositivo.



Los valores que muestra el control de selección provienen de un adaptador. Un adaptador representa el modelo de datos: el conjunto de datos que el control de selección mostrará. Además, el adaptador proporciona la vista: la manera en la que se presentarán los datos dentro del control de selección. Esta vista puede ser una simple lista de cadenas de texto, imágenes o una combinación de las dos anteriores.Android proporciona varios tipos de adaptadores. El *ArrayAdapter* es un adaptador sencillo que provee de datos a un control de selección a partir de un array de objetos de cualquier tipo. A continuación, se muestra un ejemplo de su uso.

```
public class MainActivity extends AppCompatActivity {
    // 1. declaración de variables
    private Spinner listaValores;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //2. Data binding: enlace de datos
        listaValores = findViewById(R.id.spinner);
        //3. Adaptador (datos) del Spinner
        String[] ciudades = {"París","Londres","Berlín"};
        ArrayAdapter<String> adaptador = new ArrayAdapter<String>
              (this,android.R.layout.simple_spinner_item, ciudades);
        listaValores.setAdapter(adaptador);
    }
}
```

En este ejemplo, se utiliza un array de objetos del tipo String llamado **ciudades** con los valores que mostrará el control de selección. El adaptador carga los valores del array anterior. El método setAdapter() asigna el adaptador al control de selección.

Recursos y enlaces

- Guías para desarrolladores de Android

- Diseña para Android

- Principios de diseño de las aplicaciones para dispositivos móviles

Conceptos clave

- **Actividad**: Una Actividad representa una pantalla individual con una interfaz de usuario.
- **Archivo de manifiesto**: Una aplicación debe declarar todos sus componentes en este archivo, que debe encontrarse en la raíz de directorios del proyecto de la aplicación.
- **Componentes**: Los componentes de una aplicación son puntos de entrada por los que el sistema o un usuario entran en una aplicación. Las Actividades son el tipo de componente más habitual.
- **Layout:** Elemento no visual destinado a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior.
- **Proyecto**: Jerarquía donde se ubican todos los archivos de código fuente Java, los recursos, los ficheros de configuración y los ficheros de construcción Gradle.
- **Servicio:** Componente que permite mantener la ejecución de la aplicación en segundo plano.

Test de autoevaluación

1. ¿Cómo podríamos definir un layout?
   a) Un contenedor pensado para controlar la distribución, posición y tamaño de elementos gráficos.
   b) Una herramienta utilizada en el diseño de una interfaz gráfica.
   c) El conjunto de recursos gráficos y multimedia de una aplicación móvil.
   d) Todas las anteriores son correctas.

2. ¿De qué clase hereda un objeto de tipo Button?
   a) android.view.Widget.
   b) android.view.Graphics.
   c) android.view.ViewGroup.
   d) android.view.View.

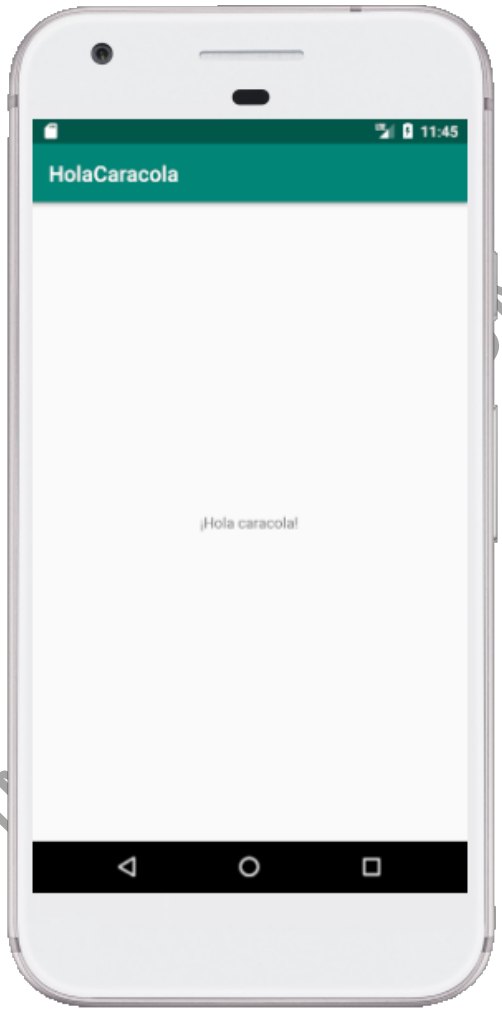
3. ¿En qué carpeta del proyecto se guardan los recursos adicionales como imágenes y vídeos?
   a) media.
   b) misc.
   c) res.
   d) Ninguna de las respuestas anteriores es correcta.

4. ¿Qué herramienta utiliza Android Studio para buscar dependencias y facilitar la construcción del proyecto?
   a) CMake.
   b) SCons.
   c) Rake.
   d) Gradle.

Ponlo en práctica

Actividad 1

Crea un nuevo proyecto en Android Studio que muestre el mensaje “¡Hola caracola!” en la pantalla del dispositivo tal y como se muestra en la imagen siguiente.



Solucionarios

Test de autoevaluación

1. ¿Cómo podríamos definir un layout?
   a) **Un contenedor pensado para controlar la distribución, posición y tamaño de elementos gráficos.**
   b) Una herramienta utilizada en el diseño de una interfaz gráfica.
   c) El conjunto de recursos gráficos y multimedia de una aplicación móvil.
   d) Todas las anteriores son correctas.

