



android

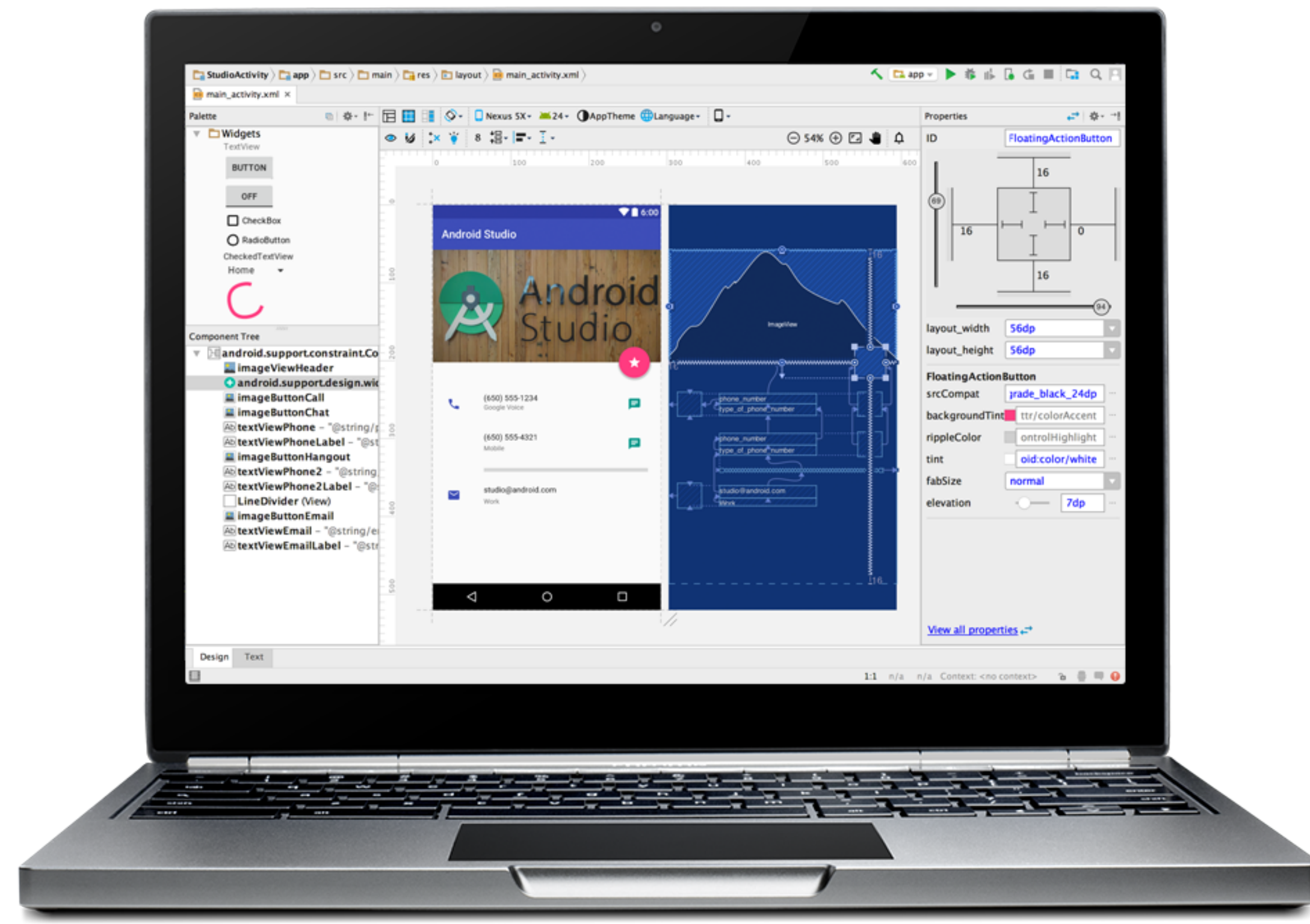
Desarrollo de
Aplicaciones Móviles

Ing. Inf. Miguel Ángel Hernández Muñoz

Objetivo

Con este curso el usuario aprenderá a desarrollar aplicaciones móviles para el sistema operativo Android, mediante las herramientas y el entorno de desarrollo Android Studio.





