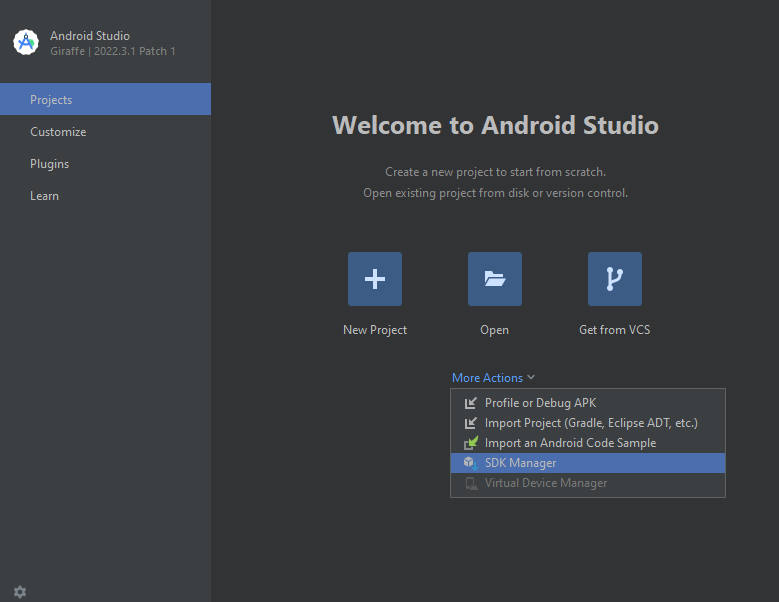
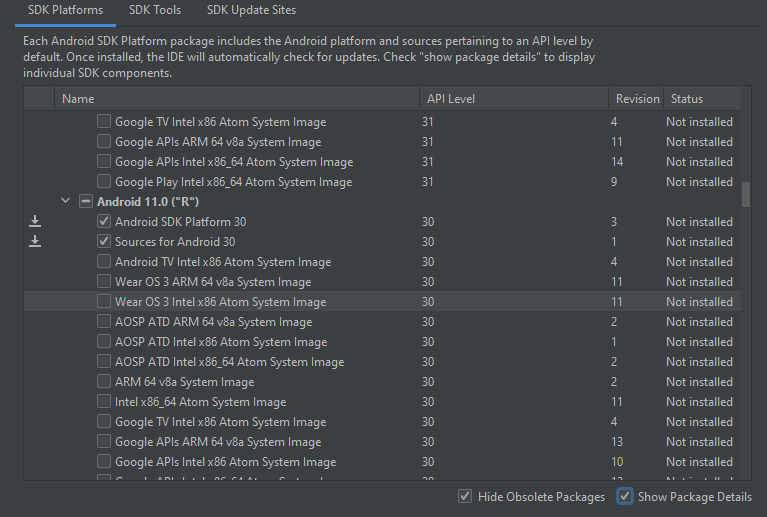
**Preludio**

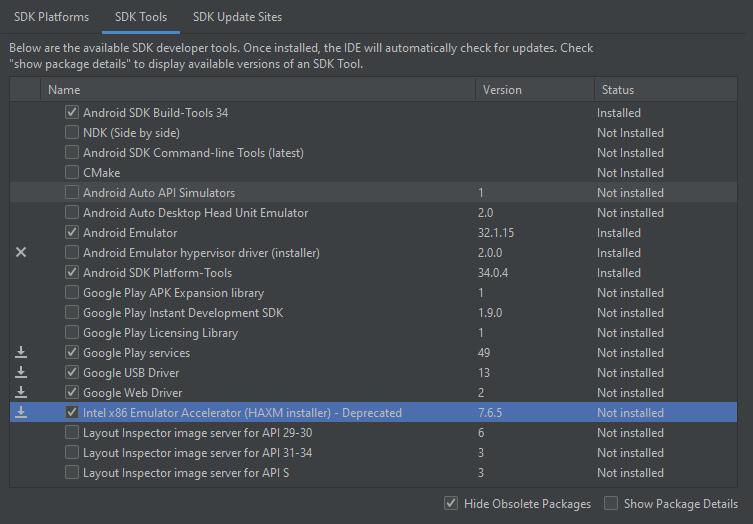
Antes de comenzar, seleccionaremos e instalaremos las extensiones necesarias en Android Studio. Para hacerlo, vamos a la opción "More Actions" (o "Más Acciones" en español) y seleccionamos "SDK Manager".