2. ¿De qué clase hereda un objeto de tipo Button?
   a) android.view.Widget.
   b) android.view.Graphics.
   c) android.view.ViewGroup.
   d) **android.view.View.**

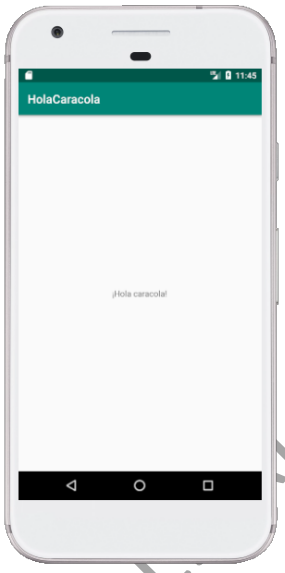
3. ¿En qué carpeta del proyecto se guardan los recursos adicionales como imágenes y vídeos?
   a) media.
   b) misc.
   c) **res.**
   d) Ninguna de las respuestas anteriores es correcta.

4. ¿Qué herramienta utiliza Android Studio para buscar dependencias y facilitar la construcción del proyecto?
   a) CMake.
   b) SCons.
   c) Rake.
   d) **Gradle.**

Ponlo en práctica

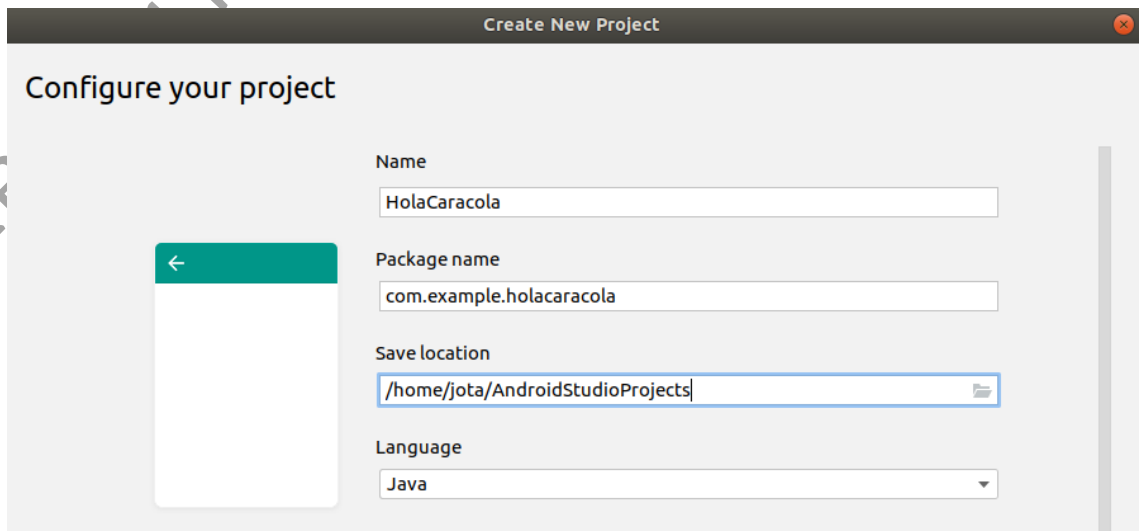
Actividad 1

Crea un nuevo proyecto en Android Studio que muestre el mensaje “¡Hola caracola!” en la pantalla del dispositivo tal y como se muestra en la imagen siguiente.

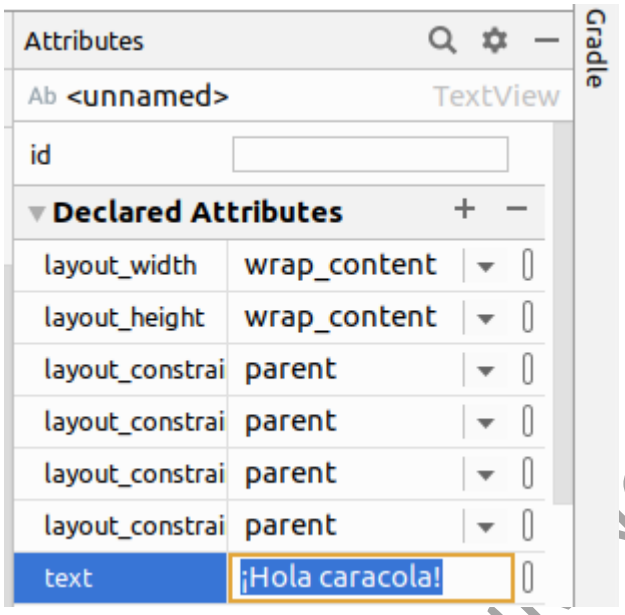


Solución:

Una vez descargado e instalado Android Studio (consulta la documentación del Tema 1 para realizar estos pasos), inicia un nuevo proyecto en blanco. El nuevo proyecto requiere un nombre y una versión de la API de Android; puedes utilizar la información de la imagen.



Una vez se ha creado el proyecto, utiliza el Editor de Diseños para modificar el valor del atributo *text* del widget TextView del layout; los atributos se encuentran en la parte derecha del editor.



1. **Haz doble clic sobre el valor del atributo *text***
2. **Teclea el texto ¡Hola caracola! y a continuación pulsa Intro.**
3. **El nuevo valor del atributo quedará reflejado en el TextView del layout.**

Finalmente, ejecuta el proyecto en un emulador o en tu propio dispositivo.