El IDE oficial para Android

Entorno de Desarrollo

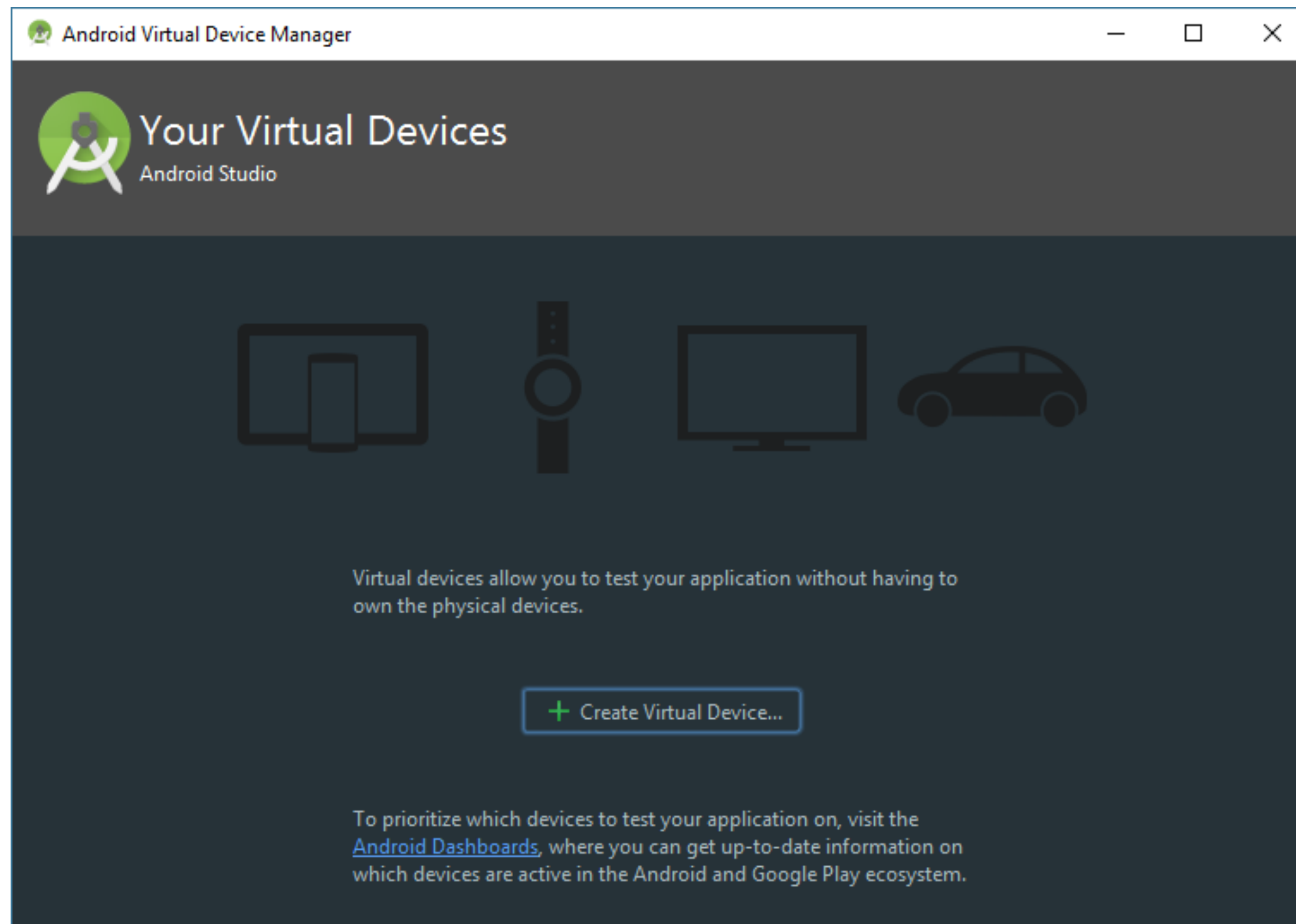
ANDROID STUDIO

Android Studio proporciona las herramientas más rápidas para la creación de aplicaciones en todos los tipos de dispositivos Android.

La edición de códigos de primer nivel, la depuración, las herramientas de rendimiento, un sistema de compilación flexible y un sistema instantáneo de compilación e implementación te permiten concentrarte en la creación de aplicaciones únicas y de alta calidad.

Dispositivos Virtuales

Android Virtual Device Manager



El AVD Manager te permite definir las características de un teléfono, tablet, wearable o tv de Android para poder simularlo en el emulador. Así mediante el administrador de dispositivos de Android puedes crear diferentes dispositivos virtuales.