En primer lugar, instalaremos Android 11.0 ("R"), centrándonos especialmente en "Android SDK Platform 30" y "Sources for Android 30". Esto se debe a que Android 11.0 (API nivel 30) es una de las versiones más importantes.



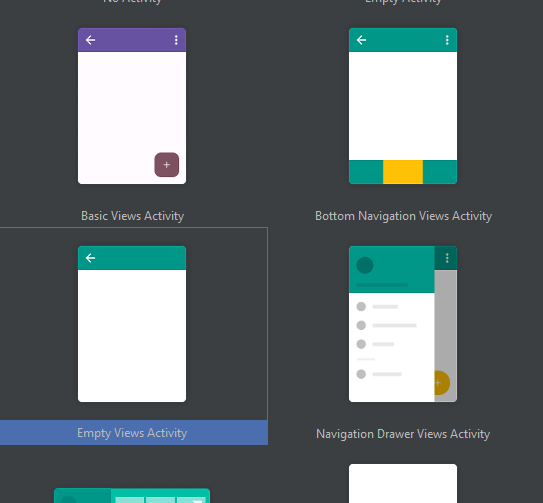
Luego, vamos a "SDK Tools" y desactivamos "Android Emulator hypervisor driver" y activamos "Google Play services", "Google USB Driver", "Google Web Driver" y "Intel x86 Emulator Accelerator...". El resto de opciones pueden mantenerse como están.

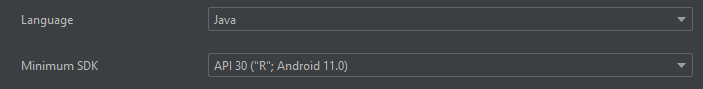


Con estos pasos, estamos listos para crear un nuevo proyecto.

**Primeros pasos**

El primer proyecto que crearemos será un "Empty Views Activity". Seleccionamos Java como lenguaje, API 30 y hacemos clic en "Finish" para iniciar la creación del proyecto.





Una vez dentro del proyecto, veremos su estructura jerárquica.

**Jerarquía de un proyecto Android**

Dentro de la jerarquía de un proyecto Android, a la izquierda, podemos encontrar la aplicación con todas sus carpetas y los archivos de configuración en "Gradle Scripts". Estos archivos están escritos en el lenguaje de construcción Gradle y se utilizan para definir cómo se construirá y compilará el proyecto de Android, similar al Maven de NetBeans. Entre los archivos, el "build.gradle.kts" almacena las dependencias.

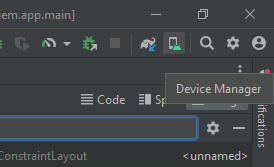
Volviendo a la carpeta de la aplicación, en "src > res" encontramos el archivo "AndroidManifest.xml", que describe la secuencia de métodos y el orden en el que se ejecutarán. El primero de ellos es el método "onCreate", equivalente al "main" de NetBeans. También tenemos una carpeta llamada "test" donde se realizan pruebas, que no abordaremos por el momento. Luego está la carpeta "res", donde se almacenan recursos como imágenes y sonidos. Por último, en "layout", encontramos el archivo "activity\_main.xml", que técnicamente es similar a la vista en NetBeans.

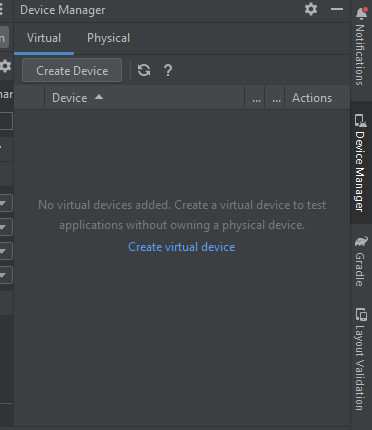
Otro archivo importante es "string.xml", que almacena todo el texto de la aplicación y es útil para realizar traducciones.