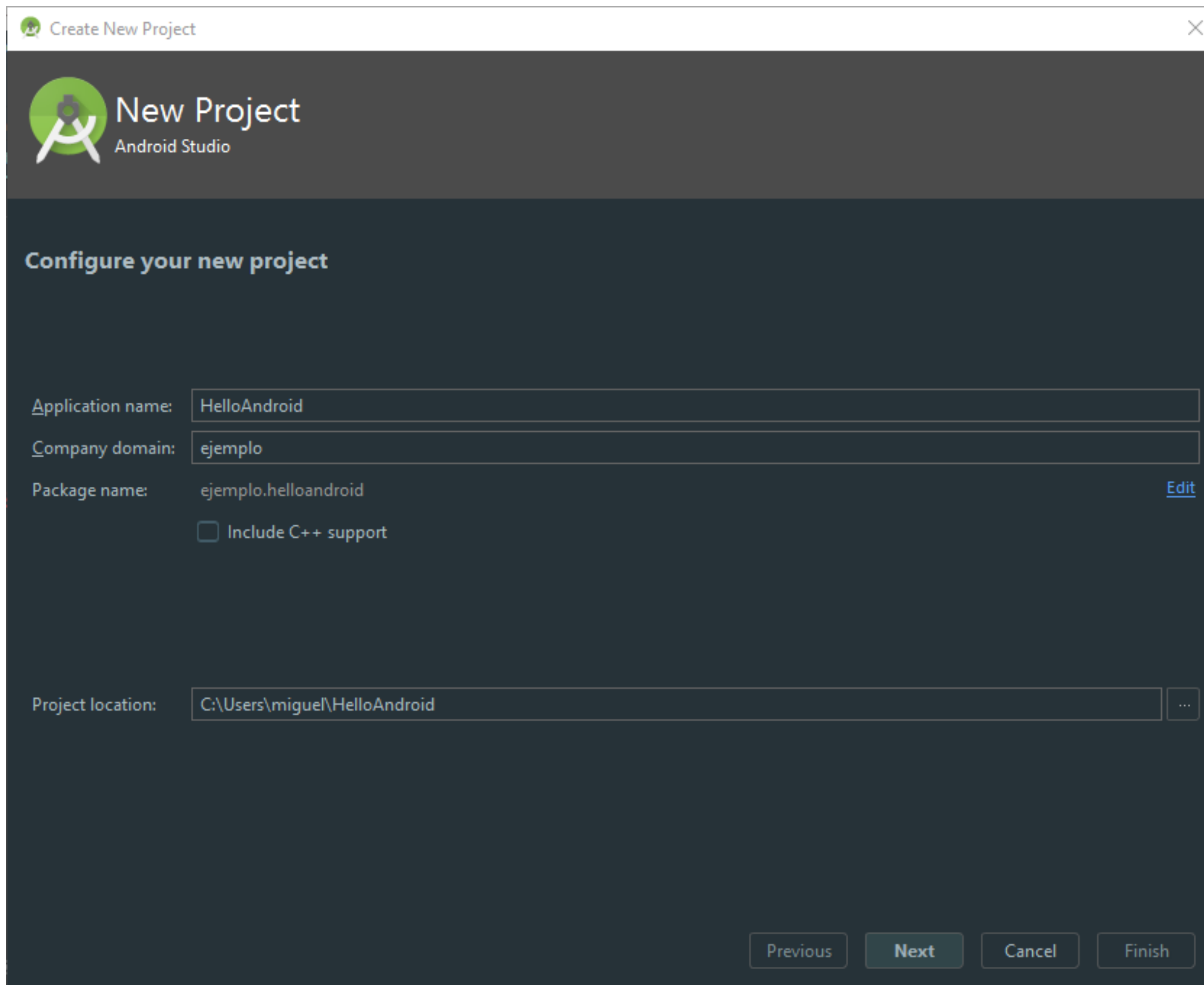
Dispositivos Físicos

Para poder correr las aplicaciones que se realizan en Android Studio en un dispositivo físico es necesario activar las opciones de desarrollador y la depuración de Android.




Hello Android

Vamos a crear nuestro primer proyecto en Android Studio, el cual es un simple Hello World que es compilable y ejecutable en dispositivos Android o emulador.



Create New Project

 **New Project**
Android Studio

Configure your new project

Application name: HelloAndroid

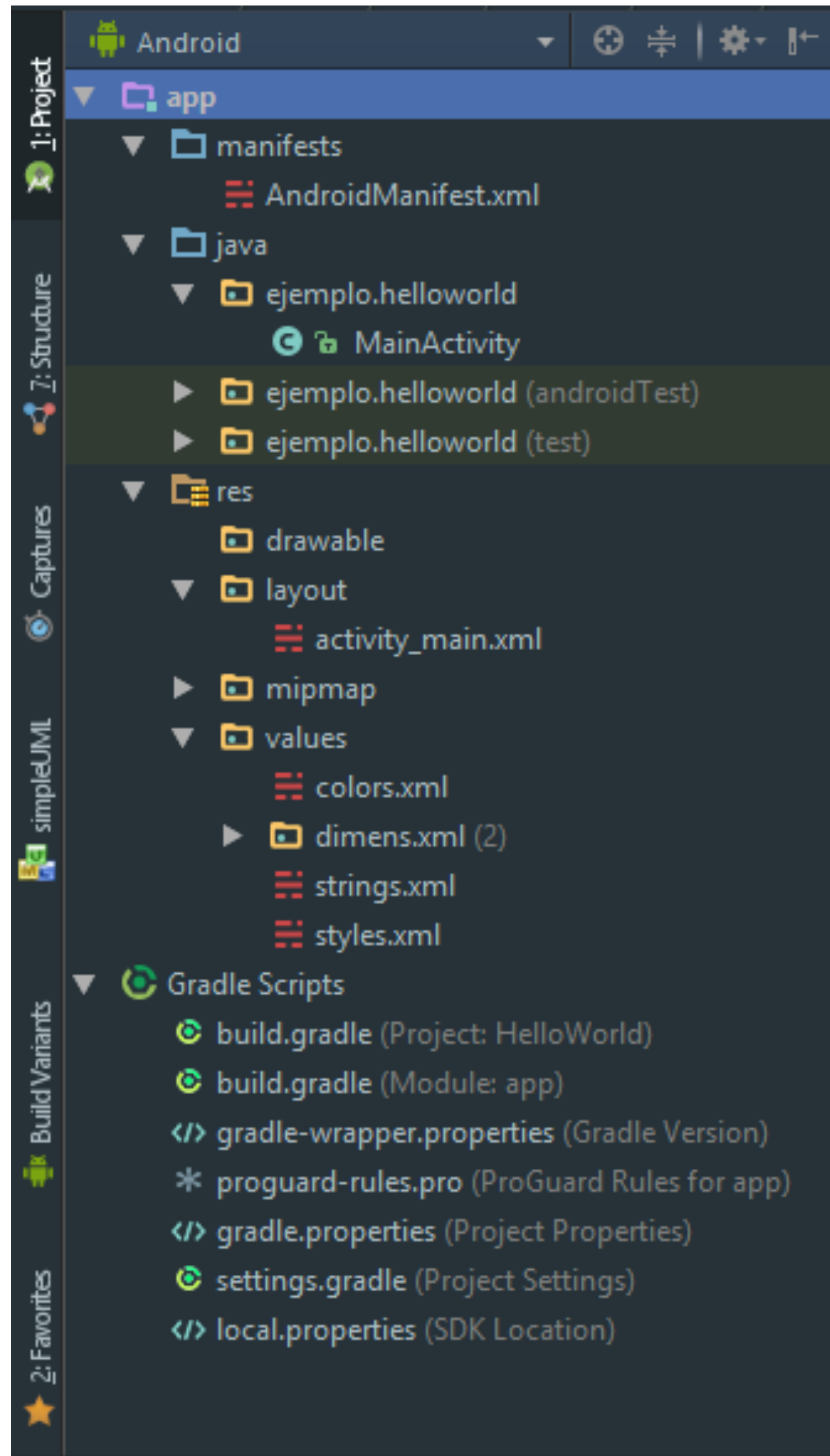
Company domain: ejemplo

Package name: ejemplo.helloandroid [Edit](#)

☐ Include C++ support

Project location: C:\Users\miguel\HelloAndroid ...

Previous Next Cancel Finish



Estructura del Proyecto

1. Carpeta manifests: Contiene el archivo del manifiesto de la aplicación.
2. Carpeta java: Contiene los archivos de código fuente de Java separados por nombre de paquetes, incluido el código de prueba JUnit.
3. Carpeta res: Contiene todos los recursos sin código, como diseños XML, strings de IU e imágenes de mapa de bits, divididos en subdirectorios pertinentes

Estructura del Proyecto

Carpeta manifests:

Todas las aplicaciones deben tener un archivo `AndroidManifest.xml` (con ese nombre exacto) en el directorio raíz. El archivo de manifiesto proporciona información esencial sobre tu aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la app.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ejemplo.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```


Estructura del Proyecto

```
res/  
    drawable/  
        graphic.png  
    layout/  
        main.xml  
        info.xml  
    mipmap/  
        icon.png  
    values/  
        strings.xml
```

Carpeta res:

1. Carpeta drawable: Contiene los recursos gráficos de la aplicación.
2. Carpeta layout: Contiene los diseños de las pantallas en formato xml.
3. Carpeta mipmap: Contiene los iconos a utilizar en la aplicación.
4. Carpeta values: Contiene diversos ficheros como por ejemplo cadenas, colores o estilos.