**Empecemos con activity\_main.xml**

La vista "activity\_main.xml" tiene, por defecto, "Constraint Layout", pero para evitar problemas futuros, cambiaremos a "Linear Layout". Para hacerlo, hacemos clic derecho y seleccionamos "Convert View". También podemos cambiar la orientación de vertical a horizontal buscando la propiedad "orientation".

Antes de continuar con los componentes, hablaremos sobre la creación de una máquina virtual. Para ello, vamos a "Device Manager", ubicado en la parte superior derecha, y luego a "Create Device". Elegimos el dispositivo, como Pixel 5 o Pixel 6, y seleccionamos la versión con "R" o "S" si es necesario. Para ejecutarlo, hacemos clic en el botón de reproducción. Se recomienda cambiar el "View Mode" a "Window" para una mejor experiencia.





**Componentes (**[**API**](https://developer.android.com/guide/topics/manifest/uses-sdk-element?hl=es-419)**)**

Comencemos a explorar los componentes de Android.

**View**

Los componentes y elementos en Android Studio se conocen como "View" o "vista".

**TextView**

El "TextView" es un componente de texto básico que hereda de "View".

**Button**

El "Button" o botón también hereda de "TextView" y representa un elemento de texto que funciona como botón.

**EditText**

El "EditText" es un campo de texto editable en la aplicación y hereda de "TextView". A veces, se le llama "PlainText".

**Métodos**

* *Compact Activity*: Este método se utiliza varias veces en el desarrollo de aplicaciones Android.
* *FindViewById*: Este método se utiliza para buscar elementos por su ID, que son identificadores comúnmente utilizados. Android genera automáticamente una clase de constantes que almacena estos IDs.
* *Métodos de componentes*: Para seleccionar métodos de un componente específico, debemos importar la clase con "Alt + Intro" y seleccionar el componente deseado. De lo contrario, Android Studio seleccionará automáticamente los métodos de "View".

**Depuración**

La depuración es el proceso de identificar, analizar y solucionar problemas en un programa de software. Los desarrolladores utilizan mensajes de depuración para obtener información sobre lo que está sucediendo dentro del programa mientras se ejecuta.

Los mensajes de depuración aparecerán en la pestaña de Logcat, una pantalla que se muestra en la parte inferior de la ventana. Si no está visible, puedes activarla de las siguientes maneras:

* Haciendo clic en el icono de "Android Profiler" en la parte inferior derecha y seleccionando "Logcat" en la pestaña que se abre.
* Utilizando la combinación de teclas "Alt + 6" (en Windows/Linux) o "Cmd + 6" (en macOS) para alternar la vista de Logcat.

En la pestaña de Logcat, puedes aplicar filtros, seleccionar entre dispositivos y aplicaciones, e incluso interactuar con los mensajes para obtener más información. Para generar mensajes de depuración, es importante utilizar la función “Log”, como “Log.d()”, “Log.e()”, “Log.i()”, etc.

**Funciones de Botón**

Vamos a ver cómo asignar una acción a un botón cuando se pulsa. Primero, guardamos el botón en una variable “Button” utilizando el ID que le asignamos en el diseño. Esto se hace así:

Button boton = findViewById(R.id.boton);

Una vez que tenemos el botón en una variable, podemos cambiar sus propiedades, como el texto dentro de él:

boton.setText("Hola Mundo");

Luego, para indicarle que realice una acción cuando se presione, necesitaremos utilizar un "listener". Los listeners son interfaces que se utilizan para manejar eventos. Hay muchos tipos diferentes de listeners, y en este caso, necesitamos uno para manejar el evento de hacer clic en el botón, así que utilizamos “setOnClickListener”.