Java



Android utiliza como lenguaje de programación Java, y admite todas las funciones del lenguaje Java 7 y un subconjunto de funciones del lenguaje Java 8 que varían según la versión de la plataforma.

Las clases en Java son plantillas para la creación de objetos, es lo que se conoce como programación orientada a objetos. Una clase en Java puede tener cualquier número de métodos para acceder o modificar el comportamiento de dicha clase.

Cada clase tiene un constructor. Si no escribimos explícitamente un constructor para una clase el compilador de Java genera un constructor predeterminado para esa clase.

Un objeto es creado a partir de una clase y básicamente existen tres pasos al crear un objeto de una clase:

Declarar: Debemos declarar una variable con su nombre y con el tipo de objeto que va a contener.

Instanciar: La palabra clave `new` se utiliza para crear el objeto.

Inicialización: La palabra clave `new` va seguida de una llamada a un constructor. Esta llamada inicializa el nuevo objeto.

Hello World

1
↓
public class HelloWorld{
 public static void main(String[] args){
 System.out.println("Hola mundo");
 }
}

1.- class: Es una palabra reservada de Java y se utiliza para definir la implementación de un Objeto. Una clase es un molde o modelo utilizado para crear instancias de Objetos.

Modificadores de Acceso

2



```
public class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hola mundo");  
    }  
}
```

2.- La palabra reservada `public` es un modificador de acceso. Los modificadores de acceso se utilizan para controlar el acceso a atributos, métodos o clases, en Java existen 4 tipos:

- `public`: Te permite acceder al elemento desde cualquier clase.
- `private`: Sólo puede ser accedido desde su propia clase.
- `protected`: Puede ser accedido desde su clase o por una subclase de su clase.
- `default`: También conocido como `package-private` permite acceso desde su propia clase o por otras dentro del mismo paquete.

Nombre de Clase

3



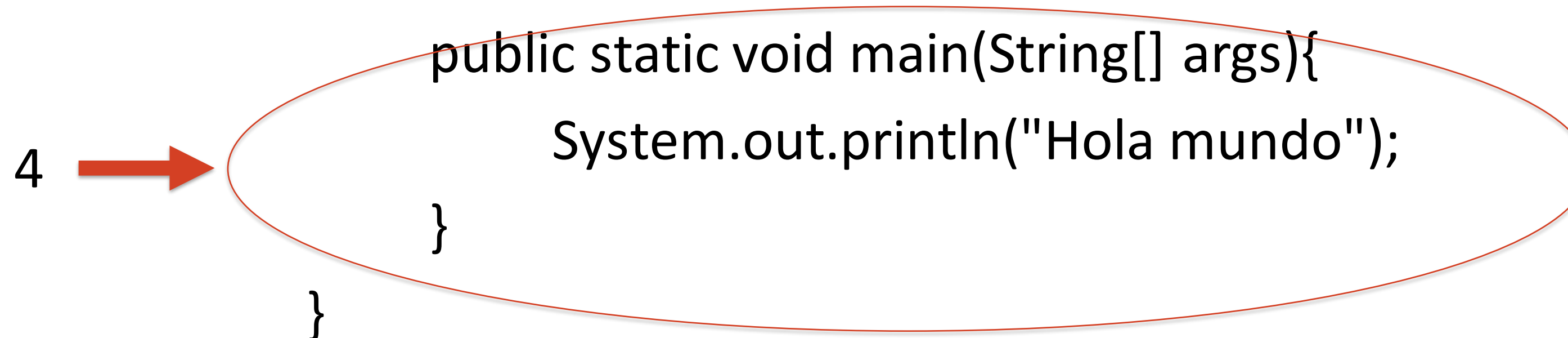
```
public class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hola mundo");  
    }  
}
```

3.- El nombre se utiliza para identificar la clase, algunas reglas para nombrar las clases son las siguientes:

- El nombre debe de ser sustantivo.
- La primera letra de cada palabra debe ser capitalizada.
- Utiliza palabras completas descriptivas, evita el uso de abreviaciones.

Método main


```
public class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hola mundo");  
    }  
}
```

A red oval highlights the main method block within the HelloWorld class. A red arrow points from the number '4' to the start of the main method's opening curly brace.

4.- El método main o método principal es el punto de partida de tu programa, este debe ser declarado obligatoriamente en alguna de tus clases, para poder indicarle en dónde va a empezar la ejecución. Cada declaración dentro del método main es ejecutada en orden hasta llegar al final de este.

static


```
public class HelloWorld{  
    public static void main(String[] args){  
5      System.out.println("Hola mundo");  
    }  
}
```



5.- La palabra reservada static, le indica a la clase que el método o variable declarado es estático para todas las instancias de dicha clase, es decir no importa la cantidad de veces que instancias una clase el método o variable va a ser compartida por todas las instancias.

void

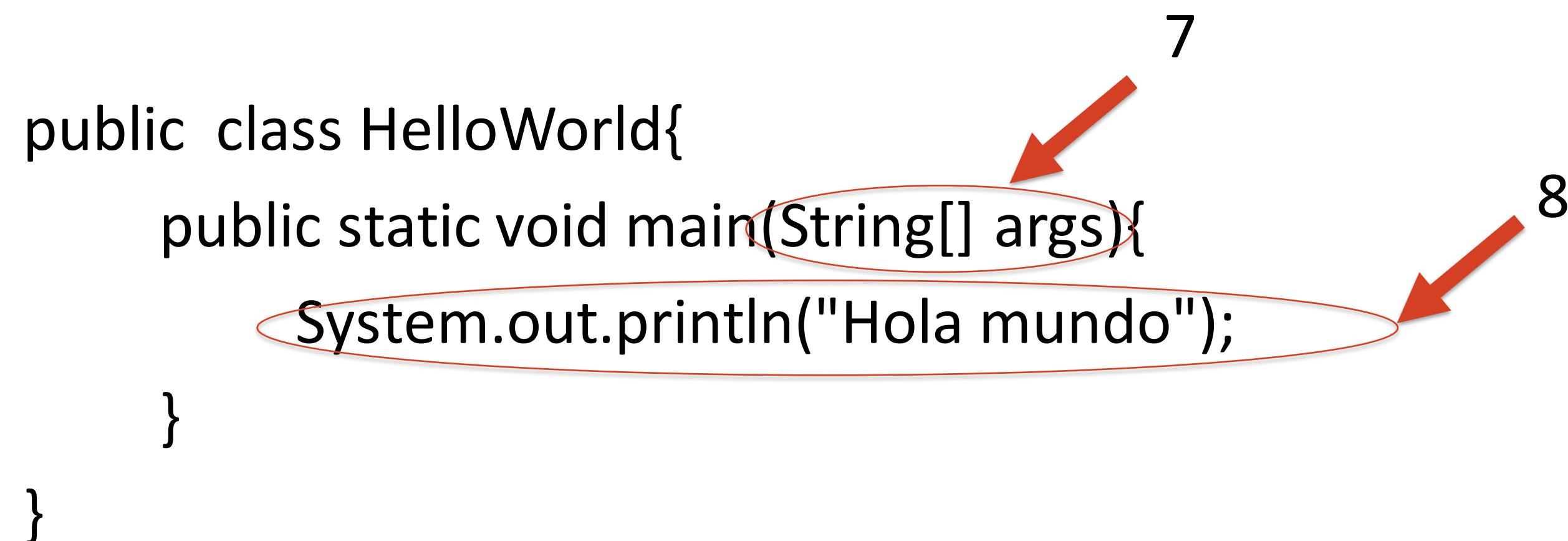
```
public class HelloWorld{  
    public static void main(String[] args){  
6      System.out.println("Hola mundo");  
    }  
}
```



6.- La palabra reservada void, le indica al método que al terminar su ejecución no va a retornar ningún valor, los métodos que creemos nos permiten retornar diferentes valores como retroalimentación del proceso que realicen.

Argumentos

```
public class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hola mundo");  
    }  
}
```



- 7.- `String[] args` : Es un arreglo de cadenas, se utiliza cuando al programa se le pasan algunos parámetros iniciales, mediante línea de comandos.
- 8.- `System.out.print("")` : Es una clase que nos permite realizar impresiones en la pantalla de la consola.