Existen varias formas de implementar este método:

1. **Clase Externa**:

Crear una clase externa para manejar el evento de clic del botón. Esto implica crear una clase separada que implementa “View.OnClickListener" y, a continuación, asignar una instancia de esta clase como el “OnClickListener” del botón. Aunque es menos común, se ve de la siguiente manera:

public class ClasePulsacion implements View.OnClickListener {

@Override

public void onClick(View v) {

// Código que se ejecutará cuando se haga clic en el botón.

Log.d(TAG, "Mensaje de depuración");

}

}

// En la actividad MainActivity:

Button boton = findViewById(R.id.boton);

ClasePulsacion pulsacion = new ClasePulsacion();

boton.setOnClickListener(pulsacion);

1. **Implementar el Método en los Paréntesis:**

Implementar el método onClick directamente dentro de los paréntesis del “setOnClickListener". Esto se hace mediante una clase anónima:

Button boton = findViewById(R.id.boton);

boton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

// Código que se ejecutará cuando se haga clic en el botón.

Log.d(TAG, "Mensaje de depuración");

}

});

1. **Expresión Lambda:**

Utilizar una expresión lambda, que es una forma más simplificada de implementar el “OnClickListener” cuando solo se necesita un método, como en este caso:

Button boton = findViewById(R.id.boton);

boton.setOnClickListener(v -> {

// Código que se ejecutará cuando se haga clic en el botón.

Log.d(TAG, "Mensaje de depuración");

});

**Kotlin**

Kotlin es un lenguaje derivado de Java que busca hacer la programación más sencilla. Introduce características como la prevención de NullPointerException mediante la prohibición de la creación de objetos no inicializados (indicados con ?), eliminación de puntos y comas, y castings más concisos entre otras cosas.

**Spinner**

Un Spinner es un desplegable que puede trabajar con datos de manera estática, lo que significa que estos datos no cambian durante la ejecución de la aplicación, a menos que los manipulemos en tiempo de ejecución.

Puedes llenarlo de dos formas diferentes:

Crear un array de cadenas en los recursos XML, como se muestra a continuación:

<string-array name="ciudades">

<item>Barcelona</item>

<item>Madrid</item>

<item>Valencia</item>

</string-array>

Luego, puedes enlazarlo al Spinner buscando la propiedad "entries" en la vista de diseño y seleccionando el array con @array/.

Llenarlo en tiempo de ejecución, generalmente dentro del método onCreate del MainActivity. Por lo general, se crea un método separado, como "inicializarSpinner". Primero, creas un array o ArrayList con los datos que deseas mostrar. Luego, obtienes una referencia al Spinner y creas un adaptador, como ArrayAdapter, que requiere el contexto, un diseño y el array que creaste. El código se vería así:

ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_spinner\_item, array);

spinner.setAdapter(adapter);

Puedes personalizar la acción que ocurre cuando se selecciona un elemento del Spinner utilizando setOnItemSelectedListener(). Por ejemplo:

spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> parentView, View selectedItemView, int position, long id) {

// Código que se ejecutará cuando se seleccione un elemento

}

@Override

public void onNothingSelected(AdapterView<?> parentView) {

// Código que se ejecutará cuando no se seleccione ningún elemento

}

});

**RadioGroup y RadioButton**

Un RadioGroup es un conjunto de botones de opción (RadioButton) donde solo se puede seleccionar uno a la vez. Los botones de opción se encuentran dentro del RadioGroup.

Puedes utilizar el método setOnCheckedChangeListener para saber cuál de los RadioButton ha sido seleccionado en el grupo. Un ejemplo sería:

RadioGroup radioGroup = findViewById(R.id.radioGroup);

radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(RadioGroup group, int checkedId) {

RadioButton radioButton = findViewById(checkedId);

if (radioButton != null) {

String opcionSeleccionada = radioButton.getText().toString();

// Realiza acciones basadas en la opción seleccionada

}

}

});

**Toast**

Por último, mencionar el uso de Toast, que se utiliza para mostrar un mensaje breve en la pantalla. Puedes utilizarlo para proporcionar información o retroalimentación al usuario.