Las variables son utilizadas para almacenar información en memoria, en Java no todas las variables son objetos a estas variables se les conoce como primitivas, a continuación las listamos:

- byte (números, 1 byte)
- short (números, 2 bytes)
- int (números, 4 bytes)
- long (números, 8 bytes)
- float (número flotante, 4 bytes)
- double (número flotante, 8 bytes)
- char (un carácter, 2 bytes)
- boolean (true o false, 1 byte)

Arreglos

Un arreglo es un conjunto de variables, referenciadas con un único nombre adicionalmente cuentan con un número índice, dentro del arreglo cada elemento es una variable.

Por ejemplo podemos crear un arreglo de 10 números de la siguiente manera:

```
char[] vocales = new char[5];
```



Representación visual de un arreglo.

ArrayList

La clase ArrayList es una clase que nos permite almacenar datos en memoria de forma similar a los arreglos, con la ventaja de que los elementos que almacena se agregan de forma dinámica, es decir, que no es necesario declarar su tamaño inicialmente como pasa con los arreglos. Declaramos un ArrayList de la manera siguiente:

```
ArrayList<String> vocales = new ArrayList<>();
```

ArrayList

La clase ArrayList provee una serie de herramientas para realizar diversas acciones con los datos, por ejemplo para agregar un nuevo elemento a la lista lo hacemos con el método add(), si queremos agregar una nueva vocal a nuestra lista de vocales sería de la siguiente forma:

```
vocales.add("a");  
vocales.add("e");  
vocales.add("i");  
vocales.add("o");  
vocales.add("u");
```

Para obtener un elemento de la lista de vocales escribimos lo siguiente:

```
vocales.get(0);
```

Esto nos regresaría el elemento "a";

ArrayList

Para obtener el tamaño del ArrayList lo hacemos mediante el método `size()` de la siguiente forma:

```
vocales.size();
```

Lo cual retorna un entero con el tamaño del ArrayList en nuestro caso nos retorna 5, por que agregamos cinco elementos.

```
vocales.get(0);
```

Esto nos regresaría el elemento “a”;

ArrayList

Otra de las operaciones que se pueden realizar con un ArrayList es asignar nuevos valores a índices existentes, si decidimos cambiar el valor de un elemento se hace de la siguiente forma:

```
vocales.set(0,"x");
```

También podemos eliminar un elemento del ArrayList, esto se hace con el método `remove()`, así para eliminar una vocal de la lista queda de la siguiente forma:

```
vocales.remove(0);
```

Por último podemos limpiar completamente nuestra lista con el siguiente comando:

```
vocales.clear();
```

HashMap

Un HashMap es una clase que almacena datos asociando una llave a un valor, los HashMap no permiten llaves duplicadas, cada llave tiene que estar asociada a un valor como máximo, si agregas una llave que ya existe sobrescribe el valor de la llave anterior, solo permite agregar valores de tipo objeto que no sean primitivos.

Para crear un nuevo mapa escribimos lo siguiente:

```
HashMap<String,ArrayList<String>> animales = new HashMap<>();
```

HashMap

La clase HashMap provee una serie de herramientas para realizar diversas acciones con los datos, por ejemplo para agregar un nuevo elemento al mapa lo hacemos con el método put(), si queremos agregar una lista de animales al mapa primero tenemos que crear una lista de animales de la siguiente forma:

```
ArrayList<String> carnivoros = new ArrayList<>();  
ArrayList<String> herviboros = new ArrayList<>();
```

```
    carnivoros.add("Leon");  
    carnivoros.add("Oso");  
    carnivoros.add("Cocodrilo");  
    herviboros.add("Vaca");  
    herviboros.add("Conejo");  
    herviboros.add("Caballo");
```

Escribimos lo siguiente para agregar las listas al mapa:

```
animales.put("carnivoros",carnivoros);  
animales.put("herviboros",herviboros);
```

HashMap

Para obtener un elemento del mapa escribimos lo siguiente:

```
animales.get("carnivoros");
```

Esto nos regresaría un ArrayList con los elementos de la lista de carnívoros.

Para obtener el tamaño del mapa lo hacemos mediante el método `size()` de la siguiente forma:

```
animales.size();
```

Lo cual retorna un entero con el tamaño del HashMap en nuestro caso nos retorna 2, por que agregamos dos elementos es decir dos ArrayList.

HashMap

Otra de las operaciones que se pueden realizar con un HashMap es asignar nuevos valores a llaves existentes, si decidimos cambiar el valor de un elemento se hace de la siguiente forma:

```
animales.put("carnivoros",nuevosCarnivoros);
```

Siendo nuevosCarnivoros un ArrayList con nuevos elementos.

También podemos eliminar un elemento del HashMap, esto se hace con el método `remove()`, así para eliminar una elemento del mapa queda de la siguiente forma:

```
animales.remove("carnivoros");
```


HashMap

Para limpiar completamente nuestra mapa lo hacemos con el siguiente comando:

```
animales.clear();
```

Podemos checar si el mapa esta vacío con el método isEmpty() lo cual nos retorna un booleano con true si no tiene valores o false si contiene elementos.

```
animales.isEmpty();
```

En nuestro caso como acabamos de limpiar el mapa, nos va a retornar true.

Condicionales y Operadores

Las estructuras condicionales nos permiten ejecutar una serie de instrucciones si cumple una determinada condición que nosotros le indiquemos. Es importante recordar que la condición debe dar un resultado booleano.

if: le indicamos una condición si esta es verdadera se ejecuta, sino no se ejecuta las instrucciones de dentro. Se suele traducir como “Si se cumple esta condición haz esto”.

if – else: es como el anterior solo que después de cerrarse la llave de if, se añade else sin indicarle ninguna condición. Esto sirve para que si la condición del if no es verdadera, ejecute otras instrucciones que estarán dentro de else. Se suele traducir como “Si se cumple esta condición haz esto y sino haz esto”

Condicional if

Para declara un condicional if-else escribimos lo siguiente:

```
if(17>18){ //mayor que 18
    System.out.println("Eres mayor de edad");
}else{
    System.out.println("Eres menor de edad");
}
```

OPERADOR

==

!=

<, <=, >, >=

&&

||

!

DESCRIPCIÓN

Es igual

Es distinto

Menor, menor o igual, mayor, mayor o igual

Operador and (y)

Operador or (o)

Operador not (no)

Condicional Switch

El condicional de selección switch es una estructura de control multiselección, es decir permite tomas de decisión multiples que tradicionalmente se harían con múltiples if-else o con if-else anidados. Podemos declarar un switch de la siguiente manera:

```
int edad = 39;

switch(edad){
    case 18:
        System.out.println("Tienes 18");
        break;
    case 19:
        System.out.println("Tienes 19");
        break;
}
```

Los ciclos o también conocidos como bucles, nos permiten repetir una o varias instrucciones cuantas veces sea necesario. Un ciclo nos ayuda a llevar a cabo una tarea repetitiva en una cantidad de líneas muy pequeña y de forma prácticamente automática.

Existen diferentes tipos de ciclos o bucles en Java, cada uno tiene una utilidad para casos específicos , tenemos a nuestra disposición los siguientes tipos de ciclos en Java:

- For
- For each
- While
- Do While

Ciclo for

El ciclo for nos provee de una manera compacta de recorrer un rango de valores, la forma general de un ciclo for puede ser expresada de la siguiente manera:

```
for (inicialización; terminación; incremento) {  
    sentencia(s) a ejecutar.  
}
```

- La expresión de inicialización inicia el ciclo, es ejecutado una sola ocasión cuando el ciclo comienza.
- Cuando la expresión de terminación es evaluada a false, el ciclo finaliza.
- La expresión de incremento es invocada después de cada iteración del ciclo, se puede incrementar o decrementar.

Ciclo for each

Una de las cosas que incorporó Java 5 fue el bucle for-each en Java. Esta estructura nos permite recorrer un arreglo de elementos de una forma sencilla. Es conocido también como el ciclo for mejorado, debido a que simplifica la manera de recorrer un arreglo.

```
for (TipoBase variable: ArrayDeTiposBase) {  
    sentencia(s) a ejecutar.  
}
```

- La expresión TipoBase se refiere al tipo de los elementos del arreglo.
- La expresión variable es mediante la cual tenemos acceso a cada elemento.
- La expresión ArrayDeTiposBase se refiere al arreglo de todos los elementos.

Ciclo while

La sentencia while ejecuta continuamente un bloque de sentencias mientras que se cumpla cierta condición, su sintaxis puede ser expresada de la manera siguiente:

```
while (expresión) {  
    sentencias(s) a ejecutar.  
}
```

El ciclo while hace la evaluación de la expresión en la parte superior.

Ciclo do while

El ciclo do while es muy similar al ciclo while sólo que el do while realiza la evaluación de la expresión en la parte inferior del código, permitiendo esto que lo que se encuentra dentro del bloque sea ejecutado por lo menos una vez, la sintaxis del ciclo do while es la siguiente:

```
do {  
    sentencia(s) a ejecutar.  
} while (expresión);